

TRANSDUCTIONS COMPUTED BY ITERATIVE ARRAYS

MARTIN KUTRIB AND ANDREAS MALCHER

Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
E-mail address: {kutrib,malcher}@informatik.uni-giessen.de

ABSTRACT. Iterative arrays are one-dimensional arrays of interconnected interacting finite automata. The cell at the origin is equipped with a one-way read-only input tape. We consider iterative arrays as transducers. To this end, the cell at the origin is additionally equipped with a one-way write-only output tape. The families of transductions computed are classified with regard to the time allowed to compute the input and the output, respectively. In detail, the time complexities of real-time and linear-time are of particular interest, for which a proper hierarchy is shown. In the second part of the paper, iterative array transducers are compared with the conventional transducer models, namely, finite state transducers and pushdown transducers. It turns out that all deterministic variants can be simulated by iterative array transducers. Moreover, nondeterministic but unambiguous finite state transducers can be simulated as well. When considering time constraints, incomparability results to almost all families are derived.

1. Introduction

Parallel processes and cooperating systems appear almost everywhere in today's world. However, the behavior of such systems, the interaction of different components, or the predictability of the behavior in the future is far from being completely understood. Iterative arrays (IA) are a model which allows to describe massive parallel systems, since they are arrays of identical copies of deterministic finite automata. Furthermore, the single nodes, except the node at the origin, are homogeneously connected to both their immediate neighbors, and they work synchronously at discrete time steps. The distinguished cell at the origin, which is called the communication cell, is equipped with a one-way read-only input tape.

In connection with formal language recognition IA have been introduced in [5]. To recognize a formal language, every word is read symbolwise from the input tape by the communication cell. An input is accepted if the communication cell enters an accepting state during the course of its computation. The computational capacity of IA has been widely studied in the literature and a recent survey may be found in [12].

Computational models are not only interesting from the viewpoint of recognizing some input, but also from the viewpoint of transforming some input into some

2000 ACM Subject Classification: F.1.1, F.1.2, F.4.3.

Key words and phrases: iterative arrays, transductions, finite state transducers, pushdown transducers.

output. For example, a parser for a formal language should not only return the information whether or not the input word can be parsed, but also the parse tree in the positive case. The simplest model in this context is the finite state transducer which is a finite automaton with an output alphabet that assigns to each input accepted at least one output word. Transductions computed by different variants of such transducers are studied in detail in [2]. Similarly, pushdown transducers are conventional pushdown automata where each input accepted is assigned to at least one output. Deterministic and nondeterministic variants are investigated in [1]. Furthermore, characterizations of pushdown transductions as well as applications to the parsing of context-free languages are given. In [9] the model of “one-way linear iterative arrays” is introduced. In this model an input is read at one end of the array and an output is written at the other end of the array. Additionally, the number of cells is a priori defined as the length of the input, the information flow is only from left to right, and the output does not depend on the fact whether or not the input is accepted.

In this paper, we will consider IA not only as a language recognizing device but as a language transforming device. Moreover, we are interested in the comparison with the conventional transducer models. To this end, we enhance the definition slightly by adding an output alphabet to an IA and, additionally, its communication cell is equipped with a one-way write-only output tape. Since IA are deterministic devices every input accepted corresponds to exactly one output. It is known that conventional IA can accept rather complicated languages such as $\{a^p \mid p \text{ is prime}\}$ [7] or $\{a^{2^n} \mid n \geq 1\}$ [4] in real time. Thus, we are interested in fast transductions of IAs as well and consider the time complexities of real-time and linear-time. Additionally, we consider the time complexities of accepting the input and computing the output separately.

The paper is organized as follows. In Section 2 we define iterative array transducers and their computed transductions with respect to the time complexities of real-time and linear-time. The relations of the families of transductions with certain time constraints are investigated in Section 3. It turns out that there exists a proper inclusion between the transductions where input and output are computed in real time and those where input and output are computed in linear time. Moreover, the transductions where the input is computed in real time and the output is computed in linear time are located properly in between both families. Section 4 is devoted to comparing iterative array transducers with finite state transducers. If the given finite state transducer is deterministic, it can be simulated by an iterative array transducer where input and output are computed in real time. This is no longer true, if the given finite state transducer is nondeterministic, but unambiguous. However, such finite state transducers can be simulated by iterative array transducers where the output is computed in linear time. Finally, we compare iterative array transducers with pushdown transducers in Section 5. The main result is that an iterative array transducer may need linear time to simulate a deterministic pushdown transduction. On the other hand, there are iterative array transductions where input and output are computed in real time which cannot be realized by any nondeterministic pushdown transducer. Thus, several incomparability results can be derived.

2. Preliminaries and Definitions

We denote the rational numbers by \mathbb{Q} , and the non-negative integers by \mathbb{N} . For the empty word we write λ , the reversal of a word w is denoted by w^R , and for the length of w we write $|w|$. The cardinality of a set M is denoted by $|M|$. We write \subseteq for set inclusion, and \subset for strict set inclusion.

A (one-dimensional) iterative array transducer is a linear array of finite automata, sometimes called cells, where each cell except the leftmost one is connected to its both nearest neighbors. The distinguished leftmost cell is the so-called communication cell that is connected to its neighbor to the right and to the input/output supply. For convenience we identify the cells by non-negative integers.

