On the Regularity of Binoid Languages:
A Comparative Approach

Zoltán L. Németh

University of Szeged, Inst. of Informatics
P.O.B. 652, 6701 Szeged, Hungary
zlnemeth@inf.u-szeged.hu

Abstract. A binoid is a set with two associative operations where the operations share a common identity element, while subsets of free binoids are called binoid languages. Two independent studies concerning the regularity of binoid languages were done by Hashiguchi et al. who used term representations and monoid automata and by Ésik and Németh who employed parenthesizing automata for the acceptance of binoid languages. The aim of this paper is to relate these two approaches, and to show how the monoid approach of Hashiguchi et al. can be extended to the algebraically recognizable class of binoid languages.

1 Introduction

A *bisemigroup* is an algebra equipped with two independent associative operations. The two operations will be called the *horizontal product* and the *vertical product*, and will be denoted by $\bullet$ and $\circ$, respectively. Following Hashiguchi et al. [11], if both operations have an identity then we get a *bimonoid*. Moreover, if the identities are the same, the resulting structure is called a *binoid*.

For each alphabet Σ consider the free binoid over Σ, denoted by $\Sigma^*(\bullet, \circ)$. The subsets of $\Sigma^*(\bullet, \circ)$ have various names like 'binoid languages' or 'B-languages' [11, 13, 14], 'bi-languages' [3, 4] and 'sp-biposet languages' [6, 20, 21]. Here we will call them 'binoid languages', while the elements of binoid languages will be called biwords as in [3].

In this paper we will concentrate on the general theory, but we believe that the concept of binoid languages is sufficiently general to have some practical applications as well. The reader can review the study of Hashiguchi et al. on bicodes [12] and on a modified RSA cryptosystem based on bicodes [10].

In the future biwords may also be used in modeling systems like sp-posets, which serve as models of modularly constructed concurrent systems. Sp-posets represent the elements of free algebras where an associative operation and an associative and commutative operation are defined. They have been studied extensively by Lodaya and Weil [17–19], and also by Kuske [16].

Binoid languages are also closely related to picture languages [8], texts [5, 15], and visibly pushdown and nested word languages [1, 2]. In [3] Dolinka demonstrated that picture languages and binoid languages satisfy the same identities (for the operations of union, the two products, the two (Kleene) iterations of the two products and some constants). See [4] as well for more details about the axiomatization of the equational theory of binoid languages.

To date there are two independent studies of automata on biwords. The first one was done by Hashiguchi et al. [11, 13, 14], which may also be called the *monoid approach*. The second one on parenthesizing automata was done by Ésik and Németh [6, 7, 20, 21].

Hashiguchi et al. regard biwords as their *term representation*[1]. They are words over the extended alphabet $E(\Sigma) = \Sigma \cup \{\langle, \rangle, \bullet, \circ\}$, where $\langle$ and $\rangle$ are parenthesis symbols. Thus ordinary finite automata (from now on *monoid automata*) can be used to define regular binoid languages. More precisely, they defined two kinds of acceptance by monoid automata: the free binoid mode and the free monoid mode. In the case of *free monoid mode* acceptance, given any word $x \in E(\Sigma)^*$ the automaton decides whether x is a valid term representation of a biword in the accepted language. In the case of *free binoid mode* acceptance, the inputs of the automaton just come from the restricted set of valid term representations, and the automaton only decides the question of whether the biword represented by the input term belongs to the accepted language or not. Let $\mathsf{Reg}^{\mathsf{FM}}$ (resp. $\mathsf{Reg}^{\mathsf{FB}}$) denote the class of binoid languages that can be accepted in the free monoid (resp. binoid) mode. Earlier it was shown that $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}^{\mathsf{FB}}$ [11]. The main result of [13] and [14] can also be expressed as $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{BRat}$, where BRat stands for the class of *birational languages*. They are those binoid languages that can be obtained from the finite binoid languages by applying the operations of union, horizontal and vertical products, horizontal iteration and vertical iteration. An obvious advantage of the monoid approach as against parenthesizing automata is that one is not forced to use automata to describe regular binoid languages. Rather, any equivalent characterization of regular word languages (e.g. regular expressions or MSO-formulas) can be used instead.

As mentioned above, the other approach by Ésik and Németh is based on *parenthesizing automata* (PA for short). These devices process biwords based on their hierarchical structures by using indexed parentheses in the transitions. Their expressive power is the same as algebraic recognizability (defined by homomorphisms and finite binoids) and monadic second order definability (based on the sp-biposet representation). For more details see [6] and the survey [22] by Weil on the general concept of recognizability. Now let Reg_i denote those binoid languages that can be accepted by parenthesizing automata with at most i pairs of parentheses symbols. In [20] it was shown that these languages form a strict hierarchy, i.e. $\mathsf{Reg}_0 \subsetneq \mathsf{Reg}_1 \subsetneq \mathsf{Reg}_2 \subsetneq \ldots$

The aim of this paper is to relate the above two approaches of regularity. A comparison can be expressed as: $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{Reg}_1 \cap \mathsf{BD}$, and $\mathsf{Reg}^{\mathsf{FB}} = \mathsf{Reg}_1'$. Here Reg_1' is a slightly modified version of Reg_1, and BD stands for the class of bounded depth binoid languages, i.e. those languages that have a uniform bound on the number of nested parentheses in their elements. So monoid automata even in the free binoid mode are less expressive than parenthesizing automata

[1] In [11, 13, 14], term representations are called *s-forms*.

with two or more parenthesis symbols. Next we extend the free binoid mode of monoid automata using so-called i-term representations of binoid languages for all $i \geq 0$. The i-term representations of binoid languages are word languages over the alphabet $E_i(\Sigma) = \Sigma \cup \{\langle_1, \rangle_1, \ldots, \langle_i, \rangle_i, \bullet, \circ\}$, so one can use i different pairs of parentheses to describe biwords. Our main result shows that this new acceptance mode of monoid automata corresponds to PA with i pairs of parentheses. Hence the class of recognizable languages can also be captured by the monoid approach of Hashiguchi et al.

