Stack Transducers for Natural Language Interfaces

Suna Bensch[1], Johanna Björklund[1], and Martin Kutrib[2]

[1] Department of Computing Science,
Umeå University, 90187 Umeå, Sweden
email: {suna,johanna}@cs.umu.se
[2] Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
email: kutrib@informatik.uni-giessen.de

Abstract. We introduce and investigate stack transducers, which are one-way stack automata with an output tape. A one-way stack automaton is a classical pushdown automaton with the additional ability to move the stack head inside the stack without altering the contents. For stack transducers, we distinguish between a digging and a non-digging mode. In digging mode, the stack transducer can write on the output tape when its stack head is inside the stack, whereas in non-digging mode, the stack transducer is only allowed to emit symbols when its stack head is at the top of the stack. These stack transducers have a motivation from natural language interface applications, as they capture long-distance dependencies in syntactic, semantic, and discourse structures. We study the computational capacity for deterministic digging and non-digging stack transducers, as well as for their non-erasing and checking versions. We finally show that even for the strongest variant of stack transducers the stack languages are regular.

1 Introduction

Natural language interfaces are prevalent. We encounter them as automated booking services, as question-answering systems, and as intelligent personal assistants (Apple's Siri and Microsoft's Cortana belong to this category). As of recent, Google can support natural-language queries and exploratory dialogues. If the search engine is asked, in sequence, "Who is the president of the US?", "Where was he born?", "Who is his wife?", and finally "Where was she born?", it will interpret the questions as intended and perform the required anaphora resolution. For example, it will understand that the subject of the last question is the same entity as the second-to-last answer [13].

Natural language interfaces (NLI) have several advantages. They are fast and intuitive to use, and inclusive for social groups such as children, illiterates, and dyslectics. They allow for different modalities to input and output data, for example, microphones, speakers, keyboards, and terminals. For this reason, NLIs are accessible also while performing manual tasks, and open new possibilities for the disabled. On the downside, more is required on the side of the computer to process and represent natural language. In particular, efficient and reliable methods are needed to translate between NL sentences and structured data.

In natural language processing, translations are often done by transducers. These are abstract devices that map input strings, trees, or graphs to some target output domain. We find them in, for example, speech processing [12], machine translation [3], and increasingly in dialog systems [9]. A disadvantage of the currently used devices is that they cannot capture long-distance dependencies, as they interpret input words in the context of a very restricted history. However, the dependency structure is a determinative factor for syntactic, semantic and discourse analyses. In response, we introduce what we believe is a promising alternative, namely finite-state transducers with stacks that can be read, but not written, in their entirety throughout the execution. The aim is a balance between expressive power on the one hand, in particular the ability to model long-distance dependencies, and computational efficiency on the other.

This paper initiates the investigation of stack transducers. We begin with stack transducers in their unweigthed and deterministic form, though as the reader will see, this also produces results for more general devices in the passing.

Stack automata were introduced in [5] as a mathematical model of compilation, with a computational power in between that of pushdown automata and Turing machines. The stack automaton in [5] is a generalization of a pushdown automaton, as its input pointer can move to the right or left while reading input symbols, and its stack pointer can move *inside* the stack while reading stack symbols. The interior part of the stack cannot be altered, the operations push and pop are only allowed at the top of the stack. In [6] the authors restrict the stack automaton model to a *one-way* automaton that moves only to the right while reading input symbols. One-way nondeterministic stack automata can be simulated by deterministic linear-bounded automata, so the accepted languages are deterministic context sensitive [8]. Although compilation is a translation process from source code to object code, the authors of [5] focus on the acceptance of the input language.

We introduce stack transducers that are one-way stack automata with an output tape, to compute relations between input and output words. Like in [5], our devices are allowed to read information from the entire stack, but the operations push and pop are only allowed at the top of the stack. The stack pointer can thus move inside the stack, but the interior stack content cannot be altered. If the stack pointer is inside the stack, we say that the stack transducer is in *internal mode*. If the stack pointers scans the top most stack symbol, the transducer is said to be in *external mode*. In external mode the stack transducers can push or pop symbols from the top of the stack, or leave the stack unchanged, and are also allowed to write on the output tape with each operation. For stack transducers in internal mode, we distinguish between a digging and a non-digging[3] mode: In the former, the stack transducer can write on the output tape, whereas in the latter, the stack transducer is not allowed to output symbols. We believe that making the interior of the stack available as a read-only memory improves the expressiveness and space-efficiency of the transduction model, at a relatively low cost in terms of computational complexity.

Due to space limitations, most of our proofs can be found in the Appendix.

[3] The term 'digging' refers to the intuition of digging up soil from a deep hole.

2 Definitions and Preliminaries

Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. We denote the powerset of a set S by 2^S. We use $\subseteq$ for inclusion, and $\subset$ for proper inclusion.