Initially, all cells are in the so-called quiescent state. At each time step the communication cell reads an input symbol and writes a possibly empty string of output symbols. To this end, we have different local transition functions. All cells but the communication cell change their state depending on their current state and the current states of their neighbors. The state transition and output of the communication cell depends on its current state, the current state of its neighbor, and on the current input symbol (or if the whole input has been consumed on a special end-of-input symbol). The cells work synchronously at discrete time steps.

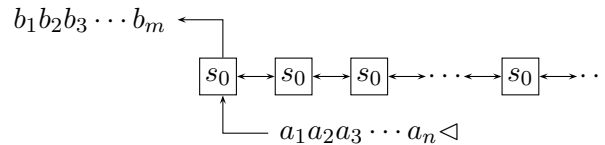


Figure 1: An iterative array transducer.

Definition 2.1. A *deterministic iterative array transducer* (IAT) is a system $\langle S, A, B, F, \triangleleft, s_0, \delta, \delta_0 \rangle$, where

- (1) S is the finite, nonempty set of *cell states*,
- (2) A is the finite, nonempty set of *input symbols*,
- (3) B is the finite set of *output symbols*,
- (4) $F \subseteq S$ is the set of *accepting states*,
- (5) $\triangleleft \notin A$ is the *end-of-input symbol*,
- (6) $s_0 \in S$ is the *quiescent state*,
- (7) $\delta : S^3 \rightarrow S$ is the *total local transition function for non-communication cells* satisfying $\delta(s_0, s_0, s_0) = s_0$,
- (8) $\delta_0 : (A \cup \{\triangleleft\}) \times S^2 \rightarrow B^* \times S$ is the *partial local transition function for the communication cell*.

Let \mathcal{M} be an IAT. A *configuration* of \mathcal{M} at some time $t \geq 0$ is a description of its global state which is a triple (w_t, v_t, c_t) , where $w_t \in A^*$ is the remaining input sequence, $v_t \in B^*$ is the output emitted so far, and $c_t : \mathbb{N} \rightarrow S$ is a mapping that maps the single cells to their current states. The configuration (w_0, λ, c_0) at time 0 is defined by the input word w_0 , an empty output string, and the mapping $c_0(i) = s_0$, $0 \leq i$, while subsequent configurations are chosen according to the global transition function Δ . Let (w_t, v_t, c_t) , $t \geq 0$, be a configuration. Then its successor configuration $(w_{t+1}, v_{t+1}, c_{t+1}) = \Delta(w_t, v_t, c_t)$ is as follows.

$$c_{t+1}(i) = \delta(c_t(i-1), c_t(i), c_t(i+1)),$$

for all $i \geq 1$. Furthermore, let $a = \triangleleft$, $w_{t+1} = \lambda$ if $w_t = \lambda$, and $a = a_1, w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Then, for $\delta_0(a, c_t(0), c_t(1)) = (v', s)$ we have

$$c_{t+1}(0) = s \text{ and } v_{t+1} = v_t v'.$$

Thus, the global transition function Δ is induced by δ and δ_0 . The IAT halts when it enters a *halting* configuration, i.e., the transition function δ_0 is not defined for the current configuration. As usual we extend Δ to sequences of configurations and denote it by Δ^* . That is, Δ^0 is the identity, $\Delta^t = \Delta(\Delta^{t-1})$, $1 \leq t$, and $\Delta^* = \bigcup_{0 \leq t} \Delta^t$. Thus, $(w_t, v_t, c_t) \in \Delta^*(w, v, c)$ indicates that it is possible for \mathcal{M} to go from the configuration (w, v, c) to the configuration (w_t, v_t, c_t) in a sequence of zero or more steps.

An input w is accepted by an IAT \mathcal{M} if at some time t during the course of its computation the communication cell enters an accepting state. In most cases t will be greater than $|w|$, but it is no restriction to accept earlier. An iterative array transducer \mathcal{M} *transforms* input words $w \in A^*$ into output words $v \in B^*$. For a successful transformation \mathcal{M} has to accept the input, otherwise the output is not recorded:

$$\mathcal{M}(w) = v,$$

if w is accepted by \mathcal{M} and $(\lambda, v, c') \in \Delta^*(w, \lambda, c_0)$, and (λ, v, c') is a halting configuration. The *transduction realized by \mathcal{M}* , denoted by $T(\mathcal{M})$, is the set of pairs $(w, v) \in A^* \times B^*$ such that $\mathcal{M}(w) = v$.

Let $t_i, t_o : \mathbb{N} \rightarrow \mathbb{N}$, $n + 1 \leq t_i(n) \leq t_o(n)$, be two mappings. If for all $(w, v) \in T(\mathcal{M})$, the input w is accepted after at most $t_i(|w|)$ time steps and \mathcal{M} halts after at most $t_o(|w|)$ time steps, then \mathcal{M} is said to be of time complexity (t_i, t_o) and we write IAT_{t_i, t_o} . The family of transductions realized by IAT_{t_i, t_o} is denoted by $\mathcal{F}(\text{IAT}_{t_i, t_o})$. If t_i, t_o are the function $n + 1$, we call it *real time* and write *rt*. Since for nontrivial computations an IAT has to read at least one end-of-input symbol, real time has to be defined as $(n + 1)$ -time. If $t_i(n), t_o(n)$ are of the form $r \cdot n$, for some $r \in \mathbb{Q}$, $r \geq 1$, we call it *linear time* and write *lt*.