2 Binoids and biwords

In the following, Σ will denote a finite nonempty *alphabet*. We will write Σ^* for the set of all words, and Σ^+ for the set of all nonempty words over Σ. The *empty word* shall be denoted by λ. As usual, $|x|$, the *length* of a word x, is the number of letters in x. The set Ω shall denote some finite *set of parentheses*. Of course, Ω and Σ are always disjoint, and elements of Ω are usually written as $\langle_1, \rangle_1, \langle_2, \rangle_2, \ldots$ We will also assume here that each Ω is partitioned into sets of *opening and closing parentheses*, denoted by Ω_{op} and Ω_{cl} respectively, which are in bijective correspondence. For any integer $j \geq 0$, let Ω_j stand for a set of j pairs of parentheses, that is $\Omega_j = \{ \langle_1, \rangle_1, \ldots, \langle_j, \rangle_j \}$. It is convenient to choose $\Omega_0 := \emptyset$.

Let $\Sigma^*(\bullet, \circ)$ denote the *free binoid generated by* Σ. For simplicity, let us call the elements of $\Sigma^*(\bullet, \circ)$, *biwords* (over Σ). We will give concrete representations of biwords in the sequel. The identity of $\Sigma^*(\bullet, \circ)$, denoted by λ, is the *empty biword*. Each generator of $\Sigma^*(\bullet, \circ)$ corresponding to a letter $\sigma \in \Sigma$ is called a *singleton biword* and will also be denoted by σ. The biwords that can be written as a horizontal (resp. vertical) product of two nonempty biwords are called *horizontal* (resp. *vertical*). We call this property the *type* of a biword.

Of course there are several possible ways of describing biwords. They may be represented by relational structures called sp-biposets [6], but biwords may also be regarded as labeled ordered unranked trees. Here we will employ two linear representations, namely terms and condensed terms.

Now we associate the term representation w^{tm} with each biword $w \in \Sigma^*(\bullet, \circ)$. To this end, we extend the alphabet Σ with operation symbols and parentheses. Let $E(\Sigma) := \Sigma \cup \{ \bullet, \circ, \langle, \rangle \}$. In the term representation we shall put parentheses around the subterm of horizontal biwords that appear as vertical factors, and symmetrically around the subterm of vertical biwords that appear as horizontal factors. This procedure can be stated more precisely as follows:

Definition 1. *If $w \in \Sigma^*(\bullet, \circ)$, then w^{tm} will denote the* term representation *of w. Let w^{tm} be a word over $E(\Sigma)$, defined inductively as follows.*

(i) *If $w = \lambda$ is the empty biword, then $w^{tm} := \lambda$.*
(ii) *If $w = \sigma \in \Sigma$ is a singleton biword, then $w^{tm} := \sigma$.*
(iii) *If $w = w_1 \bullet w_2$ with $w_1, w_2 \neq \lambda$, then $w^{tm} :=$ $\mathrm{Hform}(w_1) \bullet \mathrm{Hform}(w_2)$*[2]

[2] Here $\mathrm{Hform}(w_1) \bullet \mathrm{Hform}(w_2)$ stands for the concatenation of the word $\mathrm{Hform}(w_1)$ with the letter '$\bullet$' and with the word $\mathrm{Hform}(w_2)$.

(iv) *If* $w = w_1 \circ w_2$ *with* $w_1, w_2 \neq \lambda$, *then* $w^{tm} := \mathrm{Vform}(w_1) \circ \mathrm{Vform}(w_2)$.

In (iii), $\mathrm{Hform}(w)$ *denotes the* horizontal form *of the biword* w, *defined as*

$$\mathrm{Hform}(w) := \begin{cases} w^{tm} & \text{if } w \text{ is a singleton or horizontal biword,} \\ \langle\, w^{tm} \,\rangle & \text{if } w \text{ is a vertical biword.} \end{cases}$$

In (iv), $\mathrm{Vform}(w)$, *the* vertical form *of* w, *is defined symmetrically.*

It should be mentioned here that in cases (iii) and (iv) the definition of w^{tm} does not depend on the choice of factorization because of the associativity of the operations $\bullet$, $\circ$ and of the concatenation of words.

Another description of $\Sigma^*(\bullet, \circ)$ can be given using *condensed terms*, or *cterms* for short. The condensation of the description of the terms is based on a simple observation. It is that the operation symbols can be omitted provided we know the type of a biword in advance. Actually, the arrangement of the parentheses tells us precisely where we should put the horizontal and vertical product operations between the factors. Formally, condensed term representations are words from the set $\{\,\lambda\,\} \cup \Sigma \cup \{\,\bullet, \circ\,\}\big(\Sigma \cup \{\,\langle, \rangle\,\}\big)^+$.

