Suna Bensch, Rudolf Freund, and Friedrich Otto (eds.)

Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014)

Short Papers

Preface

The *Sixth Workshop on Non-Classical Models of Automata and Applications* (NCMA 2014) has been organized to bring together researchers who work on various aspects of non-classical and classical models of automata, providing an excellent opportunity to develop and discuss novel ideas. Numerous models of automata, both classical and non-classical, are natural objects of theoretical computer science. They are studied from different points of view in various areas, both as theoretical concepts and as formal models for applications. The purpose of the NCMA workshop series is to promote a deeper and interdisciplinary coverage of this particular area and in this way foster new insights and substantial progress in computer science as a whole.

The first workshop on Non-Classical Models of Automata and Applications, NCMA 2009, was held in Wrocław, Poland, in 2009 as a satellite event of the 17th International Symposium on Fundamentals of Computation Theory (FCT 2009). It was sponsored by the AutoMathA project of the European Science Foundation (ESF). The second workshop, NCMA 2010, was held in Jena, Germany, as an associated workshop of the 11th International Conference on Membrane Computing (CMC 11), and the third workshop, NCMA 2011, was organized at the Università degli Studi di Milano, Milan, Italy, in close proximity to the 15th Conference on Developments in Language Theory (DLT 2011). In contrast to the previous workshops, the fourth workshop, NCMA 2012, and the fifth workshop, NCMA 2013, were organized as stand-alone workshops. NCMA 2012 was held in Fribourg, Switzerland, in August 2012, and NCMA 2013 took place in Umeå, Sweden, in August 2013. Now the Sixth Workshop on Non-Classical Models of Automata and Applications (NCMA 2014) takes place in Kassel, Germany, on July 28 and 29, 2014, right before the 19th International Conference on Implementation and Application of Automata (CIAA 2014) in Giessen. We expect it to be again a scientifically valuable event with interesting presentations, exciting results, and stimulating discussions. In addition to invited talks and contributed full papers, NCMA 2014 also features six short presentations to emphasize its workshop character; the extended abstracts of the short presentations are contained in this booklet. We hope that the friendly atmosphere and the nice surroundings will also help to make this workshop a worthwhile event that will lead to new insights and possibly new cooperations.

At NCMA 2014 there are two invited presentations by

- Peter Leupold (University of Leipzig, Germany) and
- František Mráz (Charles University in Prague, Czech Republic),

and 14 presentations of contributed full papers. The invited presentations and the full papers are published in the proceedings of NCMA 2014, which appear as volume 304 in the series books@ocg.at of the Austrian Computer Society.

We are grateful to the University of Kassel for the support in the local organization of NCMA 2014, to the Deutsche Forschungsgemeinschaft (DFG) for financial support, and to the Institute of Computer Languages of the Vienna University of Technology for covering the printing costs of this booklet of short papers, as well as to Gernot Salzer (Vienna University of Technology, Austria) for his useful advice with editing these volumes.

July 2014

Suna Bensch, Umeå
Rudolf Freund, Vienna
Friedrich Otto, Kassel

Table of Contents

VISIBLY PUSHDOWN AUTOMATA AND TRANSDUCERS WITH COUNTERS

Oscar H. Ibarra

Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract
We generalize the models of visibly pushdown automata (VPDAs) and visibly pushdown transducers (VPDTs) by equipping them with reversal-bounded counters. We show that some of the results for VPDAs and VPDTs (e.g., closure under intersection and decidability of emptiness for VPDA languages) carry over to the generalized models, but other results (e.g., determinization and closure under complementation) do not carry over. We also investigate the finite-ambiguity and finite-valuedness problems concerning these devices.

Keywords: pushdown automaton, pushdown transducer, visibly pushdown automaton, visibly pushdown transducer, reversal-bounded counters, ambiguous, finite-valued, decidable, undecidable.

1 Introduction

A visibly pushdown automaton (VPDA), which was introduced in [1], is a restricted version of a nondeterministic pushdown automation (NPDA). In a VPDA, the input symbol determines when the automaton can push or pop. It was shown in [1] that the class of languages accepted by VPDAs enjoys many of the properties of regular languages. For example, the class is closed under union, intersection, complementation, renaming, concatenation, and Kleene-*. Unlike NPDAs, the containment and equivalence problems are decidable for VPDAs. It was shown in [1] that the VPDA framework unifies and generalizes many of the decision procedures in the program analysis literature, and it provides algorithmic verification of recursive programs with respect to many context-free properties. Since its introduction, many papers have been written concerning VPDAs and their variants/extensions. One, in particular, is the visibly pushdown transducer (VPDT), which is simply a VPDA with outputs [12, 4].

In this paper, we generalize the models of VPDAs and VPDTs by equipping them with reversal-bounded counters. We investigate the closure and decidability properties of these generalized models. In particular, we show that some of the results for VPDAs and VPDTs (e.g., closure under intersection and decidability of emptiness for VPDA languages) carry over to languages

Supported in part by NSF Grant CCF-1117708.

accepted by VPDAs with reversal-bounded counters, but other results (e.g., determinization and closure under complementation) do not carry over. We also look at the ambiguity questions concerning VPDAs with reversal-bounded counters and the finite-valuedness problem for VPDTs with reversal-bounded counters.

No proofs are provided in this paper. The results reported here and their proofs are part of an invited paper, "Automata with Reversal-Bounded Counters: A Survey", which will be presented at DCFS 2014.

2 Preliminaries

A counter is an integer variable that can be incremented by 1, decremented by 1, and tested for zero. It starts at zero and can only store nonnegative integer values. (Thus, one can think of a counter as a pushdown stack with a unary alphabet, in addition to the bottom of the stack symbol which is never altered.)

An automaton (DFA, NFA, DPDA, NPDA, etc.) can be augmented with multiple counters, where the "move" of the machine also now depends on the status (zero or non-zero) of the counters, and the move can update the counters. See [9] for formal definitions.

It is well known that a DFA augmented with two counters is equivalent to a Turing machine (TM) [11]. However, when we restrict the operation of the counters so that during the computation, the number of times each counter alternates between nondecreasing mode and nonincreasing mode is at most some fixed number r (such a counter is called *reversal-bounded*), the computational power of a DFA (and even an NPDA) augmented with reversal-bounded counters is significantly weaker than a TM. Note that a counter that makes r reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ 1-reversal counters.

We will use the following notations: DCM (resp., NCM, DPCM, NPCM) will denote a DFA (resp., NFA, DPDA, NPDA) augmented with a finite number of reversal-bounded counters.

The following fundamental result was shown in [9]:

Theorem 2.1. *The emptiness and infiniteness problems for NPCMs are decidable.*

Note: All proofs are omitted in this short paper. The proofs will appear elsewhere.

3 VPDAs with Reversal-Bounded Counters

The model of a visibly pushdown automaton (VPDA) was first introduced and studied in [1]. It is an NPDA where the input symbol determines the (push/stack) operation of the stack. The input alphabet Σ is partitioned into three disjoint alphabets: $\Sigma_c, \Sigma_r, \Sigma_{int}$. The machine pushes a specified symbol on the stack if it reads a call symbol in Σ_c on the input; it pops a specified symbol (if the specified symbol is at top of the stack) if it reads a return symbol in Σ_r on the input; it does not use the (top symbol of) the stack and can only change state if it reads an internal symbol in Σ_{int} on the input. The machine has no ε-moves, i.e, it reads an input at every step. An input $x \in \Sigma^*$ is accepted if the machine, starting from one of a designated set of initial states with a distinguished symbol $\perp$ at bottom of the stack (which is never altered), eventually enters an accepting sate after processing all symbols in x. For details, see [1]. In this paper, we assume without loss of generality, that the VPDA has only one initial state.

A 1-ambiguous NPDA is one where every input is accepted in at most one accepting computation. A 1-ambiguous NPDA is more powerful than a visibly pushdown automaton (VPDA) since the former can accept languages, like $L_1 = \{x\#x^R \mid x \in (a+b)^+\}$ and $L_2 = \{a^kba^k \mid k \geq 1\}$, that cannot be accepted by the latter. There are languages accepted by 1-ambiguous NPDAs that are not deterministic context-free languages (DCFLs), whereas languages accepted by VPDAs are DCFLs [1]. Note also that a 1-ambiguous NPDA can have ε-moves.

Now consider a VPDA augmented with k reversal-bounded counters. We allow the machine to have ε-moves, but in such moves, the stack is not used, only the state and counters are used and updated. Acceptance of an input string is when machine eventually falls off the right end of the input in an accepting state. Thus, a VPDA M with k reversal-bounded counters operates like a VPDA but can now use reversal-bounded counters as auxiliary memory. More precisely, M's operation is defined as follows (where q, p are states, $s_i = 0$ or $+$ is the status of counter i, and $d_i = +1$, -1, or 0 is added to counter i):

1. If the current input is $a \in \Sigma_c$, then M can only use rules of the form:
   $(q, a, \gamma, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$
   which means that γ (which cannot be $\perp$) is pushed, and the state and counters are updated, or of the form:

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$
   which means an ε-move: the input head is not moved, the stack is not used, but the state and counters are updated.

2. If the current input is $a \in \Sigma_r$, then M can only use rules of the form:
   $(q, a, \gamma, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$
   which means that if γ is top of the stack and $\neq\perp$, it is popped (otherwise, if $\gamma =\perp$, it is not popped), and the state and counters are updated, or of the form :

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$

3. If the current input is $a \in \Sigma_{int}$, then M can only use rules of the form:
   $(q, a, s_1 \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$
   which means the stack is not used, but the state and counters are updated, or of the form:

   $(q, \varepsilon, s_1, \ldots, s_k) \rightarrow (p, d_1, \ldots, d_k)$

Consider the language $L_1 = \{w \mid w = xy$ for some $x \in (a+b)^+, y \in (0+1)^+, x$ when a, b are mapped to $0, 1$, respectively, is the reverse of $y\}$. Clearly, L_1 can be accepted by a VPDA M_1.

However, the language $L_2 = \{w \mid w \in L_1$, the number of a's + number of 0's in w = the number of b's + number of 1's in $w\}$ cannot be accepted by a VPDA. But L_2 can be accepted by a VPDA M_2 with two 1-reversal counters C_1 and C_2 as follows: On a given input w, M_2 simulates M_1, and stores the number of a's and 0's (resp., the number of b's and 1's) it sees on the input in counters C_1 and C_2, respectively. Then when M_1 accepts, M_2 (on ε-moves) decreases the counters simultaneously and accepts if the counters become zero at the same time.

Denote a VPDA augmented with a finite number of reversal-bounded counters by VPCM. Clearly, VPCMs are a special case of NPCMs. Hence, from Theorem 2.1, we have:

Theorem 3.1. *The emptiness and infiniteness problems for VPCMs are decidable.*

We can show:

Theorem 3.2. *The class of languages accepted by VPCMs is closed under union, intersection, renaming, concatenation, Kleene-*.*

With respect to complementation, unlike for VPDAs, we have:

Theorem 3.3. *The class of languages accepted by VPCMs is not closed under complementation.*

Every VPDA can be converted to an equivalent deterministic VPDA [1]. In contrast, from Theorem 3.3, we have:

Corollary 3.4. *There are VPCMs that cannot be converted to equivalent deterministic VPCMs.*

We can define a deterministic VPCM (DVPCM) as follows. For every tuple $(q, \gamma, s_1, \ldots, s_k)$:

1. For every $a \in \Sigma \cup \{\varepsilon\}$, there is at most one transition $(q, a, \gamma, s_1, \ldots, s_k) \to (p, d_1, \ldots, d_k)$.
2. If there is a transition $(q, \varepsilon, \gamma, s_1, \ldots, s_k) \to (p, d_1, \ldots, d_k)$, then there is no transition $(q, a, \gamma, s_1, \ldots, s_k) \to (p, d_1, \ldots, d_k)$ for any $a \in \Sigma$.

We also assume that the machine always halts. Thus a DVPCM is a deterministic VPDA with reversal-bounded counters. (Note that a VPDA has no ε-transitions.) Since a VPDA can always be made deterministic, it follows that DVPCMs are strictly more powerful than deterministic VPDAs.