If we build the projection on the first components of $T(\mathcal{M})$, then the iterative array transducer degenerates to an iterative acceptor (IA). The projection on the first components is denoted by $L(T(\mathcal{M}))$. In order to clarify the notation we give an example.

Example 2.2. The transduction

$$T_{expo} = \{ (a^{2^n}, a^1 b a^1 b a^2 b a^4 b \cdots a^{2^{n-1}} b) \mid n \geq 1 \}$$

belongs to $\mathcal{F}(\text{IAT}_{rt, rt})$.

In [4], an iterative array has been presented that uses signals in order to construct the mapping $n \mapsto 2^n$ in real time, that is, the communication cell recognizes the time steps 2^n , $n \geq 1$. At initial time the communication cell emits a signal which moves with speed $\frac{1}{3}$ to the right. In addition, another signal is emitted which moves with maximal speed (speed 1). It bounces between the slow signal and the communication cell. One can verify that the signal passes through the communication cell exactly at the time steps 2^n , $n \geq 1$ (see Figure 2).

An $\text{IAT}_{rt, rt}$ \mathcal{M} which realizes T_{expo} basically simulates the time constructor for 2^n . It reads an input symbol a at every time step and writes the output ab when it is in a distinguished state at times 2^n , and outputs a otherwise. Finally, \mathcal{M} accepts and halts if and only if the end-of-input symbol appears while it is in a distinguished state. ■

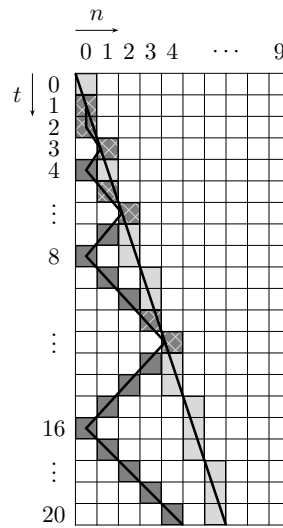


Figure 2: Space-time diagram showing signals of a time-constructor for the function $n \mapsto 2^n$.

One part of the computation of an IAT is the acceptance of the input. The previous example shows that iterative arrays can accept rather complicated languages even in real time. The language $\{a^{2^n} \mid n \geq 1\}$ is neither context free nor semilinear. Here, we consider iterative arrays from the computational point of view, that is, they are seen as massively parallel computing devices. So, we are mainly interested in fast computations, that is, small time complexities. In the sequel we focus on $\text{IAT}_{rt,rt}$, $\text{IAT}_{rt,lt}$, and $\text{IAT}_{lt,lt}$. For further reading on iterative language acceptors we refer to [10, 12].

3. Computational Capacity of Iterative Array Transducers

Roughly speaking, any transduction computed by an iterative array can be divided into two tasks. One is the acceptance of the input, the other one the transformation of the input into the output. Both tasks have to end successfully in order to obtain a valid computation. In particular, this observation implies that a language, which is not accepted by any iterative array in time t_i , cannot be the projection on the first components of any transduction belonging to any class $\mathcal{T}(\text{IAT}_{t_i,t_o})$.

Lemma 3.1. *The family $\mathcal{T}(\text{IAT}_{rt,lt})$ is strictly included in $\mathcal{T}(\text{IAT}_{lt,lt})$.*

Proof. The language $\{a, b\}^* \{w \mid w \in \{a, b\}^* \wedge |w| \geq 3 \wedge w = w^R\}$ is not accepted by any real-time IA [5], but can easily be accepted by some linear-time IA. Therefore, the transduction $\{(vw, a^{|vw|}) \mid v, w \in \{a, b\}^* \wedge |w| \geq 3 \wedge w = w^R\}$ is a witness for the assertion. ■

Since the complexity of the projections on the first components of an iterative array transduction is characterized by the power of iterative arrays when used as language acceptors, the question for the possible complexity of the projections on the second components follows immediately. It turns out that they can be arbitrarily complex within the limits of being computable.

Theorem 3.2. *Let L be an arbitrary recursively enumerable set. Then there is a transduction T belonging to $\mathcal{T}(\text{IAT}_{rt,rt})$ such that L is the projection on the second elements of T .*

Proof. Since L is recursively enumerable we may assume that it is represented by some deterministic one-tape one-head Turing machine \mathcal{M} such that any accepting computation has at least three and, in general, an odd number of steps. Therefore, it is represented by an even number of configurations. Moreover, we may assume that \mathcal{M} cannot print blanks, and that a configuration is halting if and only if it is accepting. A configuration of \mathcal{M} can be written as a string of the form Z^*SZ^* where $z_1 \cdots z_i s z_{i+1} \cdots z_n$ is used to express that \mathcal{M} is in state $s \in S$, scanning tape symbol z_{i+1} , and z_1 to z_n is the support of the tape inscription. The set of valid computations $\text{VALC}(\mathcal{M})$ is now defined to be the set of words of the form $w_1 \$ w_3 \$ \cdots \$ w_{2k-1} \pounds w_{2k}^R \$ \cdots \$ w_4^R \$ w_2^R$, where w_i are configurations, $\$$ and \pounds are symbols not appearing in w_i , w_1 is an initial configuration, w_{2k} is an accepting configuration, and w_{i+1} is the successor configuration of w_i , for $1 \leq i \leq 2k$.

