

A Note on List Languages

Jean Berstel

Institut Gaspard Monge (IGM)
Université Marne-la-Vallée
2, rue de la Butte Verte, 93166 Noisy-le-Grand Cédex

Luc Boasson

Laboratoire d'informatique algorithmique: fondements et applications (LIAFA)
Université Denis-Diderot
2, place Jussieu, 75251 Paris Cédex 05

Abstract

We prove that fair list languages are not closed under reversal. This answers a question of a paper by Breveglieri. We then introduce a “strict” parallel product that defines a family closed under reversal; we show that this product properly generalizes the parallel product.

1 Introduction

A *list* over an alphabet A is a sequence $\langle u_1; u_2; \dots; u_k \rangle$ of words over A . List languages have been considered by Breveglieri in [?]. The main difference between lists and words is that lists can be merged, in various ways, by using the catenation in the free monoid A^* on the terms of lists. In this sense, lists over an alphabet are close to lists as considered in algorithmics and programming languages where operations are not only over the structure of lists but also on the elements of a list. Typical examples are the “prefix computation” (see [?]) or other operations, as described in [?].

Several operations can be defined over lists. *Fair* list languages are languages obtained from finite list languages by union, concatenation and its star, and by a parallel product and its associated parallel star. These languages were considered in depth by [?]. The aim of this note is to give

an example showing that the set of fair list languages is not closed under reversal. This answers a question of [?].

2 Preliminaries

In [?], the notions of list language and fair list language are introduced. We briefly recall these definitions.

Given an alphabet A , a *list* is a finite sequence $\ell = \langle u_1; u_2; \dots; u_n \rangle$ of words u_1, u_2, \dots, u_n over A . The elements of the list are separated by the special symbol “;”. The integer n is the *length* of the list. The *empty list* contains no word and is denoted $\langle \rangle$. Observe that this list is different from the list composed of the single empty word $\langle \epsilon \rangle$.

A *list language* is a set of lists. Given two lists $\ell_1 = \langle u_1; u_2; \dots; u_k \rangle$ and $\ell_2 = \langle v_1; v_2; \dots; v_h \rangle$, the following operations are defined :

- the product $\ell = \ell_1.\ell_2$ is the catenation of the two lists

$$\ell = \langle u_1; u_2; \dots; u_k; v_1; v_2; \dots; v_h \rangle.$$

- the *list merge* $\ell = \ell_1 \parallel \ell_2$ defined as follows:

- if $k > h$, then $\ell = \langle u_1v_1; u_2v_2; \dots; u_hv_h; u_{h+1}; \dots; u_k \rangle$.
- if $h = k$, then $\ell = \langle u_1v_1; u_2v_2; \dots; u_kv_k \rangle$.
- if $k < h$, then $\ell = \langle u_1v_1; u_2v_2; \dots; u_kv_k; v_{k+1}; \dots; v_h \rangle$

The empty list $\langle \rangle$ is a neutral element for both products. The list merge will be called here the *parallel product* of the two lists. The shorter of the two lists can be viewed as completed by the necessary number of empty words. The product and parallel product of lists are extended to list languages as usual: given two list languages L_1 and L_2 , the *product* of L_1 and L_2 is defined by

$$L_1.L_2 = \{\ell_1.\ell_2 \mid \ell_1 \in L_1, \ell_2 \in L_2\},$$

the *parallel product* of L_1 and L_2 is defined by

$$L_1 \parallel L_2 = \{\ell_1 \parallel \ell_2 \mid \ell_1 \in L_1, \ell_2 \in L_2\}.$$

As usual again, the product and parallel product give rise to a unary closure operation each. The unary operation of product closure (= the *star*) is defined by

$$L^* = \bigcup_{i \geq 0} L_i \quad \text{where} \quad L_0 = \{\langle \rangle\} \quad \text{and} \quad L_{i+1} = L_i.L.$$

The unary operation of parallel product closure (= the *parallel star*) is then defined by :

$$L^{\parallel*} = \bigcup_{i \geq 0} L_i \quad \text{where} \quad L_0 = \{\langle \rangle\} \quad \text{and} \quad L_{i+1} = L_i \parallel L.$$

We then define (following [?]) a *fair expression* as an expression obtained by using the operations union, product, parallel product, star and parallel star on finite list languages. Such an expression represents a list language. By definition, a *fair list language* is any list language represented by some fair expression.