A one-way stack automaton is a classical pushdown automaton with the additional ability to move the stack head inside the stack without altering the contents. In this way, it is possible to read but not to change the stored information. Well-known variants are so-called non-erasing stack automata, that are not allowed to pop from the stack, and checking stack automata, that are non-erasing stack automata which cannot push any symbols once the head has moved into the stack, even if it has then returned to the top. The devices are called 'transducers' if they are equipped with an output tape, to which they can append symbols during the course of the computation. More formally:

A *deterministic one-way stack transducer (abbreviated 1DSaT)* is a system $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F\rangle$, where Q is the finite set of *internal states*, Σ is the finite set of *input symbols*, Γ is the finite set of *stack symbols*, Δ is the finite set of *output symbols*, $q_0 \in Q$ is the *initial state*, $\bot \in \Gamma$ is the *initial stack* or *bottom-of-stack symbol*, $F \subseteq Q$ is the set of *accepting states*, δ_{ext} is the *external transition function* mapping $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $Q \times (\Gamma^* \cup \{-1\}) \times \Delta^*$, ($\delta_{\text{ext}}$ is the next move mapping when the stack head is at the top of the stack. Here, -1 refers to the stack head moving down one symbol), δ_{int} is the *internal transition function* mapping $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $Q \times \{-1, 0, +1\} \times \Delta^*$, ($\delta_{\text{int}}$ is the next move mapping when the stack head is inside the stack. Here, 0 means that the stack head does not move, and $+1$ means the stack pointer moves up one cell).

A *configuration* of a stack transducer $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F\rangle$ at some time $t \geq 0$ is a quintuple (q, w, s, p, u) where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, $s \in \Gamma^*$ gives the current stack content with the topmost symbol left, $1 \leq p \leq |s|$ gives the current position of the stack pointer from the top of the stack, and $u \in \Delta^*$ gives the content of the current output tape. The *initial configuration* for input w is set to $(q_0, w, \bot, 1, \lambda)$.

During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$, and the reflexive and transitive (resp., transitive) closure of $\vdash$ by $\vdash^*$ (resp., $\vdash^+$). Let $a \in \Sigma \cup \{\lambda\}$, $x \in \Sigma^*$, $Z \in \Gamma$, $z, y \in \Gamma^*$, $u, v \in \Delta^*$, and $p, q \in Q$. We set

1. $(q, ax, Zy, u, 1) \vdash (p, x, zy, uv, 1)$ if $\delta_{\text{ext}}(q, a, Z) = (p, z, v)$, (push/pop, stack head on top),
2. $(q, ax, Zy, u, 1) \vdash (p, x, Zy, uv, 2)$ if $\delta_{\text{ext}}(q, a, Z) = (p, -1, v)$, (go inside the stack from ext mode),
3. $(q, ax, Zy, u, i) \vdash (p, x, Zy, uv, i - d)$ if $\delta_{\text{int}}(q, a, Z) = (p, d, v)$, $2 \leq i \leq |Zy|$, $d \in \{-1, 0, +1\}$, (inside the stack, up, down, stay).

To simplify matters, we require that the bottom-of-stack symbol $\bot$ can neither be pushed onto nor be popped from the stack, and that the stack head never moves below the bottom of the stack. This normal form is always available through effective constructions.

In accordance with the language acceptors, a stack transducer is said to be *non-erasing* (1DNESaT) if it is not allowed to pop from the stack, that is, δ_{ext} maps $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $Q \times (\Gamma^+ \cup \{-1\}) \times \Delta^*$. A non-erasing stack transducer is *checking* (1DCSaT) if it cannot push any further symbol once the head has moved into the storage. In order to formalize this property, it is sufficient to partition the state set into $Q_1 \cup Q_2$ with $q_0 \in Q_1$ so that once the stack head is moved down, a state from Q_2 is entered and there is no transition from a state in Q_2 to a state in Q_1. That is, δ_{ext} maps $Q_1 \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $Q_1 \times \Gamma^+ \times \Delta^*$ or to $Q_2 \times \{-1\} \times \Delta^*$, and it maps $Q_2 \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $Q_2 \times \{-1, 0\} \times \Delta^*$, while δ_{int} is defined only for states from Q_2.

Finally, we distinguish two modes of writing to the output tape. So far, the stack transducers are allowed to write in any step, even if the stack head is not at the top. These transducers are called *digging* stack transducer (or simply, digger). In *non-digging* mode a stack transducer may only write when the stack head is at the top. Formally, this means that δ_{int} maps to $Q \times \{-1, 0, +1\}$. Non-digging stack transducers and their non-erasing and checking versions are abbreviated as ndi-1DSaT, ndi-1DNESaT, and ndi-1DCSaT.

A stack transducer *halts* if the transition function is not defined for the current configuration. It *transforms* an input word $w \in \Sigma^*$ into an output word $v \in \Delta^*$. For a successful computation M has to halt in an accepting state after having read the whole input, otherwise the output is not recorded: $M(w) = v$, where $(q_0, w, \bot, 1, \lambda) \vdash^* (p, \lambda, y, i, v)$ with $p \in F$, $1 \leq i \leq |y|$, and M halts in configuration (p, λ, y, i, v). The *transduction realized by* M, denoted by $\tau(M)$, is the set of pairs $(w, v) \in \Sigma^* \times \Delta^*$ such that $v = M(w)$. If we build the projection on the first components of $\tau(M)$, denoted by $L(M)$, then the transducer degenerates to a deterministic language acceptor.