The first part of the next result is obvious. The second part follows from the first using Theorem 3.1.

Corollary 3.5.

1. *The class of languages accepted by DVPCMs is closed under Boolean operations.*
2. *The containment and equivalence problems for DVPCMs are decidable.*

The second part of the above corollary does not hold for VPCMs, since the universe problem (does a given transducer accept all strings?) is already undecidable even for an NFA augmented with one 1-reversal counter [2]. A VPCM is k-ambiguous ($k \geq 1$) if every input is accepted in at most k distinct accepting computations. We can show:

Theorem 3.6. *It is decidable, given a VPCM M and $k \geq 1$, whether M is k-ambiguous.*

Note that the above contrasts the undecidability of k-ambiguity (even for $k = 1$) for NPDAs whose stack makes at most one reversal: when it pops, it can no longer push.

4 VPDTs with Reversal-Bounded Counters

Visibly pushdown transducers (VPDT) were introduced in [12], where ε-transitions that can produce outputs were allowed. Allowing such transitions makes some decision problems (e.g., single-valuedness) undecidable. Later, in [4], VPDTs that do not allow ε-transitions were investigated, where it was shown that the k-valuedness problem for VPDTs is decidable. (A transducer is k-valued if every accepted input generates at most k distinct outputs. It is finite-valued if it is k-valued for some k.)

We can generalize the result above for VPDTs with reversal-bounded counters. A VPCMT T is a VPCM with outputs. Since a VPCM is allowed ε-transitions (where the stack is not used), we assume that on an ε-transition, the VPCMT can only output ε.

Theorem 4.1. *It is decidable, given a VPCMT T and $k \geq 1$, whether T is k-valued.*

5 Containment and Equivalence Problems for VPCMTs

It is known that the equivalence problem for two-way deterministic finite transducers (2DFTs) is decidable [6]. However, if nondeterminism is allowed, the problem becomes undecidable even for one-way nondeterministic finite transducers (NFTs) [5]. The undecidability holds even for NFTs operating on a unary input (or output) alphabet [10]. For single-valued (i.e., 1-valued) NFTs, the problem becomes decidable [7], and the decidability result was later extended to finite-valued NFTs in [3]. The complexity of the problem was subsequently derived in [13].

Now consider the containment and equivalence problems for VPCMTs and DVPCMTs. As we mentioned earlier, the containment and equivalence problems for NFTs is undecidable. Hence these problems are also undecidable for VPCMTs. However, we can prove:

Theorem 5.1. *The following problems are decidable:*

1. *Given a VPCMT T_1 and a DVPCMT T_2, is $R(T_1) \subseteq R(T_2)$?*
2. *Given two DVPCMTs T_1 and T_2, is $R(T_1) = R(T_2)$?*

6 Conclusion

The emptiness problem for NPDAs with reversal-bounded counters has recently been shown to be NP-complete [8]. Using this result, we can derive lower and upper bounds for many decidable problems discussed in the paper. We intend to do this in the journal version of the paper. There are a number of interesting open questions that we are currently working on, two of which are:

1. Is it decidable, given a VPCM M, whether M is finitely-ambiguous (i.e., k-ambiguous for some k)?
2. Is it decidable, given a VPCMT T, whether it is finite-valued (i.e., k-valued for some k)?

References

[1] R. ALUR, P. MADHUSUDAN, Visibly Pushdown Languages. In: *Proc. STOC*. 2004, 202–211.

[2] B. BAKER, R. BOOK, Reversal-bounded Multipushdown Machines. *J. Comput. System Sci.* 8 (1974), 315–332.

[3] K. CULIK, J. KARHUMAKI, The equivalence of finite valued transducers (on HDTOL languages) is decidable. *Theoret. Comput. Sci.* 47 (1986), 71–84.

[4] E. FILIOT, J.-F. RASKIN, P.-A. REYNIER, F. SERVAIS, J.-M. TALBOT, Properties of visibly pushdown transducers. In: *Proc. of MFCS*. 2010, 355–367.

[5] T. GRIFFITHS, The unsolvability of the equivalence problem for Λ-free nondeterministic generalized sequential machines. *J. Assoc. Comput. Mach.* 15 (1968), 409–413.

[6] E. GURARI, The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM J. Comput.* 11 (1982), 448–452.

[7] E. GURARI, O. H. IBARRA, A note on finite-valued and finitely ambiguous transducers. *Math. Systems Theory* 16 (1983), 61–66.

[8] M. HAGUE, A. W. LIN, Model checking recursive programs with numeric data types. In: *Proc. of CAV*. 2011, 743–759.

[9] O. H. IBARRA, Reversal-bounded multicounter machines and their decision problems. *J. of the ACM* 25 (1978) 1, 116–133.

[10] O. H. IBARRA, The unsolvability of the equivalence problem for ε-free NGSM's with unary input (output) alphabet and applications. *SIAM J. Computing* 7 (1978), 524–532.

[11] M. MINSKY, Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.* 74 (1961), 437–455.

[12] J.-F. RASKIN, F. SERVAIS, Visibly pushdown transducers. In: *Proc. of ICALP*. 2008, 386–397.

[13] A. WEBER, Decomposing finite-valued transducers and deciding their equivalence. *SIAM. J. on Computing* 22 (1993), 175–202.

NEW SIZE HIERARCHIES FOR TWO-WAY NON-UNIFORM AUTOMATA

Kamil Khadiev$^{(A)}$ Abuzer Yakaryılmaz$^{(B)}$

$^{(A)}$Kazan Federal University, Kazan, Russia
kamilhadi@gmail.com

$^{(B)}$National Laboratory for Scientific Computing, Petrópolis, RJ, 25651-075, Brazil
abuzer@lncc.br

Abstract
In this paper, we introduce a new non-uniform two-way deterministic and nondeterministic automata, i.e. each tape square can induce a different transition function. For these models, we present some hierarchy results based on the sizes (the numbers of states) of automata where the gaps are linear and quadratic for deterministic and nondeterministic models, respectively. We obtain the results for the automata for which the size of the set of its states is less than the square root of the length of its input. We also enhance these automata by giving the capability of shuffling the symbols of a given input at the beginning of the computation with respect to a permutation. We prove that the same hierarchy results are also obtained for them. On the other hand, we prove that the original automata are incomparable with some smaller enhanced automata where the size of the enhanced one is constant times less than the original one for deterministic models and sub-quadratic size of the original one for nondeterministic models.

1 Introduction

There are many different models of non-uniform automata in the literature, e.g. [4, 7, 9, 3]. However, compared to the uniform ones, the size of non-uniform automata has been less investigated and there are still some open problems. For example, Sakoda and Sipser [13] conjectured that simulating a 2NFA by a 2DFA requires exponential number of states in the worst case. But, the best known separation is only quadratic ($O(n^2)$) [5, 8] and the researchers have succeeded to obtain better bounds only for some modified models, e.g. see [12, 10, 6, 11].

We investigate some hierarchies depending on the size of automata and so, for each input length, our models can have different number of states. Therefore, we define an automaton for a fixed non-negative integer n (the input length). Moreover, like Branching programs or the data-independent models defined by Holzer [7], the transition functions can be changed during the computation. Holzer's model can use a different transition function for each step.

$^{(A)}$Khadiev was partially supported by RFBR Grant 14-07-00557.

$^{(B)}$Yakaryılmaz was partially supported by CAPES, ERC Advanced Grant MQC, and FP7 FET project QALGO.

We restrict this property as follows: The transition function is the same for the same tape positions. Thus, we can have at most n different transition functions. If we restrict ourselves to a single transition function, then we obtain a "standard" non-uniform automaton. Our alphabet is a binary one, $\Sigma = \{0, 1\}$, and we use the terminologies of Branching programs: Our decision problems are solving/computing Boolean functions instead of recognizing languages. That is, the automaton must accept the inputs where the function gets the value true and rejects the inputs where the function gets the value false if this function is solved/computed by the automaton.

A non-uniform head-position-dependent two-way deterministic finite automaton working on the inputs of length/size of $n \geq 0$ (2DA$_n$) D_n is a 6-tuple $D_n = (\Sigma, S, s_1, \delta = \{\delta_1, \ldots, \delta_n\}, s_a, s_r)$, where (i) $S = \{s_1, \ldots, s_d\}$ is the set of states (d can be a function in n) and s_1, $s_a \in S$, and $s_r \in S$ ($s_a \neq s_r$) are the initial, accepting, and rejecting states, respectively; and, (ii) δ is a collection of n transition functions such that $\delta_i : S \setminus \{s_a, s_r\} \times \tilde{\Sigma} \to S \times \{\leftarrow, \downarrow, \to\}$ is the transition function that governs the behaviour of D_n when reading the ith symbol/variable of the input, where $1 \leq i \leq n$. Any given input $u \in \Sigma^n$ is placed on a read-only tape with a single head as u from the squares 1 to $|u| = n$. When D_n is in $s \in S \setminus \{s_a, s_r\}$ and reads $u_i \in \Sigma$ (the ith symbol of u) on the tape, it switches to state $s' \in S$ and updates the head as position with respect to $d \in \{\leftarrow, \downarrow, \to\}$ if $\delta_i(s, u_i) \to (s', d)$. If $d =$ " $\leftarrow$ " (" $\to$ "), the head moves one square to the left (the right), and, it stays on the same square, otherwise. The transition functions δ_1 and δ_n must be defined to guarantee that the head never leaves u during the computation. Moreover, the automaton enters s_a or s_r only on the right end-marker and then the input is respectively accepted or rejected.

A nondeterministic counterpart of 2DA$_n$, denoted 2NA$_n$, can choose from more than one transition in each step. So, the range of each transition function is $\mathcal{P}(S \times \{\leftarrow, \downarrow, \to\})$, where $\mathcal{P}(\cdot)$ is the power set of any given set. Therefore, a 2NA$_n$ can follow more than one computational path and the input is accepted only if one of them ends with the decision of "acceptance". Note that some paths may end without any decision since the transition function can yield the empty set for some transitions. A function $f_n : \Sigma^n \to \Sigma$ is said to be computed by a 2DA$_n$ D_n (a 2NA$_n$ N_n) if each member of $f_n^{-1}(1)$ is accepted by D_n (N_n) and each member of $f_n^{-1}(0)$ is rejected by D_n (N_n). The class $\mathsf{2DSIZE(d(n))}$ is formed by the functions $f = \{f_0, f_1, f_2, \ldots\}$ such that each f_i is computed by a 2DA$_i$ D_i whose number of states is no more than $d(i)$, where i is a non-negative integer. We can similarly define nondeterministic counterparts of this class, denoted $\mathsf{2NSIZE(d(n))}$.

We also introduce a generalization of our non-uniform models that can shuffle the input at the beginning of the computation with respect to a permutation. A non-uniform head-position-dependent shuffling two-way deterministic finite automaton working on the inputs of length/size of $n \geq 0$ (2DA$_n^\Theta$), say D_n^θ, is a 2DA$_n$ that shuffles the symbols of input with respect to θ, a permutation of $\{1, \ldots, n\}$, i.e. the jth symbol of the input is placed on $\theta(j)$th place on the tape ($1 \leq j \leq n$), and then execute the 2DA$_n$ algorithm on this new input. The nondeterministic model can be abbreviated as 2NA$_n^\Theta$. The class $\mathsf{2D\Theta SIZE(d(n))}$ is formed by the functions $f = \{f_0, f_1, f_2, \ldots\}$ such that each f_i is computed by a 2DA$_i^\Theta$ D_i^θ whose number of states is no more than $d(i)$, where i is a non-negative integer and θ is a permutation of $\{1, \ldots, n\}$. The nondeterministic class is represented by $\mathsf{2N\Theta SIZE(d(n))}$.

2 Summary of Results

We obtain the following hierarchy results for our deterministic models.

Theorem 1. *Let $d : \mathbb{N} \to \mathbb{N}$ be a non-constant function such that $d^2(n) < n$. Then, we have*

$$\mathsf{2DSIZE}\left(\left\lfloor \frac{\mathsf{d}-4}{13} \right\rfloor - 4\right) \subsetneq \mathsf{2DSIZE}(\mathsf{d}) \quad \textit{and} \quad \mathsf{2D\Theta SIZE}\left(\left\lfloor \frac{\mathsf{d}-4}{13} \right\rfloor - 4\right) \subsetneq \mathsf{2D\Theta SIZE}(\mathsf{d}).$$

We also show that our deterministic models are incomparable.

