Petri Nets, Counter Machines and parallel matching computations [1]

Stefano Crespi Reghizzi [2] Pierluigi San Pietro [2]

Abstract

For the classical family of languages recognized by Place-Transitions Petri Nets (or, equivalently, by Greibach's quasi-realtime partially blind counter machines), we propose a new implementation by means of matching finite-state computations, within the model of consensually regular languages (introduced by the authors). A Petri Net computation is mapped on multiple finite-states computations that must match in a precise sense. Such implementation maps the original place (i.e., counters) onto a multiset over the states of the finite-state machine. This approach leads to a new way of specifying Petri Nets languages by means of regular expressions that define matching computations.

1 Introduction

Petri Nets (PN) and Multi-counter Machines have been used since half a century to model computation and formal languages, yet their theory is less established than, say, the theory of pushdown machines and context-free (CF) languages. Several types exist depending on the operations permitted on places, counters, on determinism, and on other constraints on spontaneous moves and counter reversals (see [7,4,5] among others). This paper focuses on a rich family: the nondeterministic, real-time *partially blind* multi-counter machine [5], called *RT-PBLIND*. The machine has a finite state and one or more integer counters. All counters start from zero and can be incremented or decremented, but not tested for zero. Moreover, the machine crashes whenever it tries to decrement a zero counter. Acceptance is by final state and requires that all counters are back to zero. We deal with machines operating in real-time, but it is well known that the quasi-real time condition (i.e., machines read at least one input character every $c \geq 1$ moves) is equivalent to real-time. The family RT-PBLIND coincides with the family, called *CSS*, of languages computed by PN such that every transition is labeled with a terminal letter.

To study this family cutting across traditional classifications, we adopt a new approach, called the *consensual regular* model, whose initial inspiration was to model parallel synchronous language processes that interact and re-enforce each other [1,2,3]. As a bonus we obtain the possibility to specify counter languages by means of regular expressions that describe the interacting computational threads in a rather perspicuous way.

Our main result is that every RT-PBLIND language is consensually regular. The proof is based on a series of normalizations of RT-PBLIND machines, leading eventually to a machine model that can be simulated by a set of regular parallel threads that match.

Although the course is rather technical, in our opinion the final result is valuable, because it offers a new way of specifying PN or RT-PBLIND languages by means of regular

[1] Authors supported by MIUR project PRIN 2010LYA9RH-006. The complete version of this work is submitted to a journal.

[2] Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) and CNR-IEIIT,Politecnico di Milano, Piazza Leonardo da Vinci 32, Milano I-20133

expressions (under the consensual interpretation). We hope that this novel specification style will be convenient in the areas where counter machines are used.

2 Consensual Languages

Let Σ denote a finite terminal alphabet and ϵ the empty word. For a word w, $|w|$ denotes the length of w; the i-th letter of w is $w(i)$, $1 \leq i \leq |w|$, i.e., $w = w(1)w(2)\dots w(|w|)$. A *finite automaton* (FA) A is a tuple $(\Sigma, Q, \delta, q_0, F)$ where: Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is the state-transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of final states. If for every pair q, a, $|\delta(q,a)| \leq 1$ then A is deterministic (DFA), otherwise is nondeterministic (NFA).

We introduce consensual languages by means of an intuitive operational model based on NFA computations, highlighting its difference from the classical NFA. In contrast to an NFA that performs one of several possible accepting runs to recognize an input word w, to recognize "consensually", the NFA performs up to $|w|$ concurrent accepting computations, such that, for each letter $w(1), \dots, w(|w|)$, exactly one computation - the "leader", asserts the validity of the letter. The remaining computations - the "followers", give their consent to the letter chosen by the leader. Out of metaphor, such dual behavior follower/leader can be represented in the NFA transition graph by two different edge types, say, thin for follower and thick for leader; but we prefer to use instead two distinct copies of each letter $a \in \Sigma$: a dotted letter $\mathring{a}$ denotes a follower type move, and a plain (undotted) letter a a leader type move. We proceed to formalize.