The family of transductions realized by a device of type X is denoted by $\mathscr{T}(\mathsf{X})$.

In order to clarify our notion we continue with an example.

Example 1. The length-preserving transduction

$$\tau_1 = \{(a^n\$a^n\$a^m\$, a^n\$a^m\$a^n\$) \mid m \geq 0, n \geq 1\}$$

is realized by the non-digging stack transducer

$$M = \langle \{q_0, q_1, q_2, q_3, q_+\}, \{a, \$\}, \{A, \bot\}, \{a, \$\}, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, \{q_+\} \rangle,$$

where the transition functions are as follows.

(1) $\delta_{\text{ext}}(q_0, a, \bot) = (q_0, A\bot, a)$
(2) $\delta_{\text{ext}}(q_0, a, A) = (q_0, AA, a)$
(3) $\delta_{\text{ext}}(q_0, \$, A) = (q_1, A, \$)$
(4) $\delta_{\text{ext}}(q_1, a, A) = (q_1, -1, \lambda)$
(5) $\delta_{\text{int}}(q_1, a, A) = (q_1, -1, \lambda)$
(6) $\delta_{\text{int}}(q_1, \$, \bot) = (q_2, +1, \lambda)$
(7) $\delta_{\text{int}}(q_2, \lambda, A) = (q_2, +1, \lambda)$
(8) $\delta_{\text{ext}}(q_2, a, A) = (q_2, A, a)$
(9) $\delta_{\text{ext}}(q_2, \$, A) = (q_3, A, \$)$
(10) $\delta_{\text{ext}}(q_3, \lambda, A) = (q_3, \lambda, a)$
(11) $\delta_{\text{ext}}(q_3, \lambda, \bot) = (q_+, \bot, \$)$

Since δ_{int} never emits a symbol, M is non-digging.

Given an input $a^n\$a^n\$a^m\$$, the ndi-1DSaT M starts to read the prefix a^n with Transitions (1) and (2) whereby A^n is successively pushed onto the stack

and a^n is emitted. Then Transition (3) reads and writes the first \$ and sends M into state q_1 without changing the stack. State q_1 is used to move the stack head to the bottom of the stack while the next sequence of a's is read (Transitions (4) and (5)). Nothing is written during this phase. If the next \$ appears in the input exactly when the stack head reaches the bottom, the input prefix is $a^n\$a^n\$$ and M enters state q_2 with Transition (6). In state q_2 the stack head is moved to the top again (Transition (7)) whereby nothing is written to the output. At the top of the stack transition function δ_{ext} is applied again and M reads and emits the suffix $a^m\$$ with Transitions (8) and (9). The stack content is not changed in this phase. Finally, in state q_3 the stack is successively emptied with λ-moves and a^n is appended to the output tape (Transition (10)). The last λ-move at the bottom of the stack drives M into the accepting state q_+ while the concluding \$ is written (Transition (11)). ■

3 Computational Capacity

We turn to consider the computational capacity of the stack transducers. In particular, whenever two devices have different language acceptance power, then the simple identity transduction applied to a language from their symmetric difference would be a witness for separating also the power of the transducers. However, in the following we consider transductions of languages that are accepted by both types of devices in question. In this way, we are separating in fact the capabilities of computing transductions. First the relation with pushdown transducers (cf. [1]) is studied. A *pushdown transducer* (PDT) is a pushdown automaton equipped with a one-way write-only output tape. In our terms this is a stack automaton whose internal transition function δ_{int} is completely undefined. Our first result shows that pushdown transducers are strictly weaker than ndi-1DSaT, even if the language transformed is deterministic context free.

Theorem 2. *The length-preserving transduction*

$$\tau_1 = \{(a^n\$a^n\$a^m\$, a^n\$a^m\$a^n\$) \mid m \geq 0, n \geq 1\}$$

is a witness for the strictness of the inclusion $\mathscr{T}(\mathsf{PDT}) \subset \mathscr{T}(\mathsf{ndi\text{-}1DSaT})$.

The situation changes when the non-digging stack transducers are non-erasing. Clearly, the deterministic context-free language $\{ a^m\$a^n\$ \mid m \geq n \geq 1 \}$ is also accepted by a deterministic one-way checking stack automaton.

Lemma 3. *The transduction* $\tau_2 = \{(a^m\$a^n\$, a^{m-n}) \mid m \geq n \geq 1\}$ *belongs to the difference* $\mathscr{T}(\mathsf{PDT}) \setminus \mathscr{T}(\mathsf{ndi\text{-}1DNESaT})$.

Proof. A PDT realizing τ_2 is constructed from a real-time deterministic pushdown automaton that accepts the language $\{ a^m\$a^n\$ \mid m \geq n \geq 1 \}$. First the leading a's are read and pushed on the stack. When the first \$ appears, for every further input symbol a, one symbol is popped. Finally, the remaining symbols are popped and emitted.