Theorem 2. *Let $d : \mathbb{N} \to \mathbb{N}$ be a non-constant function such that $d^2(n) < \frac{n}{2} - 1$. Then for any $d' : \mathbb{N} \to \mathbb{N}$ satisfying $4 \le d'(n) \le \left\lfloor \frac{d-4}{13} \right\rfloor - 4$, $\mathsf{2DSIZE}(\mathsf{d})$ and $\mathsf{2D\Theta SIZE}(\mathsf{d'})$ are incomparable.*

We can extend our results to nondeterministic models.

Theorem 3. *Let $d : \mathbb{N} \to \mathbb{N}$ be a non-constant function such that $d^2(n) < n$. Then, we have*

$$\mathsf{2NSIZE}\left(\left\lfloor \sqrt{\left\lfloor \frac{\mathsf{d}-4}{13} \right\rfloor - 4} \right\rfloor\right) \subsetneq \mathsf{2NSIZE}(\mathsf{d}) \quad \textit{and} \quad \mathsf{2N\Theta SIZE}\left(\left\lfloor \sqrt{\left\lfloor \frac{\mathsf{d}-4}{13} \right\rfloor - 4} \right\rfloor\right) \subsetneq \mathsf{2N\Theta SIZE}(\mathsf{d}).$$

Theorem 4. *Let $d : \mathbb{N} \to \mathbb{N}$ be a non-constant function such that $d^2(n) < \frac{n}{2} - 1$. For any $d' : \mathbb{N} \to \mathbb{N}$ satisfying $4 \le d'(n) \le \left\lfloor \sqrt{\left\lfloor \frac{d-4}{13} \right\rfloor - 4} \right\rfloor$, $\mathsf{2NSIZE}(\mathsf{d})$ and $\mathsf{2N\Theta SIZE}(\mathsf{d'})$ are incomparable.*

In our proofs, we use two witness Boolean functions: *Shuffled Address Function* (see Figure 1), denoted $\mathtt{2\text{-}SAF_w}$, which is a modification of Boolean function given in [1], and the "shuffled" version [2] of well known *Equality function* $\mathsf{EQ(X)} = \bigvee_0^{\lfloor n/2 \rfloor - 1} x_i = x_{i+\lfloor n/2 \rfloor}$.

First, we present some generic lower bounds for our models by using some similar techniques from communication complexity.

Lower bounds

Let $\pi = (\{x_{j_1}, \dots, x_{j_u}\}, \{x_{i_1}, \dots, x_{i_v}\}) = (X_A, X_B)$ be a partition of the set X into two parts X_A and $X_B = X \backslash X_A$ and $f|_\rho$ be the subfunction of Boolean function f (X), where ρ is a mapping $\rho : X_A \to \{0,1\}^{|X_A|}$. Function $f|_\rho$ is obtained from f by fixing the values of the variables $x_{j_1}, \dots, x_{j_u}$ to $\rho(x_{j_1}), \dots, \rho(x_{j_u})$. We denote $N^\pi(f)$ to be the number of different subfunctions with respect to partition π. Let $\Theta(n)$ be the set of all permutations of $\{1, \dots, n\}$.

We say that partition π agrees with permutation $\theta = (j_1, \dots, j_n) \in \Theta(n)$ if for some u $(1 < u < n)$, the following holds: $\pi = (\{x_{j_1}, \dots, x_{j_u}\}, \{x_{j_{u+1}}, \dots, x_{j_n}\})$. We denote $\Pi(\theta)$ to be the set of all partitions agreeing with θ. Moreover, we define

$$N^\theta(f) = \max_{\pi \in \Pi(\theta)} N^\pi(f) \quad \text{and} \quad N(f) = \min_{\theta \in \Theta(n)} N^\theta(f),$$

and, for any Boolean function f, $N^{id}(f)$ $(id = (1, \dots, n))$ is the analogue of the rank of language L_n (the number of Myhill-Nerode classes) such that f is the characteristic function of L_n.

$\texttt{2-SAF}_{\texttt{w}}(X) : \{0,1\}^n \to \{0,1\}$ for integer $w = w(n)$ such that

$$2w(2w + \lceil \log 2w \rceil) < n. \tag{1}$$

We divide the input variables (the symbols of the input) into $2w$ blocks. There are $\left\lfloor \frac{n}{2w} \right\rfloor = a$ variables in each block. After that, we divide each block into *address* and *value* variables. The first $\lceil \log 2w \rceil$ variables of block are *address* and the other $a - \lceil \log 2w \rceil = b$ variables of block are *value*.
We call $x_0^p, \dots, x_{b-1}^p$ and $y_0^p, \dots, y_{\lceil \log 2w \rceil}^p$ are the *value* and the *address* variables of the pth block, respectively, for $p \in \{0, \dots, 2w-1\}$.
Boolean function $\texttt{2-SAF}_{\texttt{w}}(X)$ can be calculated iteratively based on the following five functions:

1. Function $Adr : \{0,1\}^n \times \{0, \dots, 2w-1\} \to \{0, \dots, 2w-1\}$ gets the address of a block:

$$Adr(X,p) = \sum_{j=0}^{\lceil \log 2w \rceil - 1} y_j^p \cdot 2^j (mod\ 2w).$$

2. Function $Ind : \{0,1\}^n \times \{0, \dots, 2w-1\} \to \{-1, \dots, 2w-1\}$ gets the number of block by address:

$$Ind(X,i) = \begin{cases} p & \text{, where } p \text{ is the minimal number such that } Adr(X,p) = i, \\ -1 & \text{, if there are no such } p \end{cases}.$$

3. Function $Val : \{0,1\}^n \times \{0, \dots, 2w-1\} \to \{-1, \dots, w-1\}$ gets the value of the block which has the address i:

$$Val(X,i) = \begin{cases} \sum_{j=0}^{b-1} x_j^p (mod\ w) & \text{, where } p = Ind(X,i) \text{ for } p \geq 0, \\ -1 & \text{, if } Ind(X,i) < 0 \end{cases}.$$

For computing $\texttt{2-SAF}_{\texttt{w}}(X)$, we make two steps of iterations. Two functions $Step_1$ and $Step_2$ get the result of the tth step of iteration.

4. Function $Step_1 : \{0,1\}^n \times \{0, \dots, 1\} \to \{-1, w \dots, 2w-1\}$ gets the first part of the tth step of iteration:

$$Step_1(X,t) = \begin{cases} -1 & \text{, if } Step_2(X, t-1) = -1, \\ Val(X, Step_2(X, t-1)) + w & \text{, otherwise} \end{cases}.$$

5. Function $Step_2 : \{0,1\}^n \times \{-1, \dots, 1\} \to \{-1, \dots, w-1\}$ gets the second part of the tth step of iteration:

$$Step_2(X,t) = \begin{cases} -1 & \text{, if } Step_1(X,t) = -1, \\ 2 & \text{, if } t = -1 \\ Val(X, Step_1(X,t)) & \text{, otherwise} \end{cases}.$$

Remark that the address of the current block is computed in the previous step. The function $\texttt{2-SAF}_{\texttt{w}}(X)$ is computed as:

$$\texttt{2-SAF}_{\texttt{w}}(X) = \begin{cases} 0, \text{ if } Step_2(X,1) \leq 0, \\ 1, \text{ otherwise} \end{cases}.$$

Figure 1: The formal definition of $\texttt{2-SAF}_{\texttt{w}}$

Theorem 5. *If function $f(X)$ is computed by a 2DA$_n^\Theta$ of size d, say A_n^θ, for a permutation θ, then $N(f) \leq (d+1)^{d+1}$.*

Corollary 6. *If the function $f(X)$ is computed by a 2DA$_n$ of size d, then $N^{id}(f) \leq (d+1)^{d+1}$.*

Theorem 7. *If the function $f(X)$ is computed by a 2NA$_n^\Theta$ of size d, say A_n^θ, for a permutation θ, then $N(f) \leq 2^{d(d+1)}$.*

Corollary 8. *If function $f(X)$ is computed by a 2NA$_n$ of size d, then $N^{id}(f) \leq 2^{d(d+1)}$.*

With a $2DA_n$ computing $\mathtt{2\text{-}SAF_w}$ we will be able to prove our hierarchy results:

Lemma 9. *There is a 2DA$_n$ A_n of size $13w+4$ which computes* $\mathtt{2\text{-}SAF_w}$.

The Proof of Theorem 1: It is obvious that $\mathsf{2D\Theta SIZE}(\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4) \subseteq \mathsf{2D\Theta SIZE(d)}$ and $\mathsf{2DSIZE}(\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4) \subseteq \mathsf{2DSIZE(d)}$. By Lemma 9, we have that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor}$ is in both $\mathsf{2D\Theta SIZE(d)}$ and $\mathsf{2DSIZE(d)}$. After some certain calculations, we can show that

$$N\left(\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor}\right) > \left(\left\lfloor\frac{d-4}{13}\right\rfloor - 4 + 1\right)^{\lfloor\frac{d-4}{13}\rfloor - 4 + 1}.$$

Then, by Theorem 5, we infer that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4} \notin \mathsf{2DSIZE}(\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4)$ and $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4} \notin \mathsf{2D\Theta SIZE}(\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4)$.

The Proof of Theorem 2: Remember that $d' \leq \lfloor\frac{d-4}{13}\rfloor - 4$. We have already shown that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \notin \mathsf{2D\Theta SIZE}(\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4)$ and $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \in \mathsf{2DSIZE(d)}$. On the other hand, we can easily obtain that $N^{id}(\mathsf{EQ}) = 2^{\lfloor n/2\rfloor}$ and $N^\theta(\mathsf{EQ}) \leq 4$ for $\theta = (0, \lfloor n/2\rfloor, 1, \lfloor n/2\rfloor + 1, \dots)$.

Therefore, $\mathsf{EQ} \in \mathsf{2D\Theta SIZE(4)}$ but, due to Corollary 6, we have $\mathsf{EQ} \notin \mathsf{2DSIZE(d')}$.

The Proof of Theorem 3: It is obvious that $\mathsf{2N\Theta SIZE}\left(\left\lfloor\sqrt{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4}\right\rfloor\right) \subseteq \mathsf{2N\Theta SIZE(d)}$ and $\mathsf{2NSIZE}\left(\left\lfloor\sqrt{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4}\right\rfloor\right) \subseteq \mathsf{2NSIZE(d)}$. By Lemma 9 we have that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \in \mathsf{2N\Theta SIZE(d)}$ and $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \in \mathsf{2NSIZE(d)}$. After some certain calculations, we can show that

$$N\left(\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor}\right) > 2^{\lfloor\sqrt{\lfloor\frac{d-4}{13}\rfloor - 4}\rfloor(\lfloor\sqrt{\lfloor\frac{d-4}{13}\rfloor - 4}\rfloor + 1)}.$$

Then, by Theorem 7, we infer that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \notin \mathsf{2N\Theta SIZE}\left(\left\lfloor\sqrt{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4}\right\rfloor\right)$ and $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \notin \mathsf{2NSIZE}\left(\left\lfloor\sqrt{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4}\right\rfloor\right)$.

The Proof of Theorem 4: Let us recall that $d' \leq \left\lfloor\sqrt{\lfloor\frac{d-4}{13}\rfloor - 4}\right\rfloor$. We have already shown that $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \notin \mathsf{2N\Theta SIZE}\left(\left\lfloor\sqrt{\lfloor\frac{\mathsf{d-4}}{13}\rfloor - 4}\right\rfloor\right)$ and $\mathsf{2\text{-}SAF}_{\lfloor\frac{\mathsf{d-4}}{13}\rfloor} \in \mathsf{2NSIZE(d)}$.

Moreover, $\mathsf{EQ} \in \mathsf{2N\Theta SIZE(4)}$, but by Corollary 8 we have $\mathsf{EQ} \notin \mathsf{2N\Theta SIZE(d')}$.