Let $\mathring{\Sigma}$ be the alphabet obtained by *marking* each letter $a \in \Sigma$ as $\mathring{a}$, and call *double* alphabet the set $\widetilde{\Sigma} = \Sigma \cup \mathring{\Sigma}$. A language $R \subseteq \widetilde{\Sigma}^*$ is called a *base language*, because it supports the definition of a language over Σ, to be called consensual. Let R be recognized by a NFA $A = (\widetilde{\Sigma}, Q, \delta, q_0, F)$, called the *base* automaton, where, without loss of generality, the transition relation δ is total. For all $k > 0$, a k-tuple Q^k is called a *state vector*. The finite *state vector transition relation* of A, $\longrightarrow_{\mathrm{SV}_A} \subset Q^{\mathbb{N}} \times \Sigma \times Q^{\mathbb{N}}$, is defined, for $a \in \Sigma$, for all $k > 0$, and for all state vectors $\boldsymbol{V}, \boldsymbol{V'} \in Q^k$, as: $\boldsymbol{V} \xrightarrow{a}_{\mathrm{SV}_A} \boldsymbol{V'}$ if $\exists j, 1 \leq j \leq k$: $v'_j \in \delta(\boldsymbol{V}(j), a)$ and $\forall i \neq j, 1 \leq i \leq k$: $\boldsymbol{V'}(i) \in \delta(\boldsymbol{V}(i), \mathring{a})$.

This definition asserts that exactly one of the k computations reads a: its next state is in $\delta(\boldsymbol{V}(j), a)$, thus acting as *leader*; all others read $\mathring{a}$: their next state is in $\delta(\boldsymbol{V}(i), \mathring{a})$, i.e., they are acting as *followers*. We say that the leader *places* a and the followers *consent* to it.

Definition 2.1 A word $w \in \Sigma^*$ is *consensually recognized* by a NFA A over $\widetilde{\Sigma}$ if there exist two finite state vectors $\boldsymbol{V}_0 \subset {q_0}^{\mathbb{N}}$ and $\boldsymbol{V}_f \subset F^{\mathbb{N}}$ such that $\boldsymbol{V}_0 \xrightarrow{w}_{\mathrm{SV}_A} \boldsymbol{V}_f$. The set $\mathcal{C}(A)$ of all such words is the *language consensually recognized* by A. The family of *consensually regular* languages, denoted by CREG, is the collection of all languages $\mathcal{C}(A)$ such that A is an NFA over $\widetilde{\Sigma}$.

Family CREG is known to be in NLOGSPACE, to include all regular languages and several non-semilinear languages, and to be incomparable with the CF family.

Fig. 1 shows an NFA recognizing the (non semilinear) language $L = \{ba^1ba^2 \dots ba^k \mid k \geq 1\}$, together with an example of consensual computation for $babaa$, which results for instance from the two finite-state computations: $q_1 \xrightarrow{\mathring{b}} q_2 \xrightarrow{a} q_3 \xrightarrow{\mathring{b}} q_2 \xrightarrow{a} q_3 \xrightarrow{\mathring{a}} q_3$ and $q_1 \xrightarrow{\mathring{b}} q_1 \xrightarrow{\mathring{a}} q_1 \xrightarrow{b} q_2 \xrightarrow{\mathring{a}} q_2 \xrightarrow{a} q_3$. For instance, at step three, the latter computation is the leader placing letter b, whereas at step four the latter computation leads and places letter a.

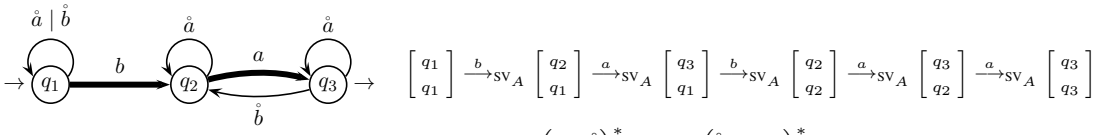

Fig. 1. An NFA A (left) accepting the base language $L(A) = \left(\mathring{a} \cup \mathring{b}\right)^* b\mathring{a}^* a\mathring{a}^* \left(b\mathring{a}^* a\mathring{a}^*\right)^*$, and consensually recognizing $\{ba\, ba^2 \dots ba^k \mid k \geq 1\}$, together with an example (right) of a state-vector relation for $babaa$. Leader (follower) transitions in A are rendered as thick (thin) edges.