In order to show that τ_2 is not realized by any ndi-1DNESaT we contrarily assume that it is realized by the ndi-1DNESaT $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F \rangle$.

We consider the situation when M has processed an input prefix $a^m\$$, for m large enough. Up to that time nothing can have been written on the output tape.

Otherwise, assume M has already written some word a^i with $i \geq 1$. Then the accepting computation on input $a^m\$a^m\$$ would produce a pair $(a^m\$a^m\$, a^j)$, for some $j \geq 1$, belonging to the transduction realized, but not to τ_2. Furthermore, by the same argumentation it follows that M cannot emit anything until the second $\$$ appears in the input, that is, until the input has been read entirely. Since M is non-erasing and non-digging, it has to write a^{m-n} on the tape with λ-moves and with the stack head on top of the stack. In between several write operations the stack head may move into the stack and back. The behavior of M in these phases can entirely be described by a table that lists for every state in which M moves the stack head into the stack what happens. This can either be halting or moving the state head back to the top in some state. Altogether there are only finitely many of such tables. We conclude that there are two numbers n_1 and n_2 so that $a^m\$a^{n_1}$ and $a^m\$a^{n_2}$ drive M into the same state, with the same topmost stack symbol, having the stack head on top, and the same table describing the behavior while the head is in the stack. So, if $a^m\$a^{n_1}\$$ is transformed into a^{m-n_1}, then so is $a^m\$a^{n_2}\$$, a contradiction. □

Since ndi-1DCSaT accept non-context-free languages the incomparabilities of the next corollary follow in general.

Corollary 4. *The family $\mathscr{T}$(PDT) is incomparable with each of the families $\mathscr{T}$(ndi-1DNESaT) and $\mathscr{T}$(ndi-1DCSaT).*

Moreover, the inclusion shown in Theorem 2 together with the transduction τ_2 belonging to the difference $\mathscr{T}(\mathsf{PDT}) \setminus \mathscr{T}(\mathsf{ndi\text{-}1DNESaT})$ by Lemma 3 reveals the strictness of the following inclusions. The inclusions themselves follows for structural reasons.

Corollary 5. *The family $\mathscr{T}$(ndi-1DSaT) properly contains the two families $\mathscr{T}$(ndi-1DNESaT) and $\mathscr{T}$(ndi-1DCSaT).*

Since the language recognition power of ndi-1DNESaT are stronger than that of ndi-1DCSaT there is a proper inclusion between the corresponding families of transductions. However, it is currently an open problem whether there is a ndi-1DNESaT M so that $L(M)$ is accepted by some ndi-1DCSaT as well, but $\tau(M)$ cannot be realized by any ndi-1DCSaT.

3.1 Digging versus Non-Digging

We turn to show that all types of stack transducers that are able to write to the output tape while the stack head is inside the stack are strictly stronger than their corresponding non-digging variant. To this end, the witness transduction $\tau_3 = \{ (a^n\$b^m\$, b^m\$a^n\$a^n\$) \mid m \geq 0, n \geq 1 \}$ is exploited.

Example 6. The transduction τ_3 is realized by the checking stack transducer

$$M = \langle \{q_0, q_1, q_2, q_3, q_+\}, \{a, b, \$\}, \{A, \bot\}, \{a, b, \$\}, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, \{q_+\} \rangle,$$

where the transition functions are as follows.

(1) $\delta_{\text{ext}}(q_0, a, \bot) = (q_0, A\bot, \lambda)$
(2) $\delta_{\text{ext}}(q_0, a, A) = (q_0, AA, \lambda)$
(3) $\delta_{\text{ext}}(q_0, \$, A) = (q_1, A, \lambda)$
(4) $\delta_{\text{ext}}(q_1, b, A) = (q_1, A, b)$
(5) $\delta_{\text{ext}}(q_1, \$, A) = (q_2, A, \$)$
(6) $\delta_{\text{ext}}(q_2, \lambda, A) = (q_2, -1, a)$
(7) $\delta_{\text{int}}(q_2, \lambda, A) = (q_2, -1, a)$
(8) $\delta_{\text{int}}(q_2, \lambda, \bot) = (q_3, 0, \$)$
(9) $\delta_{\text{int}}(q_3, \lambda, \bot) = (q_3, +1, a)$
(10) $\delta_{\text{int}}(q_3, \lambda, A) = (q_3, +1, a)$
(11) $\delta_{\text{ext}}(q_3, \lambda, A) = (q_+, A, \$)$

So, transduction τ_3 is realized by the weakest type of digging stack transducers, where the language $L(M)$ is regular. On the other hand, the next result shows that τ_3 is not even realized by the strongest type of non-digging stack transducers.

Lemma 7. *Transduction τ_3 does not belong to the family $\mathscr{T}$(ndi-1DSaT).*

Example 6 and Lemma 7 show the following proper inclusions.