Acknowledgements

We thank Farid Ablayev and Aida Gainutdinova for their helpful comments on the models and the anonymous reviewers for their constructive comments.

References

[1] F. ABLAYEV, K. KHADIEV, Extension of the hierarchy for k-OBDDs of small width. *Russian Mathematics* 53 (2013) 3, 46–50.

[2] F. M. ABLAYEV, M. KARPINSKI, On the Power of Randomized Branching Programs. In: *ICALP*. LNCS 1099, Springer, 1996, 348–356.

[3] K. BALODIS, One Alternation Can Be More Powerful Than Randomization in Small and Fast Two-Way Finite Automata. In: *FCT*. LNCS 8070, 2013, 40–47.

[4] D. A. M. BARRINGTON, H. STRAUBING, D. THÉRIEN, Non-Uniform Automata Over Groups. *Information and Computation* 89 (1990) 2, 109–132.

[5] M. CHROBAK, Finite automata and unary languages. *Theoretical Computer Science* 47 (1986) 0, 149 – 158.
http://www.sciencedirect.com/science/article/pii/0304397586901428

[6] V. GEFFERT, B. GUILLON, G. PIGHIZZINI, Two-Way Automata Making Choices Only at the Endmarkers. In: *Language and Automata Theory and Applications*. LNCS 7183, Springer Berlin Heidelberg, 2012, 264–276.

[7] M. HOLZER, Multi-head finite automata: data-independent versus data-dependent computations. *Theoretical Computer Science* 286 (2002) 1, 97–116.

[8] C. A. KAPOUTSIS, Removing Bidirectionality from Nondeterministic Finite Automata. In: *MFCS*. LNCS 3618, Springer, 2005, 544–555.

[9] C. A. KAPOUTSIS, Size Complexity of Two-Way Finite Automata. In: *Developments in Language Theory*. LNCS 5583, Springer, 2009, 47–66.

[10] C. A. KAPOUTSIS, Nondeterminism is essential in small two-way finite automata with few reversals. *Information and Computation* 222 (2013) 0, 208 – 227. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011).
http://www.sciencedirect.com/science/article/pii/S0890540112001617

[11] C. A. KAPOUTSIS, R. KRÁLOVIC, T. MÖMKE, Size complexity of rotating and sweeping automata. *Journal of Computer and System Sciences* 78 (2012) 2, 537–558.

[12] H. LEUNG, Tight Lower Bounds on the Size of Sweeping Automata. *Journal of Computer and System Sciences* 63 (2001) 3, 384 – 393.
http://www.sciencedirect.com/science/article/pii/S0022000001917830

[13] A. SALOMAAA, M. SOITTOLA, *Automata-Theoretic Aspects of Formal Power Series*. Texts and monographs in computer science, Springer-Verlag (New York), 1978.

FINITE AUTOMATA WITH CONTINUOUS INPUT

András Kornai[†]

Department of Algebra, Budapest University of Technology and Economics
andras@kornai.com

Abstract
Euclidean Automata (EA) are finite state computational devices that take continuous parameter vectors as input. We investigate decompositions of EA into simpler Euclidean and classical finite state automata, relate them to better known computational devices such as artificial neural nets and electronic devices, and show that they are plausible Big Mechanism candidates.

1 Introduction

Euclidean automata (EA) were introduced and motivated from the perspective of Artificial Intelligence, in [8]. For convenience, we repeat some of the definitions and examples here, but the focus of the current paper is with the decomposition of EA and relating the concept to better known computational frameworks, subjects that could not be discussed in the earlier work. EA are modified Finite State Automata (FSA) that draw their input not from a finite alphabet as usual, but rather from vectors from a parameter space P, typically $\mathbb{R}^n$. For quantum applications, $\mathbb{C}^n$ would also be of interest, but we concentrate on the real case.

Definition 1.1 A *Euclidean automaton* (EA) over a parameter space P is defined as a 4-tuple $(\mathcal{P}, I, F, T)$ where $\mathcal{P} \subset 2^P$ is a finite set of states given as subsets of P; $I \subset \mathcal{P}$ is the set of initial states; $F \subset \mathcal{P}$ is the set of accepting states; and $T : P \times \mathcal{P} \to \mathcal{P}$ is the transition function that assigns for each parameter setting $\vec{v} \in P$ and each state $s \in \mathcal{P}$ a next state $t = T(\vec{v}, s)$ that satisfies $\vec{v} \in t$.

Thus, while the input is drawn (in discrete steps) from a continuous domain, EA are still finite in that *states* are simply subsets P_i of P indexed from a finite index set S. If $P_i \cap P_j = \emptyset$ for all $i, j \in S$ we call the EA *deterministic*, if $\bigcup_{i \in S} P_i = P$ we call it *complete*. If all P_i are open subsets of the the Euclidean space, we call the EA *open*.

In problems of linguistic pattern classification the number of classes (e.g. the set of valid characters in optical character recognition (OCR) or the set of phonemes in automated speech recognition (ASR)) is finite, and small continuous deformations of the input leave the output

[†]Work supported by OTKA grant #82333 and by the European Union and the European Social Fund through project FuturICT.hu (grant # TAMOP-4.2.2.C-11/1/KONV-2012-0013).

unchanged, suggesting that the classifier is an open EA. The main problem with this is that we cannot partition $\mathbb{R}^n$ or $\mathbb{C}^n$ into finitely many disjoint open sets. Approximate solutions thus must give up non-overlapping, e.g. by permitting probabilistic or fuzzy outcomes, or exhaustiveness, e.g. by leaving 'gray areas' near decision boundaries where the system produces no output. As discussed in [8], EA take the first route, sacrificing determinism (non-overlapping). This enables modeling of the nondeterminism actually seen in cases of perceptual hysteresis [12]. In fact, we observe that the appearance of hysteresis in such situations is not an accident:

Observation 1.1 There is no deterministic classification of a connected parameter space $P \subset \mathbb{R}^n$ by some open EA.

Proof By definition, connected subspaces of Euclidean space cannot be the discrete union of open sets.

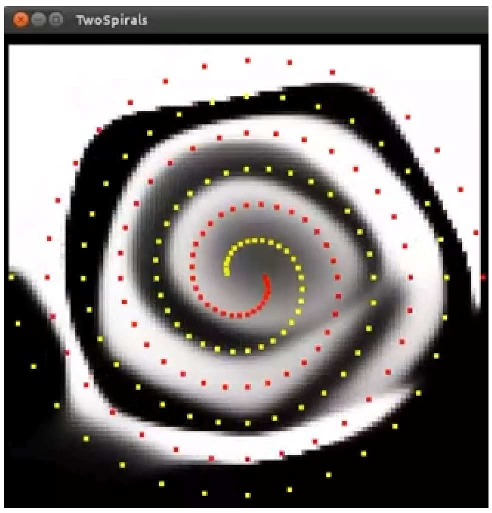


Figure 1: Decision boundary in 2-20-10-1 layer perceptron

As an example, consider the multilayer perceptron depicted in Fig. 1, trained in [5]. Despite the complexity of the decision boundary, the EA with equivalent behavior has only two states, corresponding to the black and the white subsets of the image. The input vectors are two-dimensional, and there is no output to speak of (we could designate one of the two states final). To account for output, we consider the following definitions (repeated from [8] for convenience):

Definition 1.2 A *Euclidean transducer* (ET) over a parameter space P is defined as a 5-tuple $(\mathcal{P}, I, F, T, E)$ where $\mathcal{P}, I, F$, and T are as in Def. 1.1 and E is an emission function that assigns a string (possibly empty) over a finite alphabet Σ to each transition defined by T.

Definition 1.3 A *Euclidean Eilenberg Machine* (EEM) over a parameter space P is defined as a 5-tuple $(\mathcal{P}, I, F, T, R)$ where $\mathcal{P}, I, F$, and T are as in Def. 1.1 and R is a mapping $P \times \mathcal{P} \to P$ which assigns to each transition a (not necessarily linear, or even deterministic) transformation of the parameter space.

2 Decomposition

The definition of EA leaves open the possibility that the parameter space P is embedded in $\mathbb{R}^n$ in a partially discrete manner, e.g. as indexed subsets of lower-dimensional spaces. Consider, for example, a three-stop elevator running from the basement to the top (first) floor of some building. This will have both continuous parameters, such as the reading from the position sensor, and discrete parameters, such as the reading from the engine sensor, with only three possible values "going up", "stopped", and "going down". Some of the parameters, such as the reading from the weight sensor, are seemingly continuous but effectively quantized to two discrete values, "above safety limit" or "below safety limit". In such cases, we may want to select *canonical representatives* $\vec{p}_i$ from P_i. (Other input values, pertaining to the state of call buttons at each floor and inside the cab, to accelerometer readings, or to sensors for AC power quality could be added, but we don't aim at realistic details here.)

As long as the parameters can be isolated from one another we can view P as being the direct product of smaller parameter spaces P_i, some Euclidean, some discrete. Isolating the parameters is easy enough for elevators with different sensors, but not at all trivial in pattern recognition tasks where the individual coordinates, such as spectral peaks in ASR, can show all kinds of interdependence. There is notable uncertainty how we wish to embed the discrete spaces in $\mathbb{R}$, for example two-valued parameters are often encoded as 0 or 1, but often as -1 or $+1$, and there is no easy way to select a canonical embedding. In many applications, n-valued parameters are encoded as $0, \ldots, n-1$, in others as $1, \ldots, n$, in yet others as $0, 1/(n-1), 2/(n-1), \ldots, 1$. Let us first consider a family M of $P \to P$ mappings with the goal of replacing one conventional encoding by another. As the examples show, such mappings are typically taken from continuous/differentiable families, but are not necessarily linear.

Definition 2.1 An EA Ψ is the *homeomorphic image* of Φ under a mapping $m \in M$ iff for any sequence of inputs $\vec{v}_1, \ldots, \vec{v}_n$ we have $\Psi(m(\vec{v}_1), \ldots m(\vec{v}_n)) = m(\Phi(\vec{v}_1, \ldots, \vec{v}_n))$.

Here we assume that both Φ and Ψ are started from the same unique initial state (if we permit several initial states the definition needs to be complicated accordingly) and that equality means equality of result state. This is meaningful, since m naturally maps not just inputs on inputs, but also EA states (subsets of parameter vectors) on one another. We will say Φ and Ψ are *isomorphic* if they are homomorphic images of each other under some m and m^{-1}.

Definition 2.2 The *skeleton* of an EA Φ is a standard (Mealy) FSA whose alphabet corresponds to canonical representatives from each Boolean atom of $\mathcal{P}$.

In the deterministic case, this is also a Moore automaton, as there is a one-to-one correspondence between input letters and automaton states. As is clear from Definition 1.1, the sequential behavior of EA is relatively simple in this case, since the result state depends only on the input, and not on the previous state. In the nondeterministic case (which is the more interesting case as per Observation 1.1) we may not be able to select distinct canonical representatives for each state P_i, or even for the set of Boolean atoms formed by the P_i. Here skeleta must be replaced by what we will call *subjective* EA, unique only up to canonical isomorphism, but sacrificing the one-to-one correspondence between state set and input set.

Besides the well understood serial and parallel modes of composition (see 3.2 for examples), EA admit a further possibility. This can already be illustrated in the simplest case of a mixed parameter space, where P_c, the continuous part of the parameter space, is just $\mathbb{R}$, and P_d, the discrete part, is just a binary choice $\top, \bot$. We may think of an EA Φ over $P = P_c \times P_d$ as being composed of two simpler EA $\Phi_\top$ and $\Phi_\bot$ by means of a real parameter p that *influences* whether the $\Phi_\top$ or the $\Phi_\bot$ behavior dominates. Importantly, the parameter that does the influencing may just be the input parameter, providing a crude form of memory, as in Example 3 of [8].