It is important to observe that the language consensually recognized by an NFA A does not depend on A itself, but only on the (regular) *base* language $R = L(A)$. Therefore, a consensual language can be specified also by a *regular expression over the double alphabet*. For instance, the language $\{a^n b^n c^n \mid n > 0\}$ is defined by the base $\mathring{a}^*\, a\, \mathring{a}^*\, \mathring{b}^*\, b\, \mathring{b}^*\, \mathring{c}^*\, c\, \mathring{c}^*$. Since consensual languages do not depend on the particular automaton defining a base, it is also possible to give a different, but equivalent, formal definition of $\mathcal{C}(R)$, relying on a binary relation, called *match*, over $\widetilde{\Sigma}^*$; for details see [2].

3 Main Result

We briefly recall here the classic definitions of counter machines, as in [5]. A nondeterministic, *real-time, partially blind, multi-counter machine* (RT-PBLIND) is a tuple $\mathcal{M} = \langle \Sigma, S, \gamma, m, s_0, S_{fin} \rangle$, where: S is a finite set of *states*, $s_0 \in S$ is *initial*, and $S_{fin} \subseteq S$ is the set of *final* states; $m \geq 0$ is the number of counters; $\gamma \subseteq S \times \Sigma \times \{-1, 0, 1\}^m \times S$ is a set of *transitions*, called the transition relation.

A *configuration* of $\mathcal{M}$ is a pair in $S \times \mathbb{N}^m$. Let $\overrightarrow{0}^m \in \mathbb{N}^m$ be $(0, \dots, 0)$: the *initial* configuration is $(s_0, \overrightarrow{0}^m)$; a *final* configuration has the form $(s, \overrightarrow{0}^m)$, with $s \in S_{fin}$. A *move* of $\mathcal{M}$ is an element of relation $\longrightarrow_{\mathcal{M}} \subseteq (S \times \mathbb{N}^m) \times \Sigma \times (S \times \mathbb{N}^m)$ defined, $\forall a \in \Sigma, \forall s_i, s_j \in S, \forall$ vectors $\overrightarrow{X}_i, \overrightarrow{X}_j \in \mathbb{N}^m$, as: $(s_i, \overrightarrow{X}_i) \xrightarrow{a}_{\mathcal{M}} (s_j, \overrightarrow{X}_j)$ if $\exists \overrightarrow{Y} \in \{-1, 0, 1\}^m \mid (s_i, a, \overrightarrow{Y}, s_j) \in \gamma$ and $\overrightarrow{X}_j = \overrightarrow{X}_i + \overrightarrow{Y}$. Notice that an element in $\{-1, 0, 1\}^m$ is interpreted as an *increment vector* $\overrightarrow{Y} \in \mathbb{N}^m$. A sequence $(s_0, \overrightarrow{0}^m) \xrightarrow{w(1)}_{\mathcal{M}} \cdots \xrightarrow{w(n)}_{\mathcal{M}} (s_n, \overrightarrow{X}_n)$ is called a *run* of $\mathcal{M}$ with *label* w; the run is *accepting* if $(s_n, \overrightarrow{X}_n)$ is a final configuration. The *language accepted* by $\mathcal{M}$ is the set $L(\mathcal{M})$ of labels of accepting runs.

Consensual simulation of deterministic RT-PBLIND In [3], we defined a DFA A on the double alphabet $\widetilde{\Sigma}$ for every *deterministic* RT-PBLIND $\mathcal{M} = \langle \Sigma, S, \gamma, m, s_0, S_{fin} \rangle$ such that $\mathcal{C}(A) = L(\mathcal{M})$, but only when $\mathcal{M}$ is in *simple operation form* (s.o.), i.e., at every step there may be at most one increment and one decrement of different counters.