Theorem 8.
1. $\mathscr{T}$(ndi-1DCSaT) $\subset$ $\mathscr{T}$(1DCSaT)
2. $\mathscr{T}$(ndi-1DNESaT) $\subset$ $\mathscr{T}$(1DNESaT)
3. $\mathscr{T}$(ndi-1DSaT) $\subset$ $\mathscr{T}$(1DSaT)

3.2 Relations Between Diggers

Here we compare the capacities of the three different types of stack transducers that may emit symbols while the stack head is inside the stack. Our first result separates the restricted families of non-erasing and checking transducers. Again, the witness transduction relies on an input language that is accepted by the weaker devices. We define $\tau_4 = \{\, (a^n\$a^n\$v\$, v^R\$a^n\$a^n\$) \mid n \geq 1, v \in \{a,b\}^* \,\}$. Transduction τ_4 is realized by some non-erasing stack transducer as shown in Example 18 in the appendix.

Based on transduction τ_4 the next separation is shown.

Theorem 9. *The length-preserving transduction τ_4 is a witness for the strictness of the inclusion $\mathscr{T}$(1DCSaT) $\subset$ $\mathscr{T}$(1DNESaT).*

With almost literally the same proof as in the previous theorem the next corollary can be shown.

Corollary 10. *The transductions $\{\, (a^n\$a^n\$v\$, v^R\$a^n\$) \mid n \geq 1, v \in \{a,b\}^* \,\}$ and $\{\, (a^m\$a^n\$v\$, v^R\$a^{m-n}\$) \mid m \geq 1, n \geq 0, v \in \{a,b\}^* \,\}$ do not belong to the family $\mathscr{T}$(1DCSaT).* □

The final comparison separates the most general family $\mathscr{T}$(1DSaT) of transductions considered here from the 'next' weaker family of transductions realized by non-erasing transducers. As before, the witness transduction relies on an input language that is accepted by the weaker devices. We define the transduction $\tau_5 = \{\, (a^m\$a^n\$v\$, v^R\$a^{m-n}\$) \mid m \geq 1, n \geq 0, v \in \{a,b\}^* \,\}$.

Example 11. The transduction τ_5 is realized by the stack transducer

$$M = \langle \{q_0, q_1, \ldots, q_4, q_+\}, \{a, b, \$\}, \{A, B, \$, \bot\}, \{a, b, \$\}, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, \{q_+\} \rangle,$$

where the transition functions are as follows. Let $X \in \{A, B, \$\}$.

(1) $\delta_{\text{ext}}(q_0, a, \bot) = (q_0, A\bot, \lambda)$
(2) $\delta_{\text{ext}}(q_0, a, A) = (q_0, AA, \lambda)$
(3) $\delta_{\text{ext}}(q_0, \$, A) = (q_1, A, \lambda)$

(4) $\delta_{\text{ext}}(q_1, a, A) = (q_1, \lambda, \lambda)$
(5) $\delta_{\text{ext}}(q_1, \$, \bot) = (q_2, \$\bot, \lambda)$
(6) $\delta_{\text{ext}}(q_1, \$, A) = (q_2, \$A, \lambda)$

(7) $\delta_{\text{ext}}(q_2, a, X) = (q_2, AX, \lambda)$
(8) $\delta_{\text{ext}}(q_2, b, X) = (q_2, BX, \lambda)$
(9) $\delta_{\text{ext}}(q_2, \$, \$) = (q_4, \lambda, \$)$
(10) $\delta_{\text{ext}}(q_2, \$, A) = (q_3, \lambda, a)$
(11) $\delta_{\text{ext}}(q_2, \$, B) = (q_3, \lambda, b)$

(12) $\delta_{\text{ext}}(q_3, \lambda, A) = (q_3, \lambda, a)$
(13) $\delta_{\text{ext}}(q_3, \lambda, B) = (q_3, \lambda, b)$
(14) $\delta_{\text{ext}}(q_3, \lambda, \$) = (q_4, \lambda, \$)$

(15) $\delta_{\text{ext}}(q_4, \lambda, A) = (q_4, \lambda, a)$
(16) $\delta_{\text{ext}}(q_4, \lambda, \bot) = (q_+, \bot, \$)$

Given an input $a^m\$a^n\$v\$$, the 1DSaT M starts to read the prefix a^m with the Transitions (1)–(3) whereby A^m is successively pushed onto the stack and nothing is emitted. Then Transition (4) reads the following a's as long as $n \leq m$, whereby as many stack symbols are popped as input symbols are read. If $n > m$, the computation blocks when the bottom-of-stack symbol appears. Otherwise, Transitions (5) and (6) read the next \$ and push it on the stack. Now the stack content is $\$A^{m-n}$. Next, state q_2 is used to read and push the input factor v by Transitions (7)–(11). When the last \$ appears in the input with a \$ at the top of the stack, then v is empty (Transition (9)). Finally, the stack content, that is, $v^R\$A^{m-n}$ is emitted by Transitions (12)–(15). In the last step, the closing \$ is emitted by Transition (16) that drives M into the accepting state $q+$. ■

Based on transduction τ_5 the next separation is shown.