3 Relating EA to Other Computational Mechanisms

3.1 Artificial neural nets In classification tasks our interest is with the inverse images C_i of the possible outputs i. As our example in Fig. 1 shows, EA offer a method for directly encoding the information concerning the shape of the C_i where it belongs, in $\mathbb{R}^n$, where n is the dimension of the input parameter vector, rather than in $\mathbb{R}^{m \times m}$, the matrix of connection strengths. As is well known, in pattern recognition a great deal depends on the preprocessing of the signal, and using EA can make this dependence explicit. For example, consider the "two circles" data set presented in [10] reproduced here as Fig. 2: while it is evident that no linear separator (simple NN) exists, transforming the data to a system of polar coordinates around the center of gravity of the datapoints would make the task trivial. In ASR, we routinely apply a far more elaborate sequence of data transformation steps (power cepstra [1], mel warping [3], and delta cepstra [6]) to make the data manageable. Altogether, the use of EA is expected to bring new insights, especially for the increasingly popular but not yet well understood "deep learning" neural net architectures such as LSTM [7].

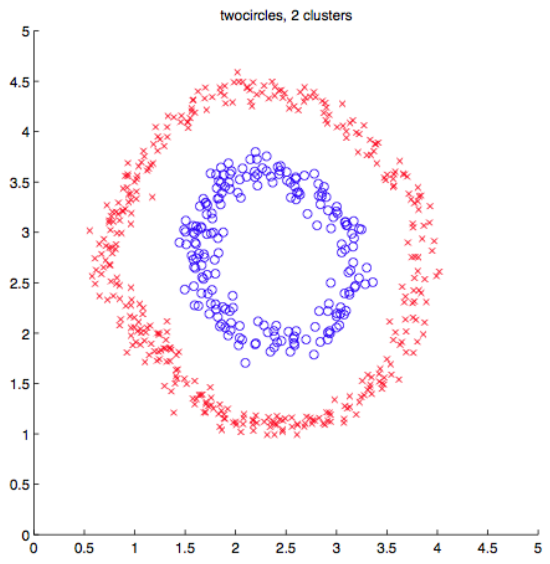


Figure 2: "Two circles" data from Ng (2001)

3.2 Electronics For many applications it makes sense to define the initial state as a parameter region P_0 that has no overlap with the other states of the automaton (even for EA not otherwise deterministic), since this will guarantee that we can *reset* the EA to the initial state by making sure that there are outbound transitions from every P_i. If we have another region we can reset to, we obtain an EA corresponding to the classical *flip-flop* (or *latch*) circuit. As discussed in [8], we can also obtain classical circuits with hysteresis, such as a Schmitt trigger [13]. By creating EA corresponding to elementary building blocks such as transistors, all forms of logic circuitry operating on continuous variables such as voltages could be recast as networks of EEMs. Modeling of transient behavior is not facilitated by the framework.

As is standard in logic design, digital circuitry can be conceptualized as series-parallel composition of standard FSTs, either with output string length limited to 1 (otherwise issues of timing and synchrony become paramount) or with clock signals added in. For semi-analog circuitry, where the outputs of each buiding block can be characterized as constant values (or values with very little variation) the same series-parallel conceptualization is available with Eucldean transducers (ETs), as long as we take the outputs of the upstream ETs to be the canonical representatives of the inputs of the downstream ETs. This means that in principle all physical models of digital computation, realized by discrete electronics as they are in current computers, are within scope of the EA/ET/EEM model as defined in D1.1-3. These are theoretical models, unlikely to gain much traction in circuit design, where transitional behavior and synchrony are highly relevant, but the connection makes clear that EA don't suffer from the kind of realizability problems that plague many theoretical computing devices from quantum gates to memristors.

3.3 WFSA, Bayes/Markov nets Here the relationship is much more tenuous, in that both weighted FSA (WFSA) and Bayesian nets take discrete inputs (in the Bayesian case just 0-1, for WFSA strings from an arbitrary finite alphabet) and produce continuous values on output, not the other way round as in EA. In the case of WFSA and WFST, there is no way to pass the weights to later automata, so in investigating how automata can influence each other we are restricted to communication by means of strings from a discrete alphabet. While the difficulties are not insurmounatble (after all, we can always encode good approximations of continuous values in discrete strings) the system is highly unnatural, requiring a great deal of decoding and re-encoding to express any simple arithmetical relation such as the activation of a node being determined by the sum of the inbound activation levels.

Bayesian/Markov nets offer a highly coherent way of thinking about probabilistic influences, but here the technical difficulties are even worse. There is nothing to say, at least in principle, that the random variables at the nodes couldn't be continuous, but to compute the marginal density of a child node we have to integrate out all the parents. Really the only parametric family that makes sense in this setting is (multivariate) normal [9], which excludes many systems of great interest to which we now turn.

3.4 Big Mechanism Following the success of Big Data, DARPA has started a search towards a *Big Mechanism* that is capable of modeling cancer signaling pathways, the brain, climate, the economy, and other large interconnected systems, focusing initial attention on cancer signaling pathways. Acquiring the data from medical journals and databases, requiring both considerable standardization of terminology and sophisticated natural language processing (NLP) capabilities, are seen as part of the task of developing Big Mechanisms, which are "large, explanatory models of complicated systems in which interactions have important causal effects" [2]. In the graphical language standard for systems biology [11], which notes the functional analogy with electronic circuit diagrams and algorithm block diagrams, we see both discrete, FSA-like elements such as a channel being open or closed, a muscle being tense or relaxed, and continuous, additive elements such as the level of various chemicals. Eilenberg Machines [4] were developed, for the discrete case, with block diagrams in mind, and EEM provide the kind of continuous generalization that makes e.g. stoichiometry possible to be modeled.

References

[1] B. P. BOGERT, M. J. HEALY, J. W. TUKEY, The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. In: M. ROSENBLATT (ed.), *Proceedings of the Symposium on Time Series Analysis*. Wiley, 1963, 209–243.

[2] P. COHEN, Big Mechamism Program. 2014. DARPA BAA, accessed 5/2014.
`http://www.darpa.mil/Our_Work/I2O/Programs/Big_Mechanism.aspx`

[3] S. B. DAVIS, P. MERMELSTEIN, Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28 (1980) 4, 357–366.

[4] S. EILENBERG, *Automata, Languages, and Machines*. A, Academic Press, 1974.

[5] A. FABISCH, *Two Spirals Problem Solved in Compressed Weight Space*, 2011.
`https://www.youtube.com/watch?v=MkLJ-9MubKQ`

[6] S. FURUI, Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 34 (1986) 1, 52–59.

[7] S. HOCHREITER, J. SCHMIDHUBER, Long Short-Term Memory. *Neural Computation* 9 (1997) 8, 1735–1780.

[8] A. KORNAI, Euclidean Automata. In: M. WASER (ed.), *Implementing Selves with Safe Motivational Systems and Self-Improvement*. Proc. AAAI Spring Symposium, AAAI Press, 2014, 25–30.

[9] S. LAUR, Bayesian Networks with Continious Distributions. 2009. Unpublished class notes, accessed 5/2014.
`http://bit.ly/1jJs2M6`

[10] A. Y. NG, M. I. JORDAN, Y. WEISS, On Spectral Clustering: Analysis and an algorithm. In: *Advances in neural information processing systems*. MIT Press, 2001, 849–856.

[11] N. L. NOVÈRE, The Systems Biology Graphical Notation. *Nature Biotechnology* 27 (2009), 735–741.

[12] S. POLTORATSKI, F. TONG, Hysteresis in the Perception of Objects and Scenes. *Journal of Vision* 13 (2013) 9, 672.

[13] O. H. SCHMITT, A thermionic trigger. *Journal of Scientific Instruments* 15 (1938) 1, 24.

DISSECTION ROTATION SCHEMES FOR CONTEXT-FREE GRAMMARS: ENHANCING PARSING AND AMBIGUITY RESOLUTION

Eli Shamir$^{(A)}$

$^{(A)}$Hebrew University of Jerusalem, Jerusalem, Israel
shamir@cs.huji.ac.il

Abstract
Main result: Two dissections rotations "top down" schemes are constructed transforming (in stages) a context-free grammar (CFG) G, provided:

1. *If G is non-expansive (NE), then G is transformed to a bounded-size vector of* **linear grammars** $(G(1), \ldots, G(k))$*. Membership/parsing (for a terminal string* w*) is reduced to that of* $G(i)$*, using rotation-extended CYK tables for the linear grammar* $G(i)$*. The overall complexity is* $O(n^2)$*, where* n *is the length of* w*.*

2. *If ambiguity-deg(G)* $= l < \infty$*, then G is transformed to a bounded-size union of deg 1-grammars. This provides a positive answer to a question Sam Eilenberg posed 1970.*
"Bounded size" means polynomial in $|G|$*, the size of the grammar G, or* $|G|$ *and* l*.*

The key operation driving those schemes is Top Trunk Rotation (TTR) of a product of two grammars, which also rotates derivation trees in a 1-1 onto manner, and induces a cyclic rotation on the derived strings.

1 Introduction

In formal language theory, some subtle transformations on grammars (e.g. to normal forms) may destroy essential structural properties. The scheme presented in Section 3, for non-expansive context tree grammars, **preserves** derivation trees, and enables their quicker discovery (i.e. parsing). Solutions of the associated algebraic system are also preserved (see Remark 1, Section 5). Section 4 describes a scheme for decomposing grammars with bounded degree of ambiguity.

2 Context-Free Grammars (CFG) - Basics

A context-free grammar $G = (V, T, P, S = \text{root})$ is a well-known model to derive/generate a set of **terminal** strings in T^*. G defines a derivation relation between strings over $V \cup T$:

One step $x \to y : y$ is obtained from x by rewriting a single occurrence of some A by $B_1 \dots B_k$ if $A \to B_1 \dots B_k$ is production rule in P.

Several steps $x \overset{\star}{\to} y$ if $x \to x_1 \cdots \to y$.

Let $L_A(G) = \{w \in T^* \mid A \overset{\star}{\to} w\}$. Then $L(G) = L_S(G)$, the language generated by G.

A derivation is best described by a labeled tree in which the k sons of a node labeled A are labeled $B_1, \dots, B_k$ if $A \to B_1 \dots B_k$ is production rule in P.

The ambiguity degree of $(A \overset{\star}{\to} w)$ is the number of distinct trees for $A \overset{\star}{\to} w$; we write $deg(A \overset{\star}{\to} w)$.

$deg(G_A) = max_{w \in T^*} deg(A \overset{\star}{\to} w)$.

$A \overset{\star}{\to} -B-$ defines a partial order on $V \cup T$, denoted $A > B$. it induces a complete order on any branch of a derivation tree.

B in G is **pumping** if $B > B' > B$. Then B' is also pumping; both belong to the pumping equivalence class $[B]$.

Normal form. For most purposes one can assume G is reduced, i.e. superfluous symbols and productions are removed, and the remaining productions are binary or unary. In addition, we call a production $B \to \alpha$ EXIT rule if B is pumping and α does not contain symbols of $[B]$. Then replace each EXIT rule $B \to \alpha$ by $B \to B^\wedge$ and $B^\wedge \to \alpha$, $B^\wedge$ is a new symbol.

Thus $B^\wedge$ takes over the non-pumping role of the symbols in $[B]$. This entails a clear-cut classification of each symbol in V as

(i) B *pumping*,

(ii) C *pre-terminal* if NOT $[C > B$, B pumping$]$,

(iii) D *spread* if D is not pumping but $D > B$, B pumping.

Lemma 2.1 (Spread).

1. *A pre-terminal C derives a bounded number of bounded terminal strings.*
2. *In each derivation tree a spread node D derives a bounded sub-tree the leaves of which are terminals or pump nodes.*
3. *In G, each spread symbol D derives a bounded number of sub-trees as mentioned in (2).*

This lemma is proved by simple enumerations, since the branches in those sub-trees are shorter than $|V|$.