For a nonempty and nonsingleton biword, the first letter –the *type-sign*– gives the type of the represented biword; namely $\bullet$ and $\circ$ designate the horizontal type and vertical type, respectively. The remaining part is just the term representation after the operation symbols have been deleted. We will write w^{ctm} for the cterm representation of biword w. Thus if $w^{tm} = a \bullet \langle b \circ \langle c \bullet d \rangle \rangle \bullet \langle e \circ f \rangle$, then $w^{ctm} = \bullet a \langle b \langle cd \rangle \rangle \langle ef \rangle$. We can extend these notations to languages. Let $L^{tm} := \{\, w^{tm} \mid w \in L\,\}$ and $L^{ctm} := \{\, w^{ctm} \mid w \in L\,\}$. Then, let $\mathrm{TM}(\Sigma) := \Sigma^*(\bullet, \circ)^{tm}$ and $\mathrm{CTM}(\Sigma) := \Sigma^*(\bullet, \circ)^{ctm}$ denote the set of all (c)terms of biwords over Σ.

As we shall see, in order to accept all recognizable binoid languages by monoid/parenthesizing automata, it is necessary to employ several pairs of parentheses. For this reason we choose an integer $i \geq 0$, and let $E_i(\Sigma) = \Sigma \cup \Omega_i \cup \{\bullet, \circ\}$ be the *extended alphabet with* i *different pairs of parentheses*. Suppose that w^{tm} (resp. w^{ctm}) is a (c)term representation of a biword $w \in \Sigma^*(\bullet, \circ)$. Now i-term (resp. i-cterm) representations of w are obtained by replacing the matching pairs of parentheses with pairs of indexed parentheses from Ω_i in w^{tm} (resp. in w^{ctm}). Note that a biword can have several different i-(c)term representations. E.g. $\langle_2 a \bullet b \rangle_2 \circ \langle_1 c \bullet d \rangle_1$ and $\langle_1 a \bullet b \rangle_1 \circ \langle_1 c \bullet d \rangle_1$ are both 2-term representations of the biword $\langle a \bullet b \rangle \circ \langle c \bullet d \rangle$. Now let $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$ stand for the i-term and i-cterm representations of the biwords in $\Sigma^*(\bullet, \circ)$, respectively. For a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$, any word language $L' \subseteq \mathrm{TM}_i(\Sigma)$ (resp. $L' \subseteq \mathrm{CTM}_i(\Sigma)$) such that $\eta_i(L') = L^{tm}$ (resp. $\eta_i(L') = L^{ctm}$) will be referred to as an *i-term* (resp. *i-cterm*) *representation* of L. Here η_i is the mapping that deletes the indices of the parentheses, i.e. the homomorphism $\eta_i : E_i(\Sigma)^* \to E(\Sigma)^*$ which extends

$$\widetilde{\eta}_i(x) = \begin{cases} x \text{ if } x \in \Sigma \cup \{\,\bullet, \circ\,\}; \\ \langle \text{ if } x \in \Omega_{i,op}; \\ \rangle \text{ if } x \in \Omega_{i,cl}, \end{cases} \qquad \text{for all } x \in E(\Sigma).$$

Proposition 2. $\mathrm{TM}_i(\Sigma)$ *and* $\mathrm{CTM}_i(\Sigma)$ *are deterministic context-free languages.*

Proof sketch. Both $\mathrm{TM}_i(\Sigma)$ and $\mathrm{CTM}_i(\Sigma)$, as certain subsets of $E_i(\Sigma)^*$, can be characterized by some simple conditions, and the conditions can be transformed into deterministic pushdown automata.

3 The Monoid Approach

In [11] Hashiguchi et al. introduced two modes of operations of monoid automata for defining binoid languages.

Definition 3. [11] *Given a monoid automaton $\mathcal{A}$ over the alphabet $E(\Sigma)$ and a binoid language $L \subseteq \Sigma^*(\bullet, \circ)$, we say that*

i) $\mathcal{A}$ accepts L in the free monoid mode *if, for any word $x \in E(\Sigma)^*$, $\mathcal{A}$ accepts x iff x is a term representation of a biword in L.*

ii) $\mathcal{A}$ accepts L in the free binoid mode *if, for any term representation $w \in$ $\mathrm{TM}(\Sigma)$, $\mathcal{A}$ accepts w iff w is a term representation of a biword in L.*

Let $\mathsf{Reg}^{\mathsf{FM}}$ and $\mathsf{Reg}^{\mathsf{FB}}$ denote the classes of binoid languages that can be accepted in the free monoid mode and in the free binoid mode, respectively.

The concept of relativized regularity below will be useful for giving brief formulations of various acceptance modes.

Definition 4. *Let Σ be an alphabet and $U \subseteq \Sigma^*$ be an arbitrary language. Now consider a language $L \subseteq U$. We say that L is* regular relative to U *(or L is U-*regular *for short), if there exists a regular language $\hat{L} \subseteq \Sigma^*$ such that $L = \hat{L} \cap U$.*

For example $L = \{\langle^n\rangle^n \mid n \geq 0\}$ is a Dyck-regular language (cf. [9]), since $L = \hat{L} \cap D_1$ with a regular language $\hat{L} = \langle^*\rangle^*$, where D_1 denotes the Dyck language over a pair of parentheses. We can now reexpress Definition 3 in the following way.

Fact 5. (i) $L \in \mathsf{Reg}^{\mathsf{FM}} \Leftrightarrow L^{tm}$ *is a regular word language.*

(ii) $L \in \mathsf{Reg}^{\mathsf{FB}} \Leftrightarrow L^{tm}$ *is* $\mathrm{TM}(\Sigma)$*-regular word language.*

To provide the main results of Hashiguchi at al. we need two additional concepts.

First we say that a binoid language L has a *bounded depth* if there is an integer K such that, for every biword $w \in L$, the maximal number of nested parentheses in w^{tm} is at most K. Let BD denote the class of binoid languages that have a bounded depth.