For some $u = w_1 \$ w_3 \$ \cdots \$ w_{2k-1} \pounds w_{2k}^R \$ \cdots \$ w_4^R \$ w_2^R \in \text{VALC}(\mathcal{M})$ we define $I(u)$ to be the support of the tape inscription of w_1 , that is, the input word from L which is accepted by \mathcal{M} with the computation represented by u . In [12, 13] it is shown that the set of valid computations is a real-time IA language.

Now, the transduction $\{(u, v) \mid u \in \text{VALC}(\mathcal{M}), v = I(u)\}$ can be realized by some $\text{IAT}_{rt,rt}$ that writes $I(u)$ while reading the input prefix w_1 , and then continues to simulate an acceptor for $\text{VALC}(\mathcal{M})$ whereby the empty word is written in every step. ■

Next we turn to separate the families $\mathcal{T}(\text{IAT}_{rt,rt})$ and $\mathcal{T}(\text{IAT}_{rt,lt})$. To this end, we recall the capability of iterative arrays to simulate data structures as pushdown stores, rings and queues without any loss of time. First we consider pushdown stores (stacks). The top of the stack is simulated by the communication cell. Assume without loss of generality that at most one symbol is pushed onto or popped from the stack at each time step. Then it suffices to use three additional tracks for the simulation. Let the three pushdown registers of each cell be numbered one, two, and three from top to bottom, and suppose that the third register is connected to the first register of the right neighbor. The content of the pushdown store is identified by scanning the registers in their natural ordering beginning in the communication cell, whereby empty registers are ignored. The pushdown store dynamics of the transition function is defined such that each cell prefers to have only the first two registers filled. The third register is used as a buffer (see Figure 3). Details of the construction can be found in [3, 6, 11].

Lemma 3.3. *The family $\mathcal{T}(\text{IAT}_{rt,rt})$ is strictly included in $\mathcal{T}(\text{IAT}_{rt,lt})$.*

Proof. The transduction $\{(u, u^R) \mid u \in \{a, b\}^*\}$ belongs to $\mathcal{T}(\text{IAT}_{rt,lt})$. An $\text{IAT}_{rt,lt}$ realizing the transduction simulates a pushdown store. It first reads and pushes u while the empty word is written. When the end-of-input symbol appears it accepts in real time, pops u^R from the stack and writes it with λ moves. Finally, it halts in linear time when the stack is emptied.

In contrast to the assertion assume that some $\text{IAT}_{rt,rt}$ \mathcal{M} realizes the transduction. On input $u = u_1 u_2 \cdots u_n$ long enough \mathcal{M} cannot write the first symbol of u^R before it reads the last symbol of u . Otherwise on input $u = u_1 u_2 \cdots u_{n-1} u'_n$, where $u_n \neq u'_n$, the same output prefix is written but $(u_1 u_2 \cdots u_{n-1} u'_n, u_n v_{n-1} \cdots v_1)$ does not belong to the transduction, for any $v_1, v_2, \dots, v_{n-1} \in \{a, b\}^*$. This implies, that u^R has to be written after reading the last input symbol in the single remaining step. If u has been chosen long enough, this is impossible. The contradiction shows that the transduction does not belong to $\mathcal{T}(\text{IAT}_{rt,rt})$. ■

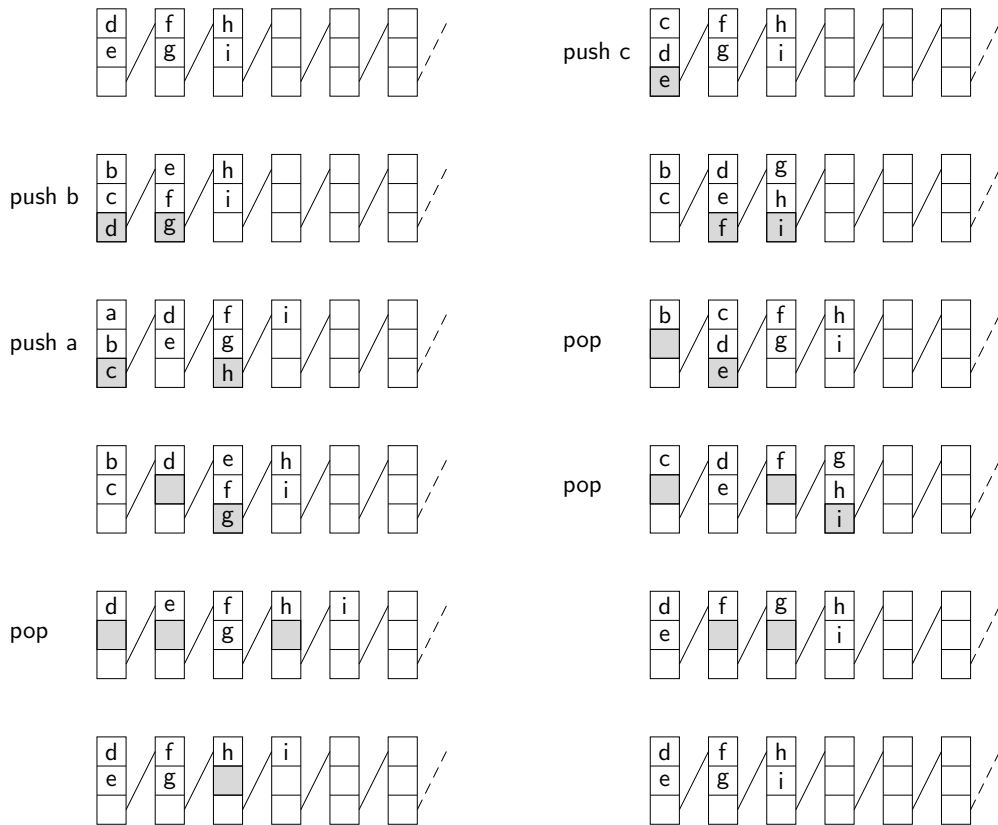


Figure 3: Principle of a pushdown store simulation. Subfigures are in row-major order.