Definition 2.2. *G is called* **non-expansive** *(NE) if no production rule has the form $B \to -B' - B''-$ where the B's are from the same pumping class (NE is the class* **quasi-rational** *in [1]). Equivalently, no derivation $B^\star \to -B - B-$ is possible (sideway pumping is forbidden!).*

Definition 2.3. *E-depth of a node D is the maximal size of a strictly decreasing chain with respect to the $>$ order along a branch from D to terminals.*

Lemma 2.4 (Top Trunk). *If G is non-expansive and B is a pump root in a sub-tree then symbols of $[B]$ appear only on the* **top trunk** *of the branch going down from B to a uniquely determined EXIT $B^\wedge$. For this $B^\wedge$ and for all other side-symbols of the sub-tree the E-depth decreases.*

3 Bounded Operation Tree for a NE Grammar G

The bounded operation tree [BOT] has nodes of degree 1 (with a single child) labeled CYC or TTR, and nodes of degree ≥ 1 labeled SPR. At each node, the **current grammar** is a product Π of factors, which are grammars or terminals. The operation at the node acts on the right end (as described below) or the left hand in a similar fashion (see TERMINATION below).

1. If $\Pi = N(1) \dots N(k)v$, where v is a bounded terminal string and $N(k)$ has a pumping root, then the operation CYC rotates Π to $vN(1) \dots N(k)$ at the child node.
2. If v is empty and $N(k)$ has a pump root, then the operation is TTR as described in Definition 3.1 below, acting on $M = N(1) \dots N(k-1), N = N(k)$. The grammar at the child node becomes $M^* N^\wedge$.
3. If (after TTR at the father node) the root of $N^\wedge$ is a spread symbol of G, the operation is SPR: by the SPREAD Lemma 2.1, this root derives $\cup_{j \in J} \Pi(j)$. This spread node then will have $|J|$ children and the current grammar at the child number j is the product $M^* \Pi(j)$.

The grammar at the root of BOT is $\#G$, $\#$ is a special marker which does not appear elsewhere in G, assuring that no proper cyclic rotation of terminal strings $w = \#v$ can have a derivation tree of G.

Definition 3.1. *The operation TTR transforms the product MN into $M^* N^\wedge$. It takes the top trunk (see Lemma 2.4) of N which is labeled by symbols of the pumping class [B], rotates it by 180° and mounts it as top trunk on M, which becomes now M^*, while $N^\wedge$ is the subgrammar of N below the removed top trunk.*

TTR on the grammars, assuming that the productions are binary, is:

$$B \to B'C \text{ in } N \text{ goes to } B' \to CB \text{ in } M^*,$$

$$B \to DB' \text{ in } N \text{ goes to } B' \to BD \text{ in } M^*.$$

All other productions not including symbols of the main trunk, in particular those of $N^\wedge$ and M, do not change.

TTR induces a cyclic relation on the derived strings (by MN and $M^*N^\wedge$):

$$w = x_1x_2\dots n^\wedge \dots y_2y_1m \quad \textit{is rotated to} \quad \dots y_2y_1mx_1x_2\dots n^\wedge = \rho w.$$

In particular, M and $N^\wedge$ switch places.

Bounded Termination for BOT

The operation at the child of a CYC node must be TTR, after which it is TTR again or SPR and after SPR the children are labeled TTR or CYC followed by TTR. Thus, TTR repeats after three steps at most. Now, TTR decreases the E-depth (see Definition 2.3) for $N^\wedge$ and for the left and right factors hanging on both sides of the main trunk of $M^\star$ (which was mounted from N after rotation by 180 degrees). These side factors will be operated upon later, from the left or right side respectively, and finally the bottom $N^\wedge$ will be operated upon. Clearly this entail bounded termination. Moreover, the "top towers" of the $M^\star$ factors, which get another floor in TTR, grow taller and thinner until they become linear grammars at the leaves of the BOT tree.

Correctness for BOT (for enhancing the membership/parsing of $L(G)$)

Following the branches from the root of BOT with grammar $\#G$ to the leaves of BOT with linear grammars, we claim: any derivation tree Γ for $w = \#v$ is transformed along a branch of BOT to a tree Γ' of grammar G' at some leaf and V.V., each such tree Γ' at a leaf comes from a tree Γ for w at the root of BOT. Indeed following the operation on the branch step by step, SPR distributes a union grammar, hence all possible trees, among its children. The operation TTR rotates the grammar, the trees and the derived strings, in a one to one invertible fashion. In the CYC operation, the grammar and trees are just copied to the child node since we use:

Definition 3.2. *A ρ-derivation tree for w is any usual derivation for some rotation ρw. This entails no loss of generality since the special marker # assures that at the root of BOT no non-trivial rotation has a derivation tree.*

For the membership testing: we construct **rotation-extended** CYK tables at all (bounded in number) leaves of BOT in time $O(n^2)$ since the grammars at the leaves are linear. There is a tree for $\#w$ at the root if and only if there is one at some leaf node, and we can extract an explicit parse tree. In fact, all trees are implicit at the CYK tables but explicitly listing them will require cubic time. Even a single linear grammar can have $O(n)$ distinct trees for the same w (as in the example given below).

Example 3.3. *(taken from [4]) $MN = uIu^Rv^RJv, \quad u, v \in \{0.1\}^* = I = Ju^R =$ reversal of u. It has unbounded "direct (product) ambiguity" which increases the time in CYK algorithm. But after one TTR step MN is rotated to*

$$M^* N^\wedge = vuIu^R v^R J,$$

which has a linear grammar. *All product ambiguity trees are rotated to union trees for $M^* N^\wedge$.*

4 Decomposing Bounded Ambiguity

We briefly sketch the scheme for result (2) in the abstract. Starting with $\#G$, and using the SPREAD Lemma 2.1, the claim is reduced to:

Lemma 4.1. *Let $\Pi = MN(1)\ldots N(k)$, $deg(M) = 1$, $deg(\Pi) = l < \infty, N(i)$ are terminals or with pump roots, then there exists a bounded J such that*

$L(M) = \cup_{j\in J} L(M(j))$ with $deg(M(j)) = 1$ for all $j \in J$. (4.1)

It suffices to prove it for a pair, starting with $MN(1)$, after which $M(j)N(2)$ are decomposed and so on. For a pair MN the operation TTR is used transforming it to $M^* N^\wedge$. Now $deg(M^*) \leq l$ and its ambiguity must be concentrated along the top trunk which it got from N. An easy direct argument shows it decomposes into a bounded union of $M(j)$ of degree 1. As for $N^\wedge$ its E-depth is smaller than that of N, for $M(j)N^\wedge$ we can use induction on the E-depth of the second factor or, more explicitly, continue the recursive descent on $N^\wedge$ until it is consumed.

5 Concluding Remarks

1. The BOT scheme for non-expansive G is analogous to producing characteristic vectors of a matrix transformation. Indeed, BOT preserves derivation trees of G and reveals them more efficiently. Moreover, it is easy to show similar preservation for solutions of the algebraic equation systems associated with a CF grammar. Hence, as for the linear case, non-expansive systems have a unique solution. Also, proving that languages are inherently expansive (e.g. Dyck languages) is simplified.
2. The concept of "**hardest context free grammar**" is due to Greibach (see [3]). The simplest one is based on Shamir's homomorphism theorem (see [1]), mapping each b in T into a finite set $\phi(b)$ of strings over the vocabulary of the Dyck language and claiming that w is in $L(G)$ if and only if $\phi(w)$ contains a string in the Dyck language (see the description in [1]).

   In fact, the categorical grammar model in the 1960 article (see [2]) provides another homomorphism which makes it a hardest CFG.

   However, those hardest CFG languages are inherently expansive. Indeed, an NE candidate grammar for Dyck will be negated by its BOT scheme, upon using local pump-shrinks, which **for linear grammars can operate near any point of the (sufficiently long) main branch of non-terminals.**

We conjecture that any hardest CFG must be expansive. Note that finding a non-expansive one would entail $O(n^2)$ complexity of membership test for any context-free grammar.

3. Ambiguity in natural languages can be resolved (or created) by cyclic rotation. Consider the Biblical verse in book of Job chapter 6 verse 14 (six Hebrew words). Translated to English: “a friend should extend # mercy to a sufferer $, even if he abandons God’s fear.”

   The ambiguity here is anaphoric, does the pronoun “he” refer to the sufferer or to the friend? The poetic beautiful answer is: to both. The rotated sentences, starting at the symbols # and $, resolve the ambiguity towards one way or the other.

   Another famous example: Tom saw # Eve $ with the telescope.

References

[1] J. AUTEBERT, J. BERSTEL, L. BOASSON, Pushdown Automata and Context Free Languages. In: *Handbook of Formal Languages*. 1. chapter 3, Springer, 1997, 111–174.

[2] Y. BAR-HILLEL, H. GAIFMAN, E. SHAMIR, On categorical and phrase structure grammars. *Bulletin of the Research Council of Israel* 9F (1960), 1–16.

[3] S. GREIBACH, The hardest context-free language. *SIAM J. on Computing* 3 (1973), 304–310.

[4] E. SHAMIR, Some inherently ambiguous context-free languages. *Information and Control* 18 (1971), 355–363.

A PROBABILISTIC GRAMMAR FOR PROCEDURAL CONTENT GENERATION

Francesco Sportelli Giuseppe Toto
Gennaro Vessio

Department of Informatics – University of Bari
`{f.sportelli1, g.toto}@studenti.uniba.it, gennaro.vessio@uniba.it`

Abstract
In game development, procedural content generation *(PCG) is an important area of research that aims at generating content algorithmically rather than manually. This paper shows the successful application of a* probabilistic *grammar in order to procedurally generate a game environment at run-time. The proposed approach helps saving memory, reduces development effort and increases longevity.*

1 Introduction

In the context of game development, *procedural content generation* (PCG) is an increasingly important area of research that aims at generating content algorithmically rather than manually. Traditionally, games are characterized by static content which are precomputed during development. For example, levels are fixed, non-player characters move along predefined paths, objects can be found often in the same places. This has some disadvantages: from the point of view of developers, the effort required during development is time-consuming; on the other hand, from the point of view of players, longevity may be affected. The goal of PCG techniques is to avoid these limitations.

PCG was pioneered in the early '80s by the game *Rogue*, in which dungeons to be explored are generated randomly. To the same period belongs *Elite*, a space adventure game in which hundreds of star systems are encapsulated in few tens of kilobytes. A game that makes extensive use of PCG is *Minecraft*: here, the initial state of the world is mostly random and new areas are generated randomly whenever the player moves towards its boundaries. Recently, PCG has been exploited in combination with *affective computing*: the aim is to adjust content at run-time according to the user needs and preferences recorded during the game (e.g. [3], [9]).

According to Togelius *et al.* [7], PCG offers several benefits from three different perspectives: memory consumption, since content can be generated only when needed; development effort, because the expense of manually creating content is alleviated; and longevity, since the same game looks different every time it is played.

However, the application of PCG presents some drawbacks. Since it can result in an unpredictable range of possible game scenarios, there is the need to impose some constraints that allow generating content in a correct and manageable way.

In this paper we explore the application of a *probabilistic right-linear grammar* [2] in order to procedurally generate the environment of *The Ball: Lost in Space*[1] game. In particular, the grammar allows for generating the path the player character has to go through and, simultaneously, the obstacles to avoid. We show the advantages of employing this technique in terms of reducing development effort and memory usage while increasing the longevity.

The usage of generative grammars in this field has been quite limited, mostly for generating buildings at development-time (e.g. [8], [5]), or for creating 2D levels at run-time (e.g. [1]). A greater contribution is provided by the so-called *L-systems*, for example see [6] and [4]; however, the main difference with respect to generative grammars is that rules are applied in parallel rather than once at a time, so resulting in fractal-like structures.

2 The Game

The Ball: Lost in Space is an *endless running* game with a third-person perspective. Like other endless running games, such as *Temple Run*, the player character has to move forward continuously through a path and the main goal is to cover as much distance as possible before inevitably succumbing. Controls are limited to making the character jump, and move left or right, in order to avoid obstacles. These features makes the game suitable for mobile platforms that typically require only a single screen tap to make an action. More precisely, the game consists in controlling the movements of a ball that goes through a wormhole. It ends when the ball crashes into one of the many asteroids that lie on the path or when the ball falls out in the cosmic void. Moreover, it is worth noting that the game includes the task of collecting coins; however, since coins are generated in a totally random way, this feature is not considered. The game was developed with Unity 3D[2].