Second, let BRat denote the class of *birational languages*. They are those binoid languages that can be obtained from the finite binoid languages by applying the operations of union, horizontal and vertical products, and the two Kleene-iterations of the two products[3].

It is not hard to see that $\mathsf{Reg}^{\mathsf{FM}} \subseteq \mathsf{BD}$ and $\Sigma^*(\bullet, \circ) \in \mathsf{Reg}^{\mathsf{FB}} \setminus \mathsf{BD}$. Hence we have $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}^{\mathsf{FB}}$, cf. [11]. The main result of [13, 14] can be summarized as follows.

[3] In [13, 14] birational languages are introduced via regular binoid expressions and BRat is also called 'the languages denoted by regular binoid expressions'.

Theorem 6. (Hashiguchi et al. [13, 14]) $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{BRat}$.

The result above gives a nice operational characterization of $\mathsf{Reg}^{\mathsf{FM}}$, but this class is not closed under complementation; see [6] for more details.

We can build another acceptance mode of monoid automata by using the cterm representation instead of terms. We say that a monoid automaton $\mathcal{A}$ *accepts a binoid language* L *in the* C_1*-mode* if, for any cterm representation $w \in \mathrm{CTM}(\Sigma)$, $\mathcal{A}$ accepts w iff w is a cterm representation of a biword in L. From now on the free binoid mode will also be called the T_1*-mode*. Moreover, we will extend the T_1 and C_1 modes using i-terms and i-cterms.

Definition 7. *Given an integer* $i \geq 0$*, a monoid automaton* $\mathcal{A}$ *over the alphabet* $E_i(\Sigma)$ *and a binoid language* $L \subseteq \Sigma^*(\bullet, \circ)$*, we say that* $\mathcal{A}$ accepts L in the T_i-mode *(resp.* in the C_i-mode*) if, for any i-term (resp. i-cterm) representation* $x \in (\mathrm{C})\mathrm{TM}_i(\Sigma)$*, automaton* $\mathcal{A}$ *accepts* x *iff the biword represented by* x *is in* L.

Let $\mathsf{Reg}_i^{\mathsf{T}}$ and $\mathsf{Reg}_i^{\mathsf{C}}$ denote the classes of languages that can be accepted by a monoid automaton over $E_i(\Sigma)$ in the T_i-mode and the C_i-mode, respectively. Furthermore, let $\mathsf{Reg}_\infty^{\mathsf{T}} := \cup_{i=0}^{\infty} \mathsf{Reg}_i^{\mathsf{T}}$ and $\mathsf{Reg}_\infty^{\mathsf{C}} := \cup_{i=0}^{\infty} \mathsf{Reg}_i^{\mathsf{C}}$.

Fact 8.

(i) $L \in \mathsf{Reg}_i^{\mathsf{T}} \Leftrightarrow$ *there exists a* $\mathrm{TM}_i(\Sigma)$*-regular* i*-term representation of* L.

(ii) $L \in \mathsf{Reg}_i^{\mathsf{C}} \Leftrightarrow$ *there exists a* $\mathrm{CTM}_i(\Sigma)$*-regular* i*-cterm representation of* L.

4 Parenthesizing Automata

Here we give a brief overview of parenthesizing automata described in [6, 20, 21].

Definition 9. [6] *A (nondeterministic)* parenthesizing automaton*, PA for short, is a 9-tuple* $\mathcal{A} := (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$*, where* S *is a nonempty, finite set of* states*;* H *and* V *are the sets of* horizontal *and* vertical states *which give a disjoint partition of* S*,* Σ *is the* input alphabet *and* Ω *is a finite set of parentheses. Furthermore,*

- $\delta \subseteq (H \times \Sigma \times H) \cup (V \times \Sigma \times V)$ *is the* labeled transition relation*,*
- $\gamma \subseteq (H \times \Omega \times V) \cup (V \times \Omega \times H)$ *is the* parenthesizing transition relation*,*
- $I, F \subseteq S$ *are the sets of* initial *and* final states*, respectively.*

Example 10. A simple illustration of a PA is given in Figure 1. The horizontal states are those labeled by H_i and the vertical states are those labeled by V_j, for some i and j. There is a single initial state H_1, and a single final state H_7. Later we will see that this automaton has a single run from H_1 to H_7, hence the automaton just accepts the biword $a \bullet \langle b \circ \langle c \bullet d \rangle\rangle \bullet e$. Of course, if the automaton had cycles, the accepted binoid language would be more complicated than in our example.

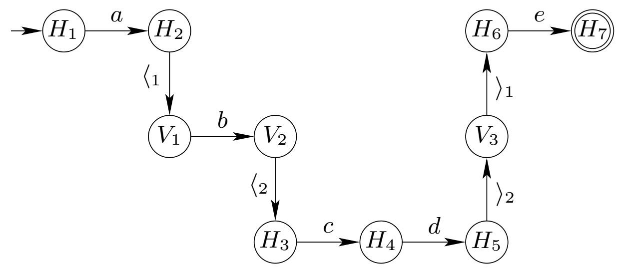

Fig. 1. A PA accepting $\{a \bullet \langle b \circ \langle c \bullet d \rangle\rangle \bullet e\}$.