We conclude this section with an example that utilizes the capability of an IA to simulate queues and rings without any loss of time (see [11]). The main difference between pushdown stores and rings or queues is the way how to access the data. A *ring* obeys the principle first in first out, that is, the first symbol of the stored string is read and possibly erased while, in addition, a new symbol may be added at the end of the string. So, a ring can write and erase at the same time. A *queue* is a special case of a ring. It can either write or erase a symbol, but not both at the same time.

Example 3.4. For any $k \geq 1$, the transduction $\{(u, u^k) \mid u \in \{a, b\}^*\}$ belongs to $\mathcal{T}(\text{IAT}_{rt,lt})$.

An $\text{IAT}_{rt,lt}$ realizing the transduction simulates a ring. It first reads and stores u where the first symbol of u in the ring is marked. When the end-of-input symbol appears the transducer accepts in real time. Then it starts to read u successively from the ring, where each symbol read is written and stored again into the ring. By the marked symbol the transducer recognizes when u has completely been processed. After the last process has been repeated k times it halts in linear time. ■

4. Comparison with and Simulation of Unambiguous Finite State Transducers

A nondeterministic finite state (rational) transducer is basically a nondeterministic finite automaton with output. At each time step the transducer reads a symbol

or the empty word from the input tape in some internal state, goes nondeterministically into another state, and writes a symbol or the empty word to the output tape. So, the partial transition function δ maps from $S \times (A \cup \{\lambda\})$ into the subsets of $S \times (B \cup \{\lambda\})$. Alternatively, one could allow that longer input words may be read and longer output words may be emitted at every time step. This generalization yields the same family of transductions. Here, we use the single symbol mode which is called standard form in [15]. A nondeterministic finite state transducer \mathcal{M} is said to be *single valued* (SFST) if for all $(u_1, v_1), (u_2, v_2) \in T(\mathcal{M})$ either $(u_1, v_1) = (u_2, v_2)$ or $u_1 \neq u_2$. An SFST is said to be *unambiguous* (UFST) if for all $(u, v) \in T(\mathcal{M})$ there is a unique computation transforming u into v . It has been shown in [14] that every single-valued finite state transducer can be simulated by an unambiguous one. A UFST is *deterministic* (DFST) if any computation is deterministic.

Since, in general, nondeterministic transducers can transform an input into different outputs, which is impossible for a deterministic device such as an IAT, we consider single-valued transducers only. While deterministic and nondeterministic finite automata accept the same family of languages, deterministic and nondeterministic finite state transducers have different power.

Lemma 4.1. *The family $\mathcal{T}(\text{DFST})$ is strictly included in $\mathcal{T}(\text{IAT}_{rt,rt})$.*

Proof. The transduction of Example 2.2 does not belong to $\mathcal{T}(\text{DFST})$ since the language $\{a^{2^n} \mid n \geq 1\}$ is not regular. On the other hand, any DFST can effectively be converted into an equivalent DFST without λ -moves [14], which in turn can be simulated in the communication cell of an $\text{IAT}_{rt,rt}$. ■

The next result shows that even the computational power of a massively parallel iterative array cannot compensate the presence of a little bit of nondeterminism.

Lemma 4.2. *The families $\mathcal{T}(\text{SFST})$ and $\mathcal{T}(\text{IAT}_{rt,rt})$ are incomparable.*

Proof. As in the previous lemma the transduction of Example 2.2 does not belong to $\mathcal{T}(\text{SFST})$ since $\{a^{2^n} \mid n \geq 1\}$ is not regular. So, it remains to be shown that there is a transduction belonging to $\mathcal{T}(\text{SFST})$ but not to $\mathcal{T}(\text{IAT}_{rt,rt})$. To this end, we use $T = \{(a^n c, a^n) \mid n \geq 1\} \cup \{(a^n d, b^n) \mid n \geq 1\}$ as witness. Transduction T is realized by an SFST that guesses initially whether the last input symbol is a c or a d . Accordingly it reads the input and emits a 's or b 's. If the guess was correct, the input is accepted and the transduction is recorded, otherwise the input is rejected and the transduction is not recorded.

Assume that some $\text{IAT}_{rt,rt}$ \mathcal{M} realizes T . On input $a^n c$, \mathcal{M} cannot write the first symbol a before it reads the last input symbol. Otherwise on input $a^n d$ always a symbol a would be emitted. So, a^n has to be written after reading the last input symbol in the sole remaining step. Since n can be arbitrary long, this is impossible. The contradiction shows that the transduction does not belong to $\mathcal{T}(\text{IAT}_{rt,rt})$. ■

Interestingly, if the iterative array is allowed to emit its output in linear time, the presence of a little bit of nondeterminism *can* be compensated.