The core algorithm of *The Ball: Lost in Space* (aiming at generating both the path and the obstacles to avoid) is based on a probabilistic right-linear grammar [2]. Briefly speaking, a *generative grammar* G is a quadruple (T, N, $\mathbf{S}$, R), where T and N are disjoint finite sets of symbols, called *terminal* and *non-terminal* symbols, respectively; $\mathbf{S} \in N$ is a non-terminal symbol called the *start symbol*; and R is a finite set of so-called *productions*. A production is a rewriting rule of the form $\alpha \to \beta$, where $\alpha \in N^+$ and $\beta \in (N \cup T)^*$. Starting from $\mathbf{S}$, productions are applied iteratively so that non-terminal symbols can be replaced by combinations of non-terminal and terminal symbols, resulting in the generation of a well-formed sentence of the language described by G.

According to the Chomsky hierarchy, a *right-linear* grammar has all productions in the form $\mathbf{A} \to b\mathbf{B}$ or $\mathbf{A} \to b$, where $\mathbf{A}$ and $\mathbf{B}$ are non-terminal symbols and b is a terminal symbol.

[1]`http://www.theball.it/`

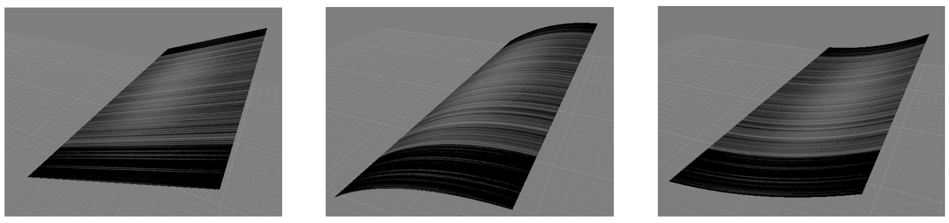
[2]`https://unity3d.com/`

A right-linear grammar becomes *probabilistic* by equipping each production with a certain probability of occurring. Obviously, the sum of the probabilities of all the productions with the same non-terminal symbol on the left-hand side must evaluate to 1. The main difference with respect to non-determinism is that in the latter it is deliberately decided to not specify how a certain choice is made.

Using grammars for generating a game environment requires that terminal symbols refer to elements to be drawn on the screen. In our case, terminal symbols refer to the 3D fragments of the wormhole and the asteroids. They are composed in order to generate at run-time a complex and each time different path along which the ball can move. Table 1 shows the correspondence between terminal symbols and the objects to be composed. Note that we have an overall number of thirteen different types of fragments plus one type of asteroid. In Fig. 1 three examples of path fragments are shown. The others are quite similar; they mostly differ from the point of view of the inclination.

Symbols	Meanings
a, b, c, d, e	Five types of plan
f, g, h, i	Four types of hump
l, m, n, o	Four types of bump
z	Obstacle

Table 1:: Terminal symbols of the grammar



(a) A plan (b) A hump (c) A bump

Figure 1:: Three examples of path fragments

The grammar allows us to manage the pure randomness of the path generation, which would be ungovernable, by applying some constraints that we have established *a priori*. The constraints can be simply inferred from the rules of the grammar: they allow us to generate paths not too dissimilar from city streets. In particular, they prevent the generation of "unwanted" shapes; for example, steep uphills immediately followed by steep downhills, or steep slopes that suddenly become straight stretches.

Given the 3D models and the constraints for composing them, we can now formalize the grammar as follows: $G = (T, N, \mathbf{S}, P)$, with $T = \{a, b, c, d, e, f, g, h, i, l, m, n, o, z\}$, $N = \{\mathbf{A},$

B, **C**, **D**, **E**, **F**, **G**, **H**, **I**, **L**, **M**, **N**, **O**, **Z**, **S**}, and productions and production probabilities in parentheses:

$\mathbf{S} \rightarrow a\mathbf{A}$
$\mathbf{A} \rightarrow a\mathbf{A}$ (0.65) | $n\mathbf{N}$ (0.05) | $l\mathbf{L}$ (0.05) | $g\mathbf{G}$ (0.05) | $i\mathbf{I}$ (0.05) | $z\mathbf{Z}$ (0.15)
$\mathbf{B} \rightarrow b\mathbf{B}$ (0.75) | $f\mathbf{F}$ (0.25)
$\mathbf{C} \rightarrow c\mathbf{C}$ (0.75) | $h\mathbf{H}$ (0.25)
$\mathbf{D} \rightarrow d\mathbf{D}$ (0.75) | $m\mathbf{M}$ (0.25)
$\mathbf{E} \rightarrow e\mathbf{E}$ (0.75) | $o\mathbf{O}$ (0.25)
$\mathbf{F} \rightarrow a\mathbf{A}$ (0.7) | $i\mathbf{I}$ (0.2) | $z\mathbf{Z}$ (0.1)
$\mathbf{G} \rightarrow d\mathbf{D}$ (0.7) | $m\mathbf{M}$ (0.3)
$\mathbf{H} \rightarrow a\mathbf{A}$ (0.65) | $g\mathbf{G}$ (0.05) | $i\mathbf{I}$ (0.05) | $l\mathbf{L}$ (0.05) | $n\mathbf{N}$ (0.05) | $z\mathbf{Z}$ (0.15)
$\mathbf{I} \rightarrow e\mathbf{E}$ (0.7) | $o\mathbf{O}$ (0.3)
$\mathbf{L} \rightarrow b\mathbf{B}$ (0.7) | $f\mathbf{F}$ (0.3)
$\mathbf{M} \rightarrow a\mathbf{A}$ (0.7) | $g\mathbf{G}$ (0.07) | $i\mathbf{I}$ (0.08) | $z\mathbf{Z}$ (0.15)
$\mathbf{N} \rightarrow h\mathbf{H}$ (0.7) | $c\mathbf{C}$ (0.3)
$\mathbf{O} \rightarrow a\mathbf{A}$ (0.7) | $g\mathbf{G}$ (0.07) | $i\mathbf{I}$ (0.08) | $z\mathbf{Z}$ (0.15)
$\mathbf{Z} \rightarrow a\mathbf{A}$.

Note that the terminal symbol z formalizes the positioning of an obstacle on the last generated path fragment. For instance, the substring *aza* is translated on the screen in a short straight, where is placed an obstacle, followed by another short straight. Moreover, in order to generate a theoretically infinite path, it is worth noting that rules get stuck in an infinite loop. At any moment, only $n = 100$ path fragments are stored in memory, and whenever the ball reaches the $\left(\frac{n}{2}+1\right)$-th fragment, the previous $\frac{n}{2}$ fragments are removed from memory, and $\frac{n}{2}$ new fragments are generated.

For what concerns probabilities, it is evident that we preferred flatter paths. In fact, during tests, we realized that assuming equiprobable rules causes the generation of too fluctuating paths, such that the gameplay is negatively affected.

Finally, as regards the amount of obstacles, we opted for an average of ten obstacles each one hundred path fragments, such that the game difficulty can be considered medium.

3 Discussion

Using the classification proposed by Togelius *et al.* [7], our approach can be defined as: on-line, because content generation is performed at run-time; necessary, because content is required by the player to make progress; parametric, because generation is managed by some constraints; stochastic, since the variation in the outcome between different runs is totally random; constructive, because content is generated and outputted at once without being tested.

Therefore, the proposed approach provides advantages from three different points of views. First of all, from the point of view of the usage of memory, it is worth noting that the path is generated only when needed. Moreover, since only 100 path fragments are stored at any moment, the path does not need to be kept in memory in all its length from the starting of the execution. Secondly, development effort is reduced: few constraints allow to obtain an incredible number of different combinations in an automatic way. This means that human imagination is augmented and also the effort of composing the fragments together is alleviated. Finally, longevity is increased: since the output is totally random, the player cannot memorize either the path or the positioning of the obstacles. So, he cannot get bored easily.

Grammars are well suited when the content generation requires the composition of few and no complex elements in order to obtain more articulated structures. However, in cases where the elements to be composed are more complex we recognize the difficulty to manage constraints and, so, production rules and their probabilities. Nevertheless, we think probabilistic grammars represent an excellent tool for developing mobile games; in fact, in such a context of use, players typically prefer simple and every time different games.

4 Conclusions

In this paper we have presented *The Ball: Lost in Space*: an endless running game whose core algorithm is based on a probabilistic right-linear grammar. The grammar allows the algorithmic generation of the environment of the game on the basis of few constraints. The proposed approach helps saving memory, reduces development effort and increases longevity. These features makes it suitable for mobile game development.

An interesting future development of our work could be to explore our grammar in combination with affective computing techniques. In this way, it could be possible to monitor players' emotional states and adjust the path generation at run-time. For example, if the player is bored, the algorithm could make the path more tortuous; vice versa, if the player is nervous, the algorithm could make the path more flat.

Acknowledgements

We wish to thank several people for their contribution to this research: Prof. Fabio Abbattista, whose lectures allowed the development of the game; Giuseppe Scavo, for his help in modeling the path fragments; and Alvin Pastore and Alessandro Bianchi – PhD, for their valuable and constructive suggestions.

References

[1] J. DORMANS, Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. In: *Workshop on Procedural Content Generation in Games*. ACM, 2010, 1–8.

[2] S. GEMAN, M. JOHNSON, Probabilistic Grammars and their Applications. In: *International Encyclopedia of the Social and Behavioral Sciences*. 2002, 12075–12082.

[3] E. HASTINGS, R. GUHA, K. STANLEY, Evolving Content in the Galactic Arms Race Video Game. In: *5th International Conference on Computational Intelligence and Games*. IEEE Press, 2009, 241–248.

[4] J. HIDALGO, E. CAMAHORT, F. ABAD, M. VICENT, Procedural Graphics Model and Behavior Generation. In: *8th International Conference on Computational Science*. Springer-Verlag, 2008, 106–115.

[5] P. MÜLLER, P. WONKA, S. HAEGLER, A. ULMER, L. V. GOOL, Procedural modeling of buildings. *ACM Transactions on Graphics* 26 (2006) 3, 614–623.

[6] Y. PARISH, P. MÜLLER, Procedural Modeling of Cities. In: *28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 2001, 301–308.

[7] J. TOGELIUS, G. YANNAKAKIS, K. STANLEY, C. BROWNE, Search-based Procedural Content Generation. In: *European Conference on Applications of Evolutionary Computation*. Springer-Verlag, 2010, 141–150.

[8] P. WONKA, M. WIMMER, F. SILLION, W. RIBARSKY, Instant Architecture. *ACM Transactions on Graphics* 22 (2003) 3, 667–677.

[9] G. YANNAKAKIS, J. TOGELIUS, Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing* 2 (2011) 3, 147–161.

WEIGHTED RESTARTING AUTOMATA

Qichao Wang

Fachbereich Elektrotechnik/Informatik
Universität Kassel, D-34109 Kassel
wang@theory.informatik.uni-kassel.de

Abstract
Restarting automata have been introduced to model the linguistic technique of analysis by reduction, which is used for checking the correctness or incorrectness of a sentence of a natural language. In order to study quantitative aspects of restarting automata, we introduce the concept of a weighted restarting automaton. In this work, we describe the weighted restarting automaton in detail, present some examples, state a few preliminary results, and give some hints on future research.

1 Introduction

Analysis by reduction [15] is a linguistic technique used to verify the correctness of sentences of natural languages by a stepwise simplification. During the process of simplifying a given sentence, each step preserves the correctness or incorrectness of the sentence. After a finite number of steps, either we obtain a correct simple sentence, or an error is found. Furthermore, by such a simplification we can also analyze the structure of the sentence and gain some deeper information on dependencies and independencies between certain parts of the sentence.

The *restarting automaton* [6] was invented by Jancăr et al. in 1995 as a useful tool for modeling *analysis by reduction*. As defined in [9] a restarting automaton consists of a finite control, a read/write window and a flexible tape. Initially, the tape contains a string as the input as well as border markers. When reading the input, the window moves along the tape performing various operations. During a Rewrite operation, some symbols from the content of the read/write window are deleted or replaced by a shorter string. After performing a Restart operation, the automaton places its window to the left end of the tape and re-enters the initial state, so that a new restarting configuration is reached. Such a phase that starts in a restarting configuration and ends with a restart operation is called a cycle. During any cycle, the automaton performs exactly one Rewrite operation, therefore each cycle starts on a shorter string than the previous one.