Let $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ be a PA. If $t = (p, x, q)$ is a labeled or parenthesizing transition of $\mathcal{A}$, i.e. $t \in \delta \cup \gamma$, then the *starting* and the *ending state* of t will be denoted by $\mathrm{start}(t) := p$ and $\mathrm{end}(t) := q$, respectively. Two transitions t_1 and t_2 are *adjacent* (in this order) if $\mathrm{end}(t_1) = \mathrm{start}(t_2)$. From now on we will demand that in any transition sequence the consecutive transitions shall be adjacent. If $\mathbf{r} = t_1 t_2 \dots t_n \in (\delta \cup \gamma)^*$ is a transition sequence, then let $\mathrm{start}(\mathbf{r}) := \mathrm{start}(t_1)$ and $\mathrm{end}(\mathbf{r}) := \mathrm{end}(t_n)$. Here we say that two parenthesizing transitions $t_1 = (p, \omega_1, q)$ and $t_2 = (s, \omega_2, t) \in \gamma$ form a *parenthesizing transition pair* if ω_1 is an opening parenthesis and ω_2 is its closing partner.

Definition 11. [21] *Let $\mathcal{A}$ be a parenthesizing automaton. The set of its* runs*, $\mathrm{Runs}(\mathcal{A})$, is the least set of transition sequences that contains*

(i) *the* singleton runs: (p, σ, q), *for all* $(p, \sigma, q) \in \delta$*;*
(ii) *the* direct runs: $\mathbf{r_1 r_2}$, *for every* $\mathbf{r_1}, \mathbf{r_2} \in \mathrm{Runs}(\mathcal{A})$ *with* $\mathrm{end}(\mathbf{r_1}) = \mathrm{start}(\mathbf{r_2})$*;*
(iii) *the* indirect runs: $t_1 \mathbf{r}\, t_2$, *for every direct run* $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$, *and parenthesizing transition pair* t_1, t_2 *with* $\mathrm{end}(t_1) = \mathrm{start}(\mathbf{r})$ *and* $\mathrm{end}(\mathbf{r}) = \mathrm{start}(t_2)$.

Suppose that $\mathcal{A}$ is a PA and $\mathbf{r} = t_1 \dots t_n \in \mathrm{Runs}(\mathcal{A})$. A parenthesizing transition pair t_i, t_j, $(i < j)$ is said to be a *matching parenthesizing transition pair in* $\mathbf{r}$ if $t_i \dots t_j$ is an indirect run of $\mathcal{A}$. Note that not every parenthesizing transition pair t_i, t_j with $i < j$ is a matching parenthesizing transition pair in $\mathbf{r}$.

Definition 12. *Suppose that $\mathcal{A}$ is a PA and $\mathbf{r} \in \mathrm{Runs}(\mathcal{A})$. The* label *of $\mathbf{r}$ is a biword from $\Sigma^*(\bullet, \circ)$ defined inductively as follows:*

(i) *If* $\mathbf{r} = (p, \sigma, q)$, *then* $\mathrm{Label}(\mathbf{r}) := \sigma$.
(ii) *If* $\mathbf{r}$ *is a direct run, and* $\mathbf{r} = \mathbf{r_1 r_2}$ *for some* $\mathbf{r_1}, \mathbf{r_2} \in \mathrm{Runs}(\mathcal{A})$, *then*
  - *if* $\mathrm{end}(\mathbf{r_1}) \in H$, *then* $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r_1}) \bullet \mathrm{Label}(\mathbf{r_2})$*;*
  - *if* $\mathrm{end}(\mathbf{r_1}) \in V$, *then* $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r_1}) \circ \mathrm{Label}(\mathbf{r_2})$.
(iii) *If* $\mathbf{r}$ *is an indirect run* $\mathbf{r} = t_1 \mathbf{r}'\, t_2$, *then* $\mathrm{Label}(\mathbf{r}) := \mathrm{Label}(\mathbf{r}')$.

Since $\bullet$ and $\circ$ are associative, the definition of $\mathrm{Label}(\mathbf{r})$ does not depend on the choice of factorization in case (ii) above.

A run from an initial state to a final state will be called an *accepting run*, and the binoid language accepted by a PA is defined as the set of labels of the accepting runs.

Definition 13. *The binoid* language $L(\mathcal{A})$ accepted by a PA $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ *is defined as* $\{\,\mathrm{Label}(\mathbf{r}) \mid \mathbf{r} \in \mathrm{Runs}(\mathcal{A}), \mathrm{start}(\mathbf{r}) \in I, \mathrm{end}(\mathbf{r}) \in F\,\}$, *and, additionally, if* $I \cap F \neq \emptyset$ *then* $L(\mathcal{A})$ *also contains* λ*, the empty biword.*

Definition 14. *A binoid language* $L \subseteq \Sigma^*(\bullet, \circ)$ *is called* regular *if there exists a PA that accepts it. Let* Reg *denote the* class of regular binoid languages *over all alphabets. Similarly, let* $\mathsf{Reg_i}$ *denote the classes of those binoid languages that can be accepted by a PA with at most* $i \geq 0$ *pairs of parentheses.*

Recall that a binoid language is said to be *recognizable* if it is recognized by a homomorphism into a finite binoid, i.e. $L \subseteq \Sigma^*(\bullet, \circ)$ is recognizable if and only if $L = \varphi^{-1}(F)$, for some binoid homomorphism $\varphi : \Sigma^*(\bullet, \circ) \to B$, where B is a finite binoid, and $F \subseteq B$. The notion of second order definability is also quite standard, but it is based on the sp-biposet representation of binoid languages. For the definitions see [6]. Next, some key results for regular binoid languages are the following.

Theorem 15. [6] *A binoid language* $L \subseteq \Sigma^*(\bullet, \circ)$ *is regular if and only if it is recognizable; and it is recognizable if and only if it is* MSO*-definable.*

Theorem 16. [6, 20] $\mathsf{BRat} = \mathsf{Reg} \cap \mathsf{BD} = \mathsf{Reg_1} \cap \mathsf{BD}$.