In this case, define A to be composed of m copies, numbered $1, \dots, m$, of the transition graph γ of $\mathcal{M}$. At every instant, a configuration $(s, \overrightarrow{X})$ is simulated by the matching computations of A as follows: every computation must be in one of the states $s^1, s^2, \dots, s^m$, where each s^i is the i-th copy of state s; the number of matching computations of A that are in s^i is equal to the value $X(i)$ of counter i of $\mathcal{M}$. The transitions of A are defined accordingly. For instance, suppose that $(s, a, \overrightarrow{Y}, r) \in \gamma$, with $\overrightarrow{Y}$ incrementing a counter h and decrementing a counter j: for every copy s^i, r^i of s, r, in A there is a (follower) transition from s^i to r^i while reading $\mathring{a}$, i.e., all the follower computations in state s^i go to state r^i for every i, $1 \leq i \leq m$; also, there is a (leader) transition from s^h to r^j while reading a, i.e., exactly one computation is transferred from s^h to r^j. Therefore, a leader transition is used to increment (or decrement) one counter, thus the need for the s.o. form since only one leader transition can be taken. For instance, the simultaneous increment of two counters

cannot be represented in this construction. For the simulation to work, moreover, no other transition of the form $(s, a, \overrightarrow{Y}', t)$ must be in γ for $t \neq r$, otherwise, for all i, there may be some consensual computation entering state r^i, and some other entering state t^i, violating the condition that the value of counter i is "stored" as computations in the copies of one single state. Hence, the result in [3] applies only to deterministic RT-PBLIND machines, in s.o. form.

Quasi-deterministic RT-PBLIND machines Looking carefully at the above simulation it is possible to notice that, although developed for the deterministic case, it does actually work even for some weak form of nondeterminism, called *nondeterministic counter operations*, namely when both $(s, a, \overrightarrow{Y}, r), (s, a, \overrightarrow{Y}', t) \in \gamma$, with $\overrightarrow{Y} \neq \overrightarrow{Y}'$ but $r = t$, since the consensual mechanism allows a nondeterministic choice of one leader transition among many spanning over the same pair of states. This contrasts with a *nondeterministic state choice*, when $r \neq t$, which cannot be simulated by the above model. We call then *quasi-deterministic* (QD) a nondeterminininistic RT-PBLIND machine without nondeterministic state choices (but possibly with some nondeterministic counter operations).

Lemma 3.1 *The language of a QD RT-PBLIND in s.o. form is consensually regular.*

In general, if the s.o. form is not imposed, a nondeterministic RT-PBLIND machine with nondeterministic state choices is easily simulated by a QD RT-PBLIND, simply by adding n extra counters to mimic the n states of the finite control: the new counter i is equal to 1 if, and only if, the original machine is in state s_i. Hence, the machine may perform the counter operations of the original machine and at the same time use an increment and a decrement on the counters representing the states (e.g.. increment counter h and decrement counter i to represent a transition going from state s_i to state s_h).

What is new and less obvious is that the same property holds also for QD RT-PBLIND machines *in s.o. form*, which, of course, mutually exclude the operations on the the original counters and on the counters representing states.

The proof of this property involves several transformations, all preserving generality, of the original RT-PBLIND machine:

- scheduling of the operations on each counter at precise times, using a *modulo-scheduling* function, so that each counter has different residue;
- h-rarefaction of a counter operation, so it is at distance at least $h > 1$ from other ones.