In recent years restarting automata gained much attention and many variants of them were developed. For example, by adding some restrictions on the Rewrite operation, we can obtain some various types of restarting automata, including RRWW-automata, RRW-automata and RR-automata (see [7] and [8]). RRWW-automata can use auxiliary symbols in the Rewrite operation, but RRW-automata cannot. RR-automata can only remove some symbols from the content of the read/write window in the Rewrite operation. In other words, RRW-automata and RR-automata have more restrictions on the Rewrite operation than RRWW-automata. In fact, we can also obtain many other types of restarting automata, e.g. by changing the size of read/write window [11] or by restricting the position of performing the Rewrite operation in each cycle [7]. Besides these, even some systems containing a finite collection of restarting automata that work together and communicate with each other in order to analyze a common sentence were introduced in [10] and [13]. A recent overview on restarting automata was given in [12].

Just as typical finite automata, also restarting automata accept or reject their input. Therefore, such a computing can be seen as a Boolean function. In 1960s, Schützenberger introduced weighted automata [14], which assign a numerical value as the weight to each transition. These weights can model the cost involved when executing a transition such as the needed resources or time, or the probability or reliability of its successful execution. For example, by using the weights, we can determine the number of ways that a word can be accepted by a finite automaton. After the introduction of weighted automata, they have been widely applied in many areas like natural-language processing, speech recognition, optimization of energy consumption and probabilistic systems. Additionally, their applications can also be found in digital image compression and model checking. Droste et al. systematically decribe weighted automata and their properties in [4], and obtained many new results on them, e.g. [1], [3] and [2]. Analogously, we introduce *weighted restarting automata* in order to study quantitative aspects of restarting automata, e.g. the minimal number of cycles in an accepting computation on an input. In the forthcoming sections, we will describe them in detail.

2 Definitions and Examples

A weighted restarting automaton is described by a couple $(\mathcal{M}, \omega)$, where $\mathcal{M}$ is a restarting automaton on some input alphabet Σ and ω is a weight function that assigns a weight from some semiring S to each transition of $\mathcal{M}$. A computation on an input consists of a finite number of transitions, and each transition is induced by an instruction of the transition function. The weight of a computation p is defined by

$$weight(p) = \prod_{1 \leq i \leq n-1} \omega(r(c_i, c_{i+1})),$$

where $r(c_i, c_{i+1})$ is the instruction used for the transition from the configuration c_i to the configuration c_{i+1}. The weight of a computation is essentially an element from the semiring S.

For an input, there may be several accepting computations. By summing the weights of all accepting computations, we obtain the following function:

$$f_\omega^{\mathcal{M}}(w) = \sum_{p \in P} weight(p),$$

where P is the set of accepting computations of $\mathcal{M}$ on the input w. Since the number of accepting computations is finite, this sum is also finite. Additionally, the value of this function is an element from the semiring S. Hence, a weighted restarting automaton induces a functions from some input alphabet into a semiring. We describe the function shown above as the behavior of a weighted restarting automaton. By using different semirings and weight functions, we can study various quantative aspects of a restarting automaton.

However, as mentioned above, there are various types of restarting automata. For different types of restarting automata, the induced functions may also be different. Thus, for a type t of restarting automata, we define the class of representable functions as follows:

> $F^t_{\Sigma,S} = \{f : \Sigma^* \to S \mid \exists$ a restarting automaton $\mathcal{M}$ of type t over Σ and a weight function ω with $f = f^{\mathcal{M}}_\omega\}$.

In the following we will show some examples. The first two examples are the starting point of weighted finite automata and shown in many works on them. The third example involves a special quantitative aspect of restarting automata.

Example 2.1. *For the semiring $S = (\mathbb{N}, +, \cdot, 0, 1)$, if $\omega(r) = 1$ for all $r \in \delta$, then the product of the weight of all used transitions in a computation is 1, which is exactly the weight of this computation. By adding the weight of all accepting computations, we can count the number of accepting computations on an input.*

Example 2.2. *For the semiring $S = (\mathbb{N}, \min, +, \infty, 0)$, if $\omega(r) = 1$ for all $r \in \delta$, then the weight of a computation is the sum of the weight of all used transitions in this computation, which is the number of steps in this computation. By using the min operation, we can determine the minimal number of steps in an accepting computation on an input.*

Example 2.3. *Similarly, for the semiring $S = (\mathbb{N}, \min, +, \infty, 0)$, if $\omega(r) = 1$ for $r \in \delta$ involving the* Restart *operation and $\omega(r) = 0$ otherwise, then we can determine the minimal number of cycles in an accepting computation on an input, because after performing a Restart operation the automaton enters a new cycle.*

3 Results

In this section, we will present some preliminary results on the properties of weighted restarting automata. As mentioned above, each weighted restarting automaton represents a function from some input alphabet into a semiring. For some linearly ordered semirings, e.g. the semiring of natural numbers with addition and multiplication, there is a maximal element. Given a linearly ordered semiring S, a restarting automaton $\mathcal{M}$ and a weight function ω, we define the function $\hat{f}^{\mathcal{M}}_\omega : \mathbb{N} \to S$ as follows:

$$\hat{f}_\omega^{\mathcal{M}}(n) = \max\{\, f_\omega^{\mathcal{M}}(w) \mid w \in \Sigma^n \,\}.$$

First, we found that the functions $\hat{f}_\omega^{\mathcal{M}}$ can represent any polynomial without auxiliary symbol.

Theorem 3.1. *Any polynomial can be represented by the functions $\hat{f}_\omega^{\mathcal{M}} : \mathbb{N} \to S$, where $\mathcal{M}$ is of type RRW.*

For a linearly odered semring, another interesting question is what is the upper bound of the functions $\hat{f}_\omega^{\mathcal{M}}$. We obtained the next theorem.

Theorem 3.2. *For the semiring $S = (\mathbb{N}, +, \cdot, 0, 1)$, the functions $\hat{f}_\omega^{\mathcal{M}} : \mathbb{N} \to S$ are bounded by $2^{O(n^2)}$.*

Additionally, we successfully showed that the upper bound given above is sharp. This means that this upper bound can be reached by some functions.

Theorem 3.3. *The upper bound $2^{O(n^2)}$ is sharp for the functions $\hat{f}_\omega^{\mathcal{M}} : \mathbb{N} \to S$ with $S = (\mathbb{N}, +, \cdot, 0, 1)$.*

Now we turn to the closure properties. We mainly studied the closure properties of the functions from $F_{\Sigma,S}^{RRWW}$. Until now, we considered three semirings, including Boolen semiring, the semiring of natural numbers with addition and multiplication, and tropical semiring. But we guess that this result holds for all semirings. As shown in Table 1, the functions from $F_{\Sigma,S}^{RRWW}$ are closed under pointwise addition, Cauchy product, and scalar multiplication. But it is still open, if they are also closed under pointwise multiplication.

4 Future Work

There are many open problems on weighted restarting automata. Firstly, as mentioned above, we considered only three semirings. It is still unknown whether the functions from $F_{\Sigma,S}^{RRWW}$ are closed also for other semirings under those operations.

Another question is whether the restrictions on the Rewrite operation influence the class of representable functions. For example, as introduced above, RRW-automata and RR-automata have more restrictions on the Rewrite operations than RRWW-automata. However, until now, we do not know whether RRWW-automata can represent more functions than RRW-automata and RR-automata. In other words, it remains the question of whether the inclusions $F_{\Sigma,S}^{RR} \subseteq F_{\Sigma,S}^{RRW} \subseteq F_{\Sigma,S}^{RRWW}$ are proper.

Additionally, by using the semiring of formal languages over Γ (i.e. $(\mathcal{P}(\Gamma^*), \cup, \cdot, \varnothing, \{\varepsilon\})$), the weight of a computation is essentially a language over Γ, and summing the weight of all accpeting computations yields a further language. Now the question arises, which languages can be

	$\mathbb{B}$	$\mathbb{N}$	$\mathbb{T}$
Pointwise Addition	$\surd$	$\surd$	$\surd$
Pointwise Multiplication	?	?	?
Cauchy Product	$\surd$	$\surd$	$\surd$
Scalar Multiplication	$\surd$	$\surd$	$\surd$

Table 1: The closure properties of the functions from $F^{RRWW}_{\Sigma,S}$. $\mathbb{B}$ denotes Boolean semiring $(\{1,0\},\vee,\wedge,0,1)$. $\mathbb{N}$ denotes the semiring of natural numbers $(\mathbb{N},+,\cdot,0,1)$. $\mathbb{T}$ denotes tropical semiring $(\mathbb{N},\min,+,\infty,0)$.

expressed by the functions $f^{\mathcal{M}}_{\omega}:\Sigma^*\to S$. Restarting transducer [5] introduced by Hundeshagen and Otto is a model for computing a binary word relation between the input and output. A restarting transducer works similarly as a restarting automaton. The difference is only that a restarting transducer outputs a word over an output alphabet after performing a Restart operation, and there is no output for other operations. Therefore, the restarting transducer is a special case for the weighted restarting automaton with the semiring $(\mathcal{P}(\Gamma^*),\cup,\cdot,\varnothing,\{\varepsilon\})$.

In future, we will continue studying these problems.

References

[1] M. DROSTE, S. DÜCK, Weighted Automata and Logics for Infinite Nested Words. In: *LATA*. 2014, 323–334.

[2] M. DROSTE, D. GÖTZE, The support of nested weighted automata. In: *NCMA*. 2013, 101–116.

[3] M. DROSTE, W. KUICH, Weighted finite automata over hemirings. *Theoretical Computer Science* 485 (2013), 38–48.

[4] M. DROSTE, W. KUICH, H. VOGLER, *Handbook of Weighted Automata*. 1st edition, Springer Publishing Company, Incorporated, 2009.

[5] N. HUNDESHAGEN, F. OTTO, Characterizing the Rational Functions by Restarting Transducers. In: *LATA*. 2012, 325–336.

[6] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Restarting Automata. In: H. REICHEL (ed.), *FCT*. Lecture Notes in Computer Science 965, Springer, 1995, 283–292.

[7] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On restarting automata with rewriting. In: G. PUN, A. SALOMAA (eds.), *New Trends in Formal Languages*. Lecture Notes in Computer Science 1218, Springer Berlin Heidelberg, 1997, 119–136.

[8] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, Different Types of Monotonicity for Restarting Automata. In: V. ARVIND, S. RAMANUJAM (eds.), *Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science 1530, Springer Berlin Heidelberg, 1998, 343–354.

[9] P. JANČAR, F. MRÁZ, M. PLÁTEK, J. VOGEL, On Monotonic Automata with a Restart Operation. *Journal of Automata, Languages and Combinatorics* 4 (1999) 4, 287–311.

[10] H. MESSERSCHMIDT, F. OTTO, Cooperating Distributed Systems of Restarting Automata. *International Journal of Foundations of Computer Science* 18 (2007) 6, 1333–1342.

[11] F. MRÁZ, Lookahead Hierarchies of Restarting Automata. *Journal of Automata, Languages and Combinatorics* 6 (2001) 4, 493–506.

[12] F. OTTO, Restarting Automata. In: Z. ÉSIK, C. MARTÍN-VIDE, V. MITRANA (eds.), *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence 25, Springer, 2006, 269–303.

[13] D. PARDUBSKÁ, M. PLÁTEK, F. OTTO, On Parallel Communicating Grammar Systems and Correctness Preserving Restarting Automata. In: A. DEDIU, A. IONESCU, C. MARTN-VIDE (eds.), *Language and Automata Theory and Applications*. Lecture Notes in Computer Science 5457, Springer Berlin Heidelberg, 2009, 660–671.

[14] M. P. SCHÜTZENBERGER, On the Definition of a Family of Automata. *Information and Control* 4 (1961) 2-3, 245–270.

[15] M. STRAKOV, Selected Types of Pg-Ambiguity: Processing Based on Analysis by Reduction. In: P. SOJKA, I. KOPEEK, K. PALA (eds.), *Text, Speech and Dialogue*. Lecture Notes in Computer Science 1902, Springer Berlin Heidelberg, 2000, 139–144.

Author Index