Theorem 17. [20] *The classes* $\mathsf{Reg_0} \subsetneq \mathsf{Reg_1} \subsetneq \mathsf{Reg_2} \subsetneq \dots$ *form a strict hierarchy of regular binoid languages.*

5 Comparison of models and modes

In this section we present our main result, namely the equivalence of PA that have i pairs of parentheses with monoid automata in both the T_i-mode and C_i-mode. For this we need to slightly modify the acceptance conditions of PA. Namely, we will not allow indirect runs as accepting runs. If $\mathcal{A} = (S, H, V, \Sigma, \Omega, \delta, \gamma, I, F)$ is a PA, let $L'(\mathcal{A}) := \{\,\mathrm{Label}(\mathbf{r}) \mid \mathbf{r} \in \mathrm{Runs}(\mathcal{A}), \mathrm{start}(\mathbf{r}) \in I, \mathrm{end}(\mathbf{r}) \in F$ and $\mathbf{r}$ *is not an indirect run*$\}$ and additionally, if $I \cap F \neq \emptyset$ then $L'(\mathcal{A})$ also contains λ, the empty biword. Moreover, let $\mathsf{Reg'_i}$ denote the class of those binoid languages that can be written as $L'(\mathcal{A})$ with a PA that has at most i pairs of parentheses.

At first sight it seems that the classes $\mathsf{Reg'_i}$ are smaller than the original classes $\mathsf{Reg_i}$. But this is not true; on the contrary, while indirect acceptance can be simulated in the new "no indirect acceptance" mode, the converse simulation is not possible. If we have a PA $\mathcal{A}$ whose initial and final states are all horizontal, then we are sure that $L'(\mathcal{A})$ just contains horizontal biwords. On the other hand, this property cannot be guaranteed in the old acceptance mode. Furthermore, it can be proved that the set of all horizontal biwords is in $\mathsf{Reg'_1} \setminus \mathsf{Reg_1}$. Thus one can verify (with considerable effort) the following correspondence between the new and the old acceptance modes of PA.

Theorem 18. *We have* $\mathsf{Reg_0} = \mathsf{Reg'_0}$*, and* $\mathsf{Reg_i} \subsetneq \mathsf{Reg'_i} \subsetneq \mathsf{Reg_{i+1}}$*, for all* $i \geq 1$.

Earlier results, namely Theorem 16 and Theorem 6, lead to the following characterization of the free monoid mode in terms of PA.

Theorem 19. $\mathsf{Reg}^{\mathsf{FM}} = \mathsf{Reg}_1 \cap \mathsf{BD} = \mathsf{Reg}'_1 \cap \mathsf{BD}$.

Now we will establish a connection between the free binoid mode and PA.

Lemma 20. *For any $i \geq 0$, there exists a finite transduction $\tau_i : E_i(\Sigma)^* \to E_i(\Sigma)^*$ which transforms every i-cterm to the equivalent i-term.*

Proof sketch. Observe that τ_i can be induced by a finite transducer like the one depicted in Figure 2.

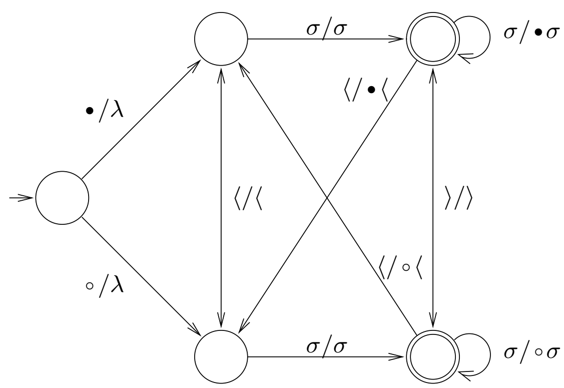

Fig. 2. A finite transducer which transforms 1-cterms into 1-terms over a one-letter alphabet $\Sigma = \{\,\sigma\,\}$.

Corollary 21. *Suppose that $L \subseteq \mathrm{CTM}_i(\Sigma)$ is a word language for some $i \geq 0$. Then L is $\mathrm{CTM}_i(\Sigma)$-regular iff $\tau_i(L)$ is $\mathrm{TM}_i(\Sigma)$-regular.*

This result can be interpreted as follows. In the description of a binoid language by words we can use the cterm (resp. i-cterm) representation instead of the term (resp. i-term) representation, i.e. we can neglect the operation symbols without affecting the regularity of the language. This simplification may be useful in syntactic proofs using representations of biwords via words as in [13] and [14], and it is also crucial in the proof of our main theorem presented below.

Theorem 22. *For any integer $i \geq 0$, we have* $\mathsf{Reg}'_\mathsf{i} = \mathsf{Reg}^{\mathsf{C}}_i = \mathsf{Reg}^{\mathsf{T}}_i$.

Proof sketch. The second equality can easily be derived from Corollary 21. On the other hand, the proof of $\mathsf{Reg}'_\mathsf{i} = \mathsf{Reg}^{\mathsf{C}}_i$ is rather technical. We need to transform a PA into an equivalent monoid automaton over $E_i(\Sigma)$ and vice versa. We will use the following notation for a (nondeterministic) monoid automaton: $\mathcal{A} = (S, \Sigma, \delta, I, F)$, where S is the set of states, Σ is the input alphabet, $\delta : S \times \Sigma \to 2^S$

is the transition function, and I and F are the sets of initial and final states respectively.