Example: Consider, over the alphabet $A = \{a, b\}$, the fair expression

$$L = (\langle a \rangle . \langle b \rangle^*)^{\parallel*}.$$

This is the set of lists

$$\{ \langle a^k; b^{k_1}; b^{k_2}; \dots; b^{k_p} \rangle \mid p \geq 0, k \geq k_1 \geq k_2 \dots \geq k_p \text{ and } k_p \geq 1 \}.$$

To a list, two ordinary words over A can be naturally associated : the first one is a word over the alphabet $A \cup \{ ; \}$ and is obtained by considering the list as an ordinary word. The second one is the word over A obtained by erasing all symbols “;” in the previous one. This first word is the *string* version, the second word the *flattened* version of the list [?]. These two transformations extend as usual to list languages. In particular, an ordinary language M over A is called a *flattened list language*, if there exist a fair list language L whose flattened image is M . Similarly, a *string list language* is the set of string versions of some list language.

Example. In [?] it is shown that the commutative image of any flattened list language is semi-linear. We show that there exists a string list language whose commutative image is not semi-linear. This answers negatively a question asked in [?].

Consider indeed, the example given above. We represent the commutative image of a word w over $\{a, b, ;\}$ by a triple $(|w|_a, |w|_b, |w|_;))$ where $|w|_c$ is the number of letters c in w . The commutative image of L is then given by

$$\{(k, \beta, p) \mid p \leq \beta \leq p.k\}.$$

Clearly, this set is not semi-linear. (See for instance [?].)

3 Fair list languages and reversal

The aim of this section is to prove that the family of fair list languages is not closed under reversal. We will show that this holds already for unary languages, i.e. for languages over a one-letter alphabet.

3.1 Unary list languages

A list language L is *length-infinite* if it contains lists of arbitrary length. A list language is *unary* if the alphabet of words is a one-letter alphabet. In this case, lists are obviously sequences of integers. Hence, we will denote them by the associated sequence of integers.

A unary list language L is *p-increasing* for some integer p if, for all $\ell = \langle n_1, \dots, n_k \rangle$ in L , one has

$$n_p \leq n_{p+1} \leq \dots \leq n_k$$

Observe that L may contain lists of less than p elements for which no conditions are requested. A list language is *ultimately increasing* if it is p -increasing for some p . Clearly, any length-finite list language is ultimately increasing.

A unary list language L is *unbounded* if, for all integers N and q , there exists a list $\ell = \langle n_1, \dots, n_k \rangle$ in L , such that $n_j \geq N$ for at least q indices j in $\{1, \dots, k\}$.

Thus, in an unbounded unary list language, you may always find a list with an integer as large as you want at a rank as large as you want. In particular, any unbounded unary list language is length-infinite.

Examples

- 1) The set of constant lists

$$\{ \langle n_1; n_2; \dots; n_p \rangle \mid p > 0, n_1 = n_2 = \dots = n_p > 0 \}$$

is 1-increasing and unbounded.

2) For a fixed p , the set of lists

$$\{ \langle n_1; n_2; \dots; n_p; \underbrace{1; 1; \dots; 1}_{q \text{ times}} \rangle \mid q \geq 0, n_1, n_2, \dots, n_p > 0 \}$$

is $p + 1$ -increasing and bounded.

3) The set of lists

$$\{ \langle n_1; 1; n_2; 1 \dots; n_{p-1}; 1; n_p \rangle \mid p > 0, n_1 = n_2 = \dots = n_p > 0 \}$$

is unbounded and non ultimately increasing.

4) The set of lists

$$\{ \langle n_1; n_2; \dots; n_p \rangle \mid p > 0, 1 \leq n_1 \leq n_2 \leq \dots \leq n_p \}$$

is unbounded and 1-increasing.

We now prove

Lemma 3.1 *A unary fair list language is not both unbounded and ultimately increasing.*

Proof : Let us call a unary, unbounded and ultimately increasing list language a UI list language for short. The proof is by structural induction; it consists in showing that it is impossible to generate a UI-language from finite list languages by using fair operations. Clearly, a finite list language is bounded, therefore is not UI. Let now L be a (infinite) UI list language.

1) Assume that $L = L_1 \cup L_2$. Clearly either L_1 or L_2 (or both) is a UI list language.

2) Assume next that $L = L_1.L_2$. We show that L_2 has to be an UI list language. Clearly, because L is length-infinite, either L_1 or L_2 or both is length-infinite. Assume first that L_1 is length-finite. Then, it is immediate that L_2 is an UI list language. So, assume that L_1 is length-infinite, and let $\ell_2 = \langle m_1; m_2; \dots; m_k \rangle$ be a fixed element of L_2 . Consider now any list ℓ_1 in L_1 with more than p elements, where p is chosen so that L is p -increasing. Due to this condition, all the integers in the list ℓ_1 beyond rank p are less than or equal to m_1 , and consequently L_1 is bounded. Moreover, $m_1 \leq m_2 \leq \dots \leq m_k$ showing that all lists in L_2 are increasing. For L to be an unbounded list

language, it is necessary that L_2 is unbounded. Since L_2 is 1-increasing, we get that L_2 is an UI list language.