Theorem 12. *The transduction τ_5 is a witness for the strictness of the inclusion $\mathscr{T}(\mathit{1DNESaT}) \subset \mathscr{T}(\mathit{1DSaT})$.*

Finally, we can derive the relationships between the family $\mathscr{T}(\mathsf{PDT})$ with all families of stack automata transductions considered. Since $\mathscr{T}(\mathsf{PDT})$ is properly included in $\mathscr{T}(\mathsf{ndi\text{-}1DSaT})$ (Theorem 2), it is properly included in $\mathscr{T}(\mathsf{1DSaT})$. With all other families we obtain incomparabilities as follows. By Corollary 4 there is a transduction in $\mathscr{T}(\mathsf{ndi\text{-}1DCSaT})$ not belonging to $\mathscr{T}(\mathsf{PDT})$. On the other hand, the stack transducer of Example 11 is in fact a pushdown transducer, since δ_{int} is completely undefined. So, transduction τ_5 belongs to $\mathscr{T}(\mathsf{PDT})$. But by Theorem 12 it does not belong even to $\mathscr{T}(\mathsf{1DNESaT})$. This implies the following corollary.

Corollary 13. *The family $\mathscr{T}(\mathit{PDT})$ is incomparable with each of the families $\mathscr{T}(\mathit{1DNESaT})$, $\mathscr{T}(\mathit{1DCSaT})$, $\mathscr{T}(\mathit{ndi\text{-}1DNESaT})$, and $\mathscr{T}(\mathit{ndi\text{-}1DCSaT})$.*

The inclusion structure of the families in question are depicted in Figure 1.

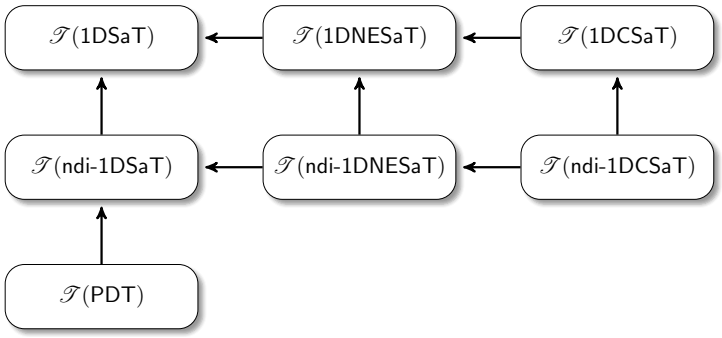

Fig. 1. Inclusion structure of transduction families realized by stack automata with different properties. The solid arrows indicate strict inclusions. The family $\mathscr{T}(\mathsf{PDT})$ is incomparable with all families to which it is not connected by a path.

4 Regularity of Stack Languages

It is well known that the set of reachable pushdown contents in a pushdown automaton is a regular language. Here we generalize this result to even the strongest type of stack transducer in question. Clearly, this implies the same result for stack automata as language acceptors.

The *stack language* of a stack transducer M is the set of all stack contents that occur in some configuration of a successful computation of M.

Before we turn to the proof of the main result in this section, we consider the notion of *stack trees* to model how the stack transducer transition function interacts with the stack. Intuitively, a stack tree t stores the stack contents as they appear in a computation, organized so that the right-most path from the root to a leaf of t holds the current stack. Without loss of generality, we restrict our attention to stack transducers that halt with an empty stack.

Definition 14 (Stack tree). *Let M be a stack transducer with stack alphabet Γ, and ρ be a computation of M on some input string w. The* stack tree t_ρ *of ρ is created as follows. At the start of the computation, the tree consists of a single node labeled $\bot$, and we place a pointer p at this node. This is the base case. Assume now that we have a stack tree t for the prefix ρ' of ρ, and that p marks one of its nodes. Depending on the next operation in ρ, the tree t is updated accordingly (see Figures 2 and 3):*

- *If M pushes the symbol $a \in \Gamma$ onto the stack, then*
  - *if p points to a leaf v, then a new leaf v' labeled a is created below it, an p is set to point to v', but*
  - *if p points to a non-leaf v at which a tree t' is rooted, then a new node v' is created below v, marked with the auxiliary symbol $\diamond$, t' is moved down and placed as the left child of v', and a new leaf $\hat{v}$ labeled a is added and placed as the right child of v'. The pointer p is set to $\hat{v}$.*
- *If M pops a symbol, then p is moved towards the root of t, to the closest ancestor node that is not an auxiliary node.*

Since ρ is a valid computation, t_ρ is well defined, and from the construction we know that it is binary.

From here on, we denote by S_Γ the set of all stack trees over the alphabet $\Gamma \cup \{\diamond\}$.

The following definition is illustrated in Figure 5.