Let $\mathcal{A} = (S, H, V, \Sigma, \Omega_i, \delta, \gamma, I, F)$ be a PA. If we do not distinguish between the horizontal and vertical states, and if we do not distinguish between labeling and parenthesizing transitions, then we obtain a monoid automaton $(S, \Sigma \cup \Omega_i, \delta \cup \gamma, I, F)$. Let us take two new states $i_*, f_* \notin S$. Now replace I with $\{i_*\}$, and add the following transitions to $\delta \cup \gamma$

$$\delta' = \{(i_*, \bullet, i) \mid i \in I \cap H\} \cup \{(i_*, \circ, i) \mid i \in I \cap V\} \cup \{(i_*, \sigma, f_*) \mid \sigma \in L'(\mathcal{A})\}.$$

We will regard i_* as final state, iff $\lambda \in L'(\mathcal{A})$. Thus we get a monoid automaton $\mathcal{A}^M := (S, E_i(\Sigma), \delta \cup \gamma \cup \delta', \{i_*\}, F')$, where $F' = F \cup \{i_*, f_*\}$, if $\lambda \in L'(\mathcal{A})$, and $F' = F \cup \{f_*\}$ otherwise. It can be proved that $\mathcal{A}^M$ in the C_i-mode accepts $L'(\mathcal{A})$. Hence $\mathsf{Reg}'_{\mathsf{i}} \subseteq \mathsf{Reg}^{\mathsf{C}}_i$.

$\mathsf{Reg}^{\mathsf{C}}_i \subseteq \mathsf{Reg}'_i$ can be proved as follows. Let $\mathcal{A} = (S, E_i(\Sigma), \delta, I, F)$ be a monoid automaton which in the C_i-mode accepts a binoid language L. A PA $\mathcal{A}'$ such that $L'(\mathcal{A}') = L$ can be defined as follows:

$$\begin{aligned}
&\mathcal{A}' = (H' \cup V', H', V', \delta', \gamma', I', F'), \text{ where} \\
&H' = \{s^H \mid s \in S\} \cup \{i_*, f_*\}, \quad V' = \{s^V \mid s \in S\}, \quad i_*, f_* \notin S, \\
&\delta' = \{(p^H, \sigma, q^H), (p^V, \sigma, q^V) \mid \sigma \in \Sigma, (p, \sigma, q) \in \delta\} \cup \{(i_*, \sigma, f_*) \mid \sigma \in L\}, \\
&\gamma' = \{(p^H, \omega, q^V), (p^V, \omega, q^H) \mid \omega \in \Omega_i, (p, \omega, q) \in \delta\}, \\
&I' = \{p^H \mid \exists i \in I : (i, \bullet, p) \in \delta\} \cup \{p^V \mid \exists i \in I : (i, \circ, p) \in \delta\} \cup \{i_*\}, \\
&F' = \begin{cases} \{f^H, f^V \mid f \in F\} & \text{if } \lambda \notin L, \\ \{f^H, f^V \mid f \in F\} \cup \{i_*\} & \text{if } \lambda \in L. \end{cases}
\end{aligned}$$

Corollary 23. $\mathsf{Reg}^{\mathsf{FB}} = \mathsf{Reg}'_1$*, so* Reg'_1 *is closed under complementation.*

The next result shows that the more general class of recognizable binoid languages can also be captured by monoid mode acceptance.

Corollary 24. $\mathsf{Reg} = \mathsf{Reg}^{\mathsf{C}}_\infty = \mathsf{Reg}^{\mathsf{T}}_\infty$.

Proposition 2 and the concept of relativized regularity lead to the following result, whose proof is straightforward.

Corollary 25. *For all $i \geq 0$, any binoid language in Reg'_i has a deterministic context-free i-term (and i-cterm) representation. In particular, for any $L \in \mathsf{Reg}'_1$ the word languages L^{tm} and L^{ctm} are deterministic context-free.*

6 Conclusions and Final Remarks

Now let us summarize certain key points about the paper. The first is that Theorem 22 is effective in the sense that, for a PA, an equivalent monoid automaton (for the T$_i$-mode or C$_i$-mode) can be constructed and vice versa. Furthermore, the transition algorithm increases the number of states of the automaton by just a constant factor.

The operation of a PA is very similar to that of a visibly pushdown automaton (or VPA for short) [1]. One can imagine that a PA uses a pushdown storage which works in the following way. During an opening parenthesizing transition labeled by $\langle_i$ the automaton puts the index i to the top of the stack, and later the automaton can perform a closing parenthesizing transition labeled $\rangle_i$, only if the index i can be popped from the stack. Notice that a PA alters the stack only when it performs parenthesizing transitions: opening parentheses correspond to push operations, while closing parentheses correspond to pop operations. This is what is called visibly pushdown behavior. Therefore each PA over $\Sigma^*(\bullet, \circ)$ with i pairs of parentheses naturally corresponds to a VPA with i stack symbols (operating on words over the pushdown alphabet $\widehat{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_\ell \rangle$, where $\Sigma_c = \{\langle\}$, $\Sigma_r = \{\rangle\}$ and $\Sigma_\ell = \Sigma$, cf. [1].) However a PA operates on biwords, not on words, hence the syntactic check of the input – not just the well balanced aspect of the parentheses, but also making sure that no empty or superfluous parenthesization occurs – is not the task of a PA. Consequently i stack symbols are not enough for a VPA to accept the words of L^{ctm} or L^{tm}, for an $L \in \mathsf{Reg}'_\mathsf{i}$. But it can be proved that both TM(Σ) and CTM(Σ) can be accepted by a VPA with 3 stack symbols, and hence $3i$ is an upper bound on the number of stack symbols that is necessary (the proofs are left to the reader). Hence we have that, for all $L \in \mathsf{Reg}$, the languages L^{tm} and L^{ctm} are visibly pushdown, hence deterministic context-free languages (cf. Corollary 26).