3) Assume now that $L = L_1 \parallel L_2$ and that L is p -increasing. We show that either L_1 or L_2 is an UI list language. If L_1 and L_2 were both bounded, then L would be bounded too. Hence, either L_1 or L_2 or both are unbounded (hence length-infinite). Assume L_1 is unbounded, and choose a fixed list $\ell_2 = \langle m_1; m_2; \dots; m_k \rangle$ in L_2 . Because L contains $L_1 \parallel \ell_2$, we get that L_1 itself is ultimately increasing : indeed, the integers composing such a list in L_1 must be increasing from the rank $\max(p, k)$ on. So, L_1 is an UI list language. The case where L_2 is unbounded is treated in the same way.

4) Assume that $L = L_1^*$. We show that this is impossible. Consider indeed any list $\ell_1 = \langle n_1; n_2; \dots; n_k \rangle$ be any list in L_1 . Because L is ultimately increasing, any power ℓ_1^h must be ultimately increasing, which implies that $n_1 \leq n_2 \leq \dots \leq n_k \leq n_1$. It follows that $n_1 = n_2 = \dots = n_k = n$ for some n . If there is any other list ℓ_2 in L_1 , we get similarly that $\ell_2 = \langle m, m, \dots, m \rangle$ for some integer m . The star of $\ell_1 \cdot \ell_2$ then implies that $n = m$, because L is ultimately increasing. Then, all lists in L_1 are sequences of the same fixed integer. But then, L cannot be unbounded.

5) Assume finally that $L = L_1^{\parallel*}$. We show that this is impossible. Clearly, L_1 has to be length-infinite. Moreover, since L is unbounded, for any q , there exists a list in L_1 with a non zero j -th element for some $j \geq q$. Hence, there exists a list $\ell_1 = \langle n_1; n_2, \dots; n_k \rangle$ in L_1 such that $k \geq p$ and $n_j > 0$ for some j greater than p . Using the parallel star on ℓ_1 , we get a list ℓ with an entry at rank j as large as we want. Now, choose a new list ℓ_2 in L_1 with more than k elements. The list $\ell \parallel \ell_2$ is not p -increasing : the element at rank j will be greater than the elements at rank greater than k .

Hence, an UI list language L cannot be a fair list language. ■

3.2 Reversal

The *reversal* of a list $\ell = \langle u_1; u_2; \dots; u_k \rangle$ is the list $\ell^r = \langle u_k^r; \dots; u_2^r; u_1^r \rangle$, where u^r denotes the usual reversal of a word u . The reversal of a list language L is defined as usual: $L^r = \{\ell^r \mid \ell \in L\}$. We now prove the following

Proposition 3.2 *The family of fair list languages is not closed under reversal.*

Proof : Consider the one letter alphabet $A = \{a\}$. Over A , define the fair list language $L = (\{a\}^*)^{\parallel^*}$. It is easily seen that

$$L = \{\langle n_1, n_2, \dots, n_k \rangle \mid k \geq 0, n_1 \geq n_2 \geq \dots \geq n_k \geq 0\}$$

Then, the language L^r is an unbounded, ultimately increasing list language and, by lemma 3.1, it is not a fair list language. ■

This property leads naturally to consider a right parallel product instead of the parallel product \parallel (which will from now on be qualified as left parallel product) : given two lists $\ell_1 = \langle u_1; u_2; \dots; u_{k-1}; u_k \rangle$ and $\ell_2 = \langle v_1; v_2; \dots; v_{h-1}; v_h \rangle$, we define the right parallel product by $\ell = \ell_1 \parallel_r \ell_2$ where

- if $k > h$, then $\ell = \langle u_1; u_2; \dots; u_{k-h}; u_{k-h+1}v_1; \dots; u_k v_h \rangle$.
- if $h = k$, then $\ell = \langle u_1 v_1; u_2 v_2; \dots; u_k v_h \rangle$.
- if $k < h$, then $\ell = \langle u_1; u_2; \dots; v_{h-k}; u_1 v_{h-k+1}; \dots; u_k v_h \rangle$

This could equivalently be defined by $\ell_1 \parallel_r \ell_2 = (\ell_2^r \parallel \ell_1^r)^r$. In this parallel product, the product of words starts at the right end instead of the beginning in the original parallel product. Clearly, if L is a fair list language, its reversal is a language obtained in the same way from finite languages by using the operations \parallel_r and \parallel_r^* instead of \parallel and \parallel^* . Hence, we naturally get two different but dual families of list languages.