Definition 15 (Composition operators). *Given $t, s \in S_\Gamma$, we denote by*

- *$t \otimes s$ the stack tree obtained by adding the root of s as a child of the right-most leaf of t, and*
- *$t \oplus s$ the stack tree obtained by creating a new node labeled with $\diamond$ and adding t and s as its left and right subtree, respectively.*

We denote by $\Gamma_{\oplus,\otimes}$ the set of stack trees that can be built from the symbols in Γ, seen as trees of height 0, and the operators $\oplus$ and $\otimes$.

Lemma 16. *For every stack alphabet Γ, we have $S_\Gamma = \Gamma_{\oplus,\otimes}$.*

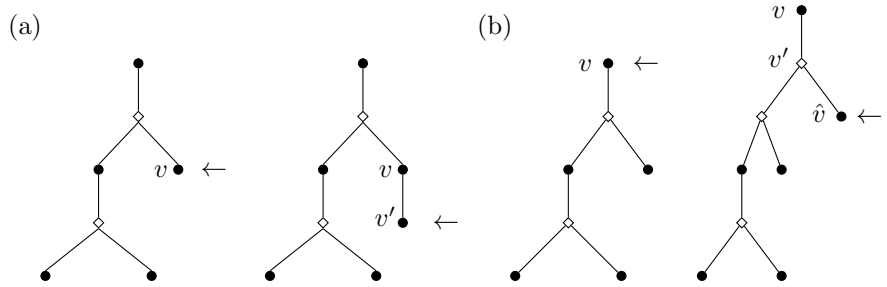

Fig. 2. Stack trees associated to a push operation of the stack transducer. Figure (a) shows a stack tree whose tree pointer points at a leaf node v, and after a push operation on the stack, a new leaf v' is created and p is set to point at v'. Figure (b) shows a stack tree whose tree pointer points at a non-leaf v, and after a push operation on the stack, a new node v' is created and marked with $\diamond$. The subtree that was rooted at v becomes the left child of v'.

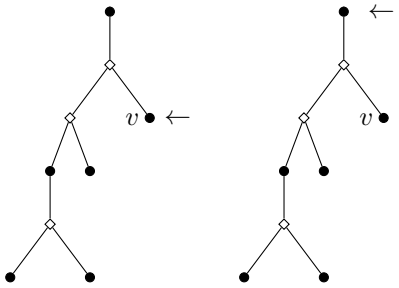

Fig. 3. Stack trees associated to a pop operation of the stack transducer. The figure shows a stack tree whose tree pointer points at a leaf node v, and after a pop operation on the stack, the tree pointer moves towards the root and to the closest ancestor node that is not an auxiliary node $\diamond$.

Proof. It is easy to see that every tree in $\Gamma_{\oplus,\otimes}$ is the stack tree for some choice of M, w, and ρ. So $\Gamma_{\oplus,\otimes} \subseteq S_\Gamma$.

The fact that every tree in S_Γ is in $\Gamma_{\oplus,\otimes}$ can be shown by induction on the height of the tree. The statement is true for all trees of height zero, that is, for Γ. Assume then that the inclusion holds for all trees of height n or less. The root of a tree t of height $n+1$ is either $\diamond$ or in Γ. In the first case, t can be constructed by applying the $\oplus$ operator to two trees of height at most n. In the second case, t can be constructed by applying the $\otimes$ operator to one tree in Γ and one tree of height at most n. By the induction hypotheses, all trees of height at most n can be constructed from Γ using $\oplus$ and $\otimes$. □

Now we are prepared to show the main result of this section.

Theorem 17. *The stack language of any stack transducer is regular.*

Proof (Sketch). Let $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F\rangle$ be a stack transducer. We associate every stack tree t with five *characteristic* functions: $\overset{\Rightarrow}{t}$, $\overset{\Leftarrow}{t}$, $\vec{t}$, $\overset{\leftarrow}{t}$,

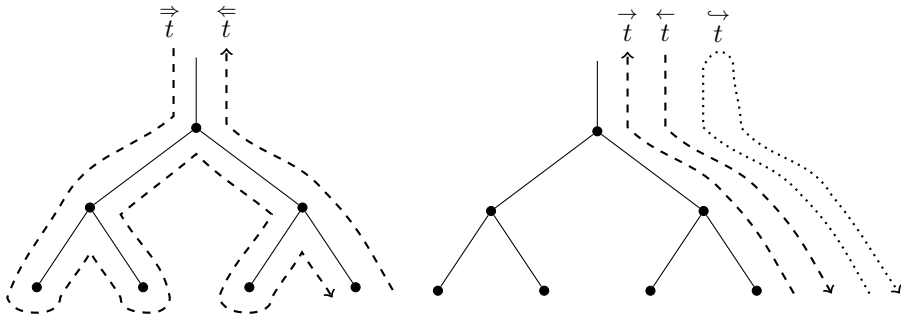

Fig. 4. The dashed and dotted lines indicate how the functions $\overset{\Rightarrow}{t}$, $\overset{\Leftarrow}{t}$, $\vec{t}$, $\overset{\leftarrow}{t}$, and $\overset{\hookrightarrow}{t}$, respectively and in order, propagate information through the stack tree.