Theorem 4.3. *The family $\mathcal{T}(\text{SFST})$ is strictly included in $\mathcal{T}(\text{IAT}_{rt,lt})$.*

Proof. Let $\mathcal{M} = \langle S, A, B, F, s_0, \delta \rangle$ be an unambiguous finite state transducer. By the construction given in [14], we may assume that \mathcal{M} works in real time.

The idea of the construction of an equivalent $\text{IAT}_{rt,lt}$ \mathcal{M}' is as follows. In order to find out whether an input w is accepted by \mathcal{M} , we first consider the nondeterministic

finite automaton (without output) $\mathcal{M}_{NFA} = \langle S, A, F, s_0, \delta' \rangle$ accepting $L(T(\mathcal{M}))$. It is converted into an equivalent deterministic finite automaton \mathcal{M}_{DFA} .

Automaton \mathcal{M}_{DFA} is simulated in the communication cell of \mathcal{M}' while reading the input w . Additionally, w is stored in a stack-like manner with the help of two tracks (see Figure 4). The IAT \mathcal{M}' accepts if and only if the simulation of \mathcal{M}_{DFA} accepts.

In order to compute the output of \mathcal{M} we have to determine the accepting computation path of \mathcal{M}_{NFA} on input w . As a first step, \mathcal{M}_{NFA} is converted into an equivalent right linear grammar \mathcal{G}_{NFA} with axiom X . The productions of \mathcal{G}_{NFA} have three forms, namely,

- (1) $X \rightarrow a[q']$ for all transitions $q' \in \delta'(s_0, a)$ such that $a \in A$,
- (2) $[q] \rightarrow a[q']$ for all transitions $q' \in \delta'(q, a)$ such that $q \in S, a \in A$,
- (3) $[q] \rightarrow a$ for all transitions $q' \in \delta'(q, a)$ such that $q \in S, q' \in F$, and $a \in A$.

So, each production in \mathcal{G}_{NFA} corresponds to a transition rule of \mathcal{M}_{NFA} and, thus, of \mathcal{M} . For each transition rule there is a unique output $z \in B^*$ emitted when the rule is applied.

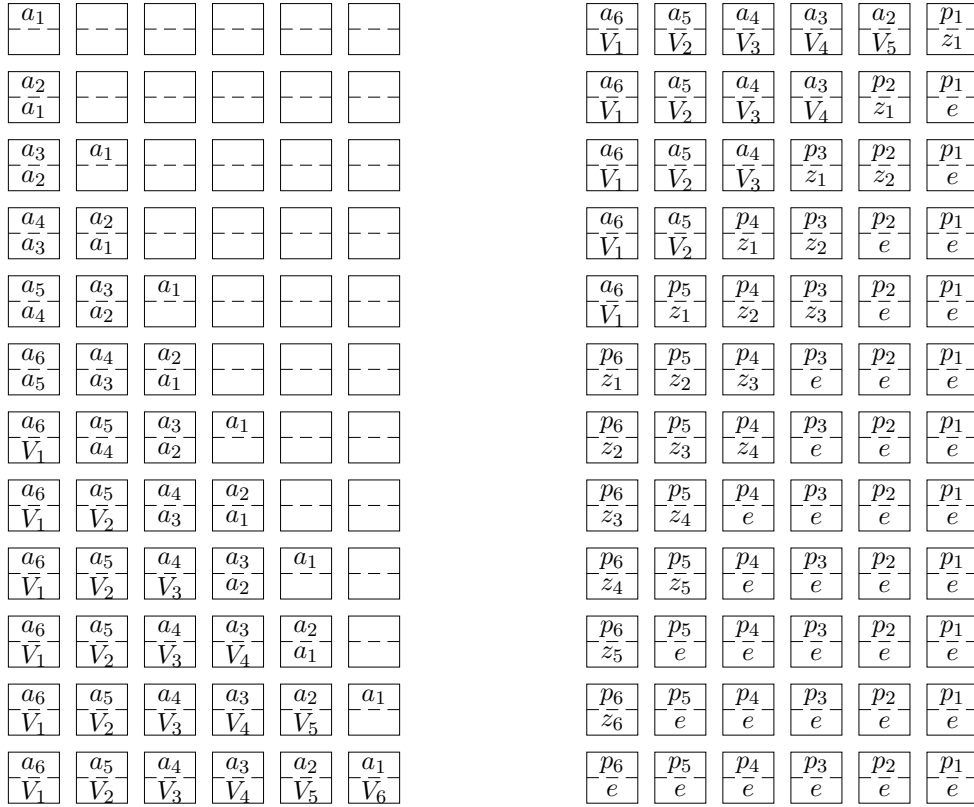


Figure 4: Schematic simulation of a single-valued finite state transducer on input $a_1a_2a_3a_4a_5a_6$. The first six cells of a simulating IAT $_{rt,lt}$ are depicted. In the first six time steps the input $a_1a_2a_3a_4a_5a_6$ is read, and in the last seven time steps the string $z_1z_2z_3z_4z_5z_6$ is emitted. The simulation of the deterministic finite automaton \mathcal{M}_{DFA} in the communication cell is not depicted.