It is known at least since [4] (their Th. 1.2) that counter operations can be scheduled and rarefied by storing intermediate counter values into extra states. The construction of a "rarefied" RT counter machine is trivial when it is possible to test a counter for zero, e.g., if we want to have increments or decrements of a counter i only at positions multiple of an integer $v > 1$, modulo a residue r, one can store in the counter $\overrightarrow{X}'$ the value $\lfloor \frac{\overrightarrow{X}(i)}{v} \rfloor$ instead, while storing $\overrightarrow{X}(i) \mod v$ in the finite control: the value of the original counter X is thus given by $\overrightarrow{X}' * v + \overrightarrow{X}(i) \mod v$. This construction requires zero testing of the counter, which is not allowed in partially blind machines, so a lengthier and more subtle construction was devised, resulting in the following lemma.

Lemma 3.2 *For all RT-PBLIND machines and all $h > 1$ there exists an equivalent h-rarefied RT-PBLIND whose counters are scheduled with a module v and different residues.*

A scheduled and rarefied machine can be simulated by a QD RT-PBLIND in s.o. form,

in spite of its limited counter capabilities: the machine is simple-operation quasi-deterministic since residues (encoded in the finite state control) are used to control increment and decrements of the various counters, and all the residues are different from each other; to make it in s.o. form, the rarefaction allows the machine to guess the correct next state when making a counter operation and then crashing if the correct state was guessed incorrectly.

However, we have to assume that the input word is terminated by an end marker, denoted by symbol $\dashv \notin \Sigma$, since otherwise the real-time constraint would prevent the QD machine to terminate correctly in some cases.

Lemma 3.3 *For every 3-rarefied RT-PLIND machine $\mathcal{M}$, there exists a QD-RT-PBLIND machine accepting $L(\mathcal{M})\,\dashv$.*

By combining the above Lemmata, the main result of this paper follows immediately:

Theorem 3.4 *For all RT-PLIND machine $\mathcal{M}$, language $L(\mathcal{M})\,\dashv$ is consensually regular.*

The inclusion of RT-PBLIND in CREG is strict, since the language $\{a^n b^n \mid n \geq 1\}^*$ is not in RT-PBLIND [5], but it is in CREG, e.g., it is consensually defined by the regular base $(\mathring{a} \cup \mathring{b})^* \mathring{a}^*\, a\, \mathring{a}^*\, \mathring{b}^*\, b\, \mathring{b}^* (\mathring{a} \cup \mathring{b})^*$.

4 Conclusion

The interest for machines using integer counters is almost as old as for the Chomsky's families of languages, dating back at least to Minsky's work [7]. Yet, unlike the latter, multi-counter languages are not associated with grammars or with other types of declarative specification. Our research contributes a new normal form for nondeterministic real-time partially blind machines, i.e., for Petri Net languages. Exploiting the normal form, we have been able to simulate such multi-counter machines on the very different device of consensually regular languages. We observe that the simulation essentially involves a transformation from a sequential, but more complex, device to a parallel, but simpler, one: a single sequential computation is converted to multiple threads specified by a finite automaton, which are step-by-step synchronized by the consensual match function. As a consequence, it becomes possible to specify RT-PBLIND (and *a fortiori* reversal bounded machines [6]) or Petri Net languages by means of regular expressions, a notation we (subjectively) find quite readable and amenable to language transformation and composition, as exemplified by our past work on consensual languages.

References

[1] S. Crespi Reghizzi and P. San Pietro. Consensual definition of languages by regular sets. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *LATA 2008*, volume 5196 of *LNCS*, pages 196–208. Springer, 2008.

[2] S. Crespi Reghizzi and P. San Pietro. Consensual languages and matching finite-state computations. *RAIRO - Theor. Inf. and Applic*, 45(1):77–97, 2011.

[3] S. Crespi Reghizzi and P. San Pietro. Deterministic counter machines and parallel matching computations. In S. Konstantinidis, editor, *CIAA 2013*, volume 7982 of *LNCS*, pages 280–291. Springer, 2013.

[4] P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Math. Syst. Theory*, 2(3):265–283, 1968.

[5] S. A. Greibach. Remarks on the complexity of nondeterministic counter languages. *Theor. Comp. Sc.*, 1(4):269–288, Apr. 1976.

[6] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[7] M. Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Annals of Math.*, 74:3:437–455, 1961.