From a regularity point of view three classes of binoid languages might be of interest to us, namely $\mathsf{Reg}^{\mathsf{FM}}$, Reg'_1 and $\mathsf{Reg} = \mathsf{Rec}$, where the inclusion relations for them are $\mathsf{Reg}^{\mathsf{FM}} \subsetneq \mathsf{Reg}'_1 \subsetneq \mathsf{Reg}$. Note that, unlike the two other classes, $\mathsf{Reg}^{\mathsf{FM}}$ is not closed under complementation. Moreover, Theorems 17 and 22 tell us that in order to attain the general concept of recognizability (Rec) we need to handle several pairs of parentheses symbols and unambiguous representations. However, this unambiguity can be avoided since from the proof of $\mathsf{Rec} \subseteq \mathsf{Reg}$ of [6] we obtain a PA that can be regarded as deterministic. This is not surprising at all, as recognizability is clearly a deterministic notion.

Another key point of our above results is that, by generalizing the idea of Hashiguchi et al., we managed to reduce the investigation of recognizable binoid languages to the classical theory of word languages. But the well-developed theory of monoid automata and word languages cannot be applied directly since the reduction has been done via the concept of relativized regularity. Hence it would be desirable to provide a detailed exposition of relativized regularity and find out how the methods of monoid automata (determinization, minimization and so on) can be transformed into automata over biwords, and, more generally, learn what effect the theory of word languages has on the theory of binoid languages. Finally, a deeper understanding of automata models, and especially the phenomenon of two-dimensional iterations, may also lead us to an operational characterization of the class of recognizable binoid languages. These things are what we plan to study in the near future.

Acknowledgements. The author wishes to thank the anonymous referees for their many valuable suggestions. He would also like to thank David Curley for checking the paper from a linguistic point of view and for providing some inspirational ideas during the writing stage.

References

1. R. Alur and P. Madhusudan. Visibly pushdown languages. In proc: *STOC 2004*, Chicago, IL, USA, June 13-16, 2004, 202–211.
2. Rajeev Alur, P. Madhusudan. Adding Nesting Structure to Words. In proc: *DLT 2006*, LNCS 4036, 1-13
3. I. Dolinka. A note on identities of two-dimensional languages. *Disc. Appl. Math.*, 146(2005), 43–50.
4. I. Dolinka. Axiomatizing the identities of binoid languages. *Theoret. Comp. Sci.*, 372(2007), 1–14.
5. A. Ehrenfeucht and G. Rozenberg. T-structures, T-functions and texts. *Theoret. Comput. Sci.*, 116(1993), 227–290.
6. Z. Ésik and Z. L. Németh. Higher dimensional automata, *J. of Autom. Lang. Comb.*, 9(2004), 3–29.
7. Z. Ésik and Z. L. Németh. Algebraic and graph-theoretic properties of infinite n-posets. *Theoret. Informatics Appl.*, 39(2005), 305–322.
8. D. Giammarresi and A. Restivo. Two-dimensional languages. In: *Handbook of Formal Languages*, Vol. 3, Springer-Verlag, Berlin, 1997, 215–267.
9. M. A. Harrison. *Introduction to formal language theory.* Addison-Wesley Publishing Co., Reading, Mass., 1978.
10. K. Hashiguchi, K. Hashimoto, S. Jimbo. Modified RSA cryptosystems over bicodes. *Advances in algebra*, World. Sci. Publ., NJ, 2003, 377–389.
11. K. Hashiguchi, S. Ichihara and S. Jimbo. Formal languages over free binoids. *J. Autom. Lang. Comb.*, 5(2000), 219–234.
12. K. Hashiguchi, T. Kunai, S. Jimbo. Finite Codes over Free Binoids. *J. Autom. Lang. Comb.*, 7(2002), 505–518.
13. K. Hashiguchi, Y. Wada and S. Jimbo. Regular binoid expressions and regular binoid languages. *Theoret. Comput. Sci.*, 304(2003), 291–313.
14. K. Hashiguchi, Y. Sakakibara and S. Jimbo. Equivalence of regular binoid expressions and regular expressions denoting binoid languages over free binoids. *Theoret. Comput. Sci.*, 312(2004), 251–266.
15. H. J. Hoogeboom and P. ten Pas. Monadic second-order definable text languages. *Theory Comput. Syst.*, 30(1997), 335–354.
16. D. Kuske. Towards a language theory for infinite N-free pomsets. *Theoret. Comput. Sci.*, 299(2003), 347–386.
17. K. Lodaya and P. Weil. Kleene iteration for parallelism. In: proc. *FST & TCS'98*, LNCS 1530, Springer-Verlag, 1998, 355–366.
18. K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property. *Theoret. Comput. Sci.*, 237(2000), 347–380.
19. K. Lodaya and P. Weil. Rationality in algebras with series operation. *Inform. and Comput.*, 171(2001), 269–293.
20. Z. L. Németh. A Hierarchy Theorem for Regular Languages over Free Bisemigroups. *Acta Cybern.*, 16(2004), 567–577.
21. Z. L. Németh. Automata on Infinite Biposets. *Acta Cybern.*, 18(2006), 765–797.
22. P. Weil. Algebraic recognizability of languages. In: proc. *MFCS 2004*, LNCS 3153, Springer-Verlag, 2004, 149–175.