Denote the family of fair list languages by \mathcal{F}_ℓ and denote the new one by \mathcal{F}_r . It is obvious that these two families will share the same properties. In particular none of them is closed under reversal. However, if we look at rational list languages as defined in [?], they do coincide in both families. Given now the left and right parallel product, it is natural to introduce the family of list languages $\mathcal{F}_{\ell,r}$ as the family of list languages obtained from finite list languages by using the operations \cup , $.$, \star , \parallel , \parallel_r , \parallel^* and \parallel_r^* . Of course, the family $\mathcal{F}_{\ell,r}$ will be closed under reversal. Moreover, it will share all the properties of the \mathcal{F}_ℓ family given in [?]. Again, in particular, the rational $\mathcal{F}_{\ell,r}$ languages will be the same than the rational \mathcal{F}_ℓ languages.

Another natural idea is to introduce a strict parallel product $\parallel\!\!\parallel$, defined in the following way : for two lists $\ell_1 = \langle u_1; u_2; \dots; u_{k-1}; u_k \rangle$ and $\ell_2 = \langle v_1; v_2; \dots; v_{h-1}; v_h \rangle$, their *strict parallel product* $\ell = \ell_1 \parallel\!\!\parallel \ell_2$ is defined only if $k = h$ (i.e. if the two lists have the same length), and in this case

$\ell = \ell_1 \parallel \ell_2 = \ell_1 \parallel_r \ell_2$. As clearly $(\ell_1 \parallel \ell_2)^r = (\ell_2^r \parallel \ell_1^r)$, the family obtained from finite list languages by the operations \cup , \cdot , \star , \parallel and \parallel^* is closed under reversal. We denote this family by \mathcal{F}_s .

We can easily check that

Lemma 3.3 *The family $\mathcal{F}_{\ell,r}$ is contained in the family \mathcal{F}_s .*

Proof : Set $E = \{\langle \epsilon \rangle\}^*$. The following two identities express parallel product in terms of strict parallel product

$$L \parallel M = L.E \parallel M \cup L \parallel M.E, \quad L \parallel_r M = E.L \parallel M \cup L \parallel E.M$$

From these we immediately get

$$L^{\parallel^*} = \{\langle \rangle\} \cup L \parallel (L.E)^{\parallel^*}, \quad L^{\parallel_r^*} = \{\langle \rangle\} \cup L \parallel (E.L)^{\parallel^*} \quad \blacksquare$$

The aim is now to prove that the inclusion is strict. This is achieved in a similar way to that used for proving proposition 3.2. We consider again unary list languages (described as sequences of integers). A unary list language L is *p-almost constant* if there exists an integer p such that for any list $\ell = \langle n_1; n_2; \dots; n_k \rangle \in L$, we have $n_p = n_{p+1} = \dots = n_{k-p}$. This means that a unary list language is almost constant if it is of length less than $2p$ or, if it is constant up to a p terms at each end of the list. A unary p -almost constant list language L is unbounded if, moreover, given two integers q and N , there exists a list $\langle n_1; \dots; n_k \rangle$ in L with more than q elements and such that $n_p \geq N$. A unary almost constant and unbounded list language will be called a AC list language. We then get a lemma similar to Lemma 3.1 :

Lemma 3.4 *A list language in $\mathcal{F}_{\ell,r}$ cannot be a AC list language.*

Proof : It is similar to the proof of Lemma 3.1 ■

This lemma can be used to show

Proposition 3.5 *The family $\mathcal{F}_{\ell,r}$ is a strict subfamily of the family \mathcal{F}_s .*

Proof : We show that the family \mathcal{F}_s contains an AC list language. Consider the list language

$$L = (\langle a \rangle^*)^{\parallel^*}.$$

Then

$$L = \{\langle n, n, \dots, n \rangle \mid n \geq 0\}.$$

which is obviously an AC list language. By Lemma 3.3, the language L is not in $\mathcal{F}_{\ell,r}$. ■

Remark : The example L of the proof gives a new example that the Parikh image over $A \cup \{;\}$ of a language in \mathcal{F}_s may not be semi-linear.

4 Complements

Clearly, the various parallel products considered here (left, right, strict) do not cover all parallel products that could be invented. For instance, given two lists $\ell = \langle u_1; \dots; u_k \rangle$ and $\ell' = \langle v_1; \dots; v_h \rangle$, one could do the product “somewhere” in the list, by

$$\ell \circ \ell' = \langle u_1; \dots; u_p; u_{p+1}v_1; \dots; u_{p+h}v_h; \dots; u_k \rangle$$

or “anywhere” by

$$\ell \circ \ell' = \langle u_1; \dots; u_{p_1}v_1; \dots; u_{p_h}v_h; \dots; u_k \rangle$$

The study of list languages could be motivated by arguments from practical computer science, such as parallel computations or scheduling of jobs by priority queues. However, it seems necessary, before considering any product more in depth, to get strong reasons proving that this product deserves to be studied.

Acknowledgement

The authors acknowledge helpful observations from the anonymous referees. One of them corrected an error in a definition that also corrected the proof of a lemma, the second has greatly contributed to improve the presentation.