When reading the end-of-input symbol, \mathcal{M}' starts to compute on an additional track all nonterminals of \mathcal{G}_{NFA} from which suffixes of the input can be derived. More

precisely, let $w = a_1 a_2 \cdots a_n$ and, thus, in the first n cells of some track, say the first one, $a_n a_{n-1} \cdots a_1$ is stored. Now, the first cell computes the set V_1 which includes all nonterminals Y for which the production $Y \rightarrow a_n$ belongs to \mathcal{G}_{NFA} , and stores it on a second track. Next, the second cell computes the set V_2 which includes all nonterminals Y for which the production $Y \rightarrow a_{n-1}[q']$ belongs to \mathcal{G}_{NFA} and $[q']$ belongs to V_1 . In general, the i th cell ($2 \leq i \leq n$) computes the set V_i which includes all nonterminals Y for which the production $Y \rightarrow a_{n-i+1}[q']$ belongs to \mathcal{G}_{NFA} and $[q']$ belongs to V_{i-1} . Since each V_i with $1 < i \leq n$ can be computed from V_{i-1} , a_{n-i+1} , and \mathcal{G}_{NFA} , the sets V_1, V_2, \dots, V_n can be computed in the first n cells in n time steps. It can be shown by induction that $Y \in V_i$ if and only if there is a derivation $Y \Rightarrow^* a_n a_{n-1} \cdots a_{n-i+1}$ of \mathcal{G}_{NFA} . Therefore, we obtain that $w \in L(\mathcal{G}_{NFA})$ if and only if $X \in V_n$.

Now, we can extract the accepting computation path from the sets V_i moving from right to left and starting in the n th cell as follows. First, some production $p_1 : X \rightarrow a_1 Y_1$ is chosen, where $Y_1 \in V_{n-1}$. The next productions p_2, p_3, \dots, p_{n-1} are chosen as $p_i : Y_{i-1} \rightarrow a_i Y_i$ such that $Y_i \in V_{n-i}$. Finally, p_n is chosen as some production $p_n : Y_{n-1} \rightarrow a_n$. The productions are stored on a third track of \mathcal{M}' and their computation takes n time steps.

Finally, the information on the output z_i of \mathcal{M} which is associated with each production p_i has to be sent to the communication cell where it is emitted. To this end, after having determined p_1 , the n th cell sends the information z_1 to the left. In the following time step, a signal e is sent to the left as well. The other cells i send their information z_i followed by e to the left when they receive the e from their neighbor to the right. The computation halts when the signal e arrives in the communication cell.

The overall simulation takes n time steps for reading the input, another n time steps to compute the sets V_i , and further $2n$ time steps to transmit signal e from the n th cell to the communication cell, that is, it takes linear time. ■

5. Iterative Arrays versus Pushdown Transducers

Similar as for finite state transducers, a nondeterministic pushdown transducer can be seen as nondeterministic pushdown automaton with output. So, the partial transition function δ maps from $S \times (A \cup \{\lambda\}) \times G$ into the finite subsets of $S \times B^* \times G^*$, where G denotes the stack alphabet. A nondeterministic pushdown transducer \mathcal{M} is said to be *single valued* (SPDT) if for all $(u_1, v_1), (u_2, v_2) \in T(\mathcal{M})$ either $(u_1, v_1) = (u_2, v_2)$ or $u_1 \neq u_2$. An SPDT is said to be *unambiguous* (UPDT) if for all $(u, v) \in T(\mathcal{M})$ there is a unique computation transforming u into v . As opposed to finite state transducers, single-valued pushdown transducers have more computational power than unambiguous pushdown transducers. For example, the transduction $T = \{ (a^n b^n a^m b^m, a^{2m+2n}) \mid m, n \geq 1 \} \cup \{ (a^n b^m a^m b^n, a^{2m+2n}) \mid m, n \geq 1 \}$ belongs to $\mathcal{T}(\text{SPDT})$ but not to $\mathcal{T}(\text{UPDT})$ because the projection on the first components is known to be an inherently ambiguous context-free language [8]. A UPDT is *deterministic* (DPDT) if any computation is deterministic, and it is *real-time deterministic* (DPDT $_\lambda$) if it is not allowed to move on empty input. Due to known results on the recognizability of context-free languages by different types of pushdown automata we have the proper hierarchy $\mathcal{T}(\text{DPDT}_\lambda) \subset \mathcal{T}(\text{DPDT}) \subset \mathcal{T}(\text{UPDT}) \subset \mathcal{T}(\text{SPDT})$.

We have already seen that the computational power of a massively parallel iterative array cannot compensate the presence of a little bit of nondeterminism. The same is true for the resource pushdown store equipped to a deterministic finite state device.

Lemma 5.1. *The family $\mathcal{T}(SFST)$ is incomparable with both families $\mathcal{T}(DPDT_\lambda)$ and $\mathcal{T}(DPDT)$.*

Proof. Since there are deterministic real-time context-free languages that are not regular there are transductions realized by some $DPDT_\lambda$ and $DPDT$ but not by any $SFST$.

Conversely, we define the homomorphism $h : \{a, b\} \rightarrow \{a', b'\}^*$ by $h(a) = a'$, $h(b) = b'$, and consider the transduction $T = \{(uc, h(u)) \mid u \in \{a, b\}^*\} \cup \{(ud, u) \mid u \in \{a, b\}^*\}$. An $SFST$ realizing T initially guesses whether the input ends with a c or with a d . Dependent on the guess it then computes the transduction as expected and records the transduction (accepts the input) when reading the last input symbol if and only if the guess was correct.

Now assume in contrast to the assertion that T is realized by some $DPDT$ \mathcal{M} . On input ud , \mathcal{M} cannot write the first output symbol a or b before it reads the last input symbol. Otherwise on input uc always a symbol a or b would be emitted. So, u has to be written after reading the last input symbol in a sequence of λ -moves. Next, a $DPDT$ \mathcal{M}' is constructed as follows. It starts to simulate \mathcal{M} until a d appears in the input. Then it continues the simulation but, in addition, tries to read the symbols it emits from the input. On input c , \mathcal{M}' rejects. So, \mathcal{M}' realizes the transduction $\{(udu, u) \mid u \in \{a, b\}^*\}$. This is a contradiction since $L(T(\mathcal{M}'))$ is not context free. ■

In order to draw an almost complete picture we next show the incomparability of transductions realizable by $DPDT$, $UPDT$ or $SPDT$, and $IAT_{rt,rt}$ or $IAT_{rt,lt}$, that is, the computational power of a massively parallel iterative array cannot compensate the presence of a pushdown store and vice versa.

Lemma 5.2. *Each of the families $\mathcal{T}(DPDT)$, $\mathcal{T}(UPDT)$, and $\mathcal{T}(SPDT)$ is incomparable with both families $\mathcal{T}(IAT_{rt,rt})$ and $\mathcal{T}(IAT_{rt,lt})$.*

Proof. The incomparability follows from the incomparability of the languages accepted by real-time iterative arrays and deterministic as well as nondeterministic context-free languages [12]. ■

However, by simulating pushdown stores as shown above $IAT_{rt,rt}$ can simulate $DPDT_\lambda$, and $IAT_{lt,lt}$ can simulate $DPDT$. Together with the previous incomparability results proper inclusions follow. It is known that Turing machines can simulate linear-time iterative arrays in quadratic time. On the other hand, it is not known whether every Turing machine working in quadratic time can be simulated by a linear-time iterative array. Since unambiguous context-free languages can be parsed in quadratic time using a variant of Earley's algorithm [1], it is an open problem to find out whether this approach can be applied for simulating $UPDT$ or $SPDT$ on linear-time iterative arrays.

Although nondeterministic devices in general have been excluded a priori, we conclude the section with an example emphasizing that there are structurally interesting “non-unary” transductions realizable by some $IAT_{rt,lt}$ but not realizable by any nondeterministic pushdown transducer.

Example 5.3. The transductions $\{(ucv, vcu) \mid u, v \in \{a, b\}^+\}$ and $\{(uc, u^Rcu) \mid u \in \{a, b\}^+\}$ belong to $\mathcal{T}(\text{IAT}_{rt,lt})$, but cannot be realized by any, even nondeterministic, pushdown transducer [1].

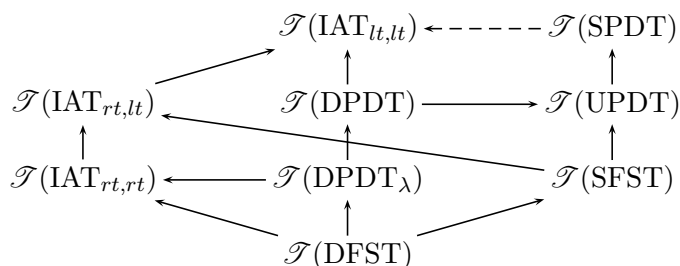


Figure 5: Summary of inclusions. Solid lines are proper inclusions, dashed lines are conjectured inclusions. All families which are not linked by a path are pairwise incomparable.

References

- [1] Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling. Volume I: Parsing. Prentice-Hall, Englewood Cliffs (1972)
- [2] Berstel, J.: Transductions and Context-Free-Languages. Teubner, Stuttgart (1979)
- [3] Buchholz, Th., Kutrib, M.: Some relations between massively parallel arrays. *Parallel Comput.* **23** (1997) 1643–1662
- [4] Choffrut, C., Čulik II, K.: On real-time cellular automata and trellis automata. *Acta Inform.* **21** (1984) 393–407
- [5] Cole, S.N.: Real-time computation by n -dimensional iterative arrays of finite-state machines. *IEEE Trans. Comput.* **C-18** (1969) 349–365
- [6] Čulik II, K., Yu, S.: Iterative tree automata. *Theoret. Comput. Sci.* **32** (1984) 227–247
- [7] Fischer, P.C.: Generation of primes by a one-dimensional real-time iterative array. *J. ACM* **12** (1965) 388–394
- [8] Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)
- [9] Ibarra, O.H., Jiang, T., Wang, H.: Parallel parsing on a one-way linear array of finite-state machines. *Theoret. Comput. Sci.* **85** (1991) 53–74
- [10] Kutrib, M.: Automata arrays and context-free languages. In *Where Mathematics, Computer Science and Biology Meet*. Kluwer Academic Publishers (2001) 139–148
- [11] Kutrib, M.: Cellular automata – a computational point of view. In *New Developments in Formal Languages and Applications*. Springer (2008) 183–227
- [12] Kutrib, M.: Cellular automata and language theory. In *Encyclopedia of Complexity and System Science*. Springer (2009) 800–823
- [13] Malcher, A.: On the descriptonal complexity of iterative arrays. *IEICE Trans. Inf. Syst.* **E87-D** (2004) 721–725
- [14] Weber, A., Klemm, R.: Economy of description for single-valued transducers. *Inform. Comput.* **118** (1995) 327–340
- [15] Yu, S.: Regular languages. In *Handbook of Formal Languages. Volume 1*. Springer, Berlin (1997) 41–110