$(t \overset{\hookrightarrow}{\otimes} s)$	=	$((\vec{s} \circ \overset{\hookrightarrow}{t} \circ \overset{\leftarrow}{s}) \cup \overset{\hookrightarrow}{s})^*$	$(t \overset{\hookrightarrow}{\oplus} s)$	=	$\overset{\hookrightarrow}{s}$
$(t \overset{\rightarrow}{\otimes} s)$	=	$(t \overset{\hookrightarrow}{\otimes} s) \circ \vec{s} \circ \vec{t}$	$(t \overset{\rightarrow}{\oplus} s)$	=	$\vec{s}$
$(t \overset{\leftarrow}{\otimes} s)$	=	$\overset{\leftarrow}{t} \circ \overset{\leftarrow}{s} \circ (t \overset{\hookrightarrow}{\otimes} s)$	$(t \overset{\leftarrow}{\oplus} s)$	=	$\overset{\leftarrow}{s}$
$(t \overset{\Rightarrow}{\otimes} s)$	=	$(t \overset{\hookrightarrow}{\otimes} s) \circ \overset{\Rightarrow}{s} \circ \overset{\Rightarrow}{t}$	$(t \overset{\Rightarrow}{\oplus} s)$	=	$\overset{\Rightarrow}{s} \circ \overset{\Leftarrow}{t} \circ \overset{\Rightarrow}{t}$
$(t \overset{\Leftarrow}{\otimes} s)$	=	$\overset{\Leftarrow}{t} \circ \overset{\Leftarrow}{s} \circ (t \overset{\hookrightarrow}{\otimes} s)$	$(t \overset{\Leftarrow}{\oplus} s)$	=	$\overset{\Leftarrow}{s}$

Table 1. The derivation of the characteristic functions for the trees $t \otimes s$ and $t \oplus s$, respectively, from the characteristic functions for t and s.

and $\overset{\hookrightarrow}{t}$ mapping Q to 2^Q. Intuitively, the functions $\overset{\Rightarrow}{t}$ and $\overset{\Leftarrow}{t}$ yield the sets of states that are reachable from any state q by building up or deleting stack tree t, respectively. That is, they describe how the states may change when the machine is in external mode, either pushing or popping the stack. The remaining functions yield the reachable states when reading the stack represented by t in internal mode. Here, it just runs up and down the stack, so the stack itself remains unaltered, but the steps it takes cause the states to change.

The characteristic functions are meant as a syntactic finger-print for a stack tree. The idea is that two stack-trees are interchangeable if their characteristic functions are the same.

By Lemma 16, every stack tree can be created from Γ through the operations $\otimes$ and $\oplus$, and the characteristic functions for $t \otimes s$ and $t \oplus s$ are uniquely defined from their counterparts on t and s (see Table 1). Since Q and hence $Q \to 2^Q$ are finite, this partitions the set of stack trees into an equivalence relation $\mathcal{E}$ of finite cardinality, that is a congruence with respect to $\otimes$ and $\oplus$. A stack tree t corresponds to an accepting computation of M if $\overset{\Leftarrow}{t}(\overset{\Rightarrow}{t}(q_0)) \cap F \neq \emptyset$, in which case we say that t is an *accepting stack tree* for M. Due to the above reasoning, a top-down nondeterministic tree automaton A can be built for the set of accepting stack trees for M, which makes this set a regular tree language.

It is known that the path languages of the regular tree languages and some larger language classes are regular string languages [4]. □

The idea of using characteristic functions to express the movements of transducers on their input is due to [2].

Theorem 17 shows that a stack transducer M with a one-way read-only input tape cannot be simulated by any stack transducer M' that receives its input directly on the stack. This holds even if M is deterministic, but M' is allowed to be nondeterministic. Moreover, since the intersection of regular languages is regular, any of the following ways of providing input to a stack transducer in place of the tape will cause the domain to be regular:

- the input is given on the stack,
- the machine guesses the input string and verifies its guess,
- the machine computes the input string on the stack, in such a way that the entire string is on the stack at once.

References

1. Aho, A.V., Ullman, J.D.: The theory of parsing, translation, and compiling, vol. I: Parsing. Prentice-Hall (1972)
2. Bojanczyk, M.: Transducers with origin information. Invited talk at the 3rd International Workshop on Trends in Tree Automata and Tree Transducers, The Queen Mary University of London (2015)
3. Braune, F., Seemann, N., Quernheim, D., Maletti, A.: Shallow local multi-bottom-up tree transducers in statistical machine translation. In: Association for Computational Linguistics (ACL 2013). vol. 1, pp. 811–821. The Association for Computer Linguistics (2013)
4. Drewes, F., van der Merwe, B.: Path languages of random permitting context tree grammars are regular. Fund. Inform. 82, 47–60 (2008)
5. Ginsburg, S., Greibach, S.A., Harrison, M.A.: Stack automata and compiling. J. ACM 14, 172–201 (1967)
6. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. J. ACM 14, 389–418 (1967)
7. Hibbard, T.N.: A generalization of context-free determinism. Inform. Control 11, 196–238 (1967)
8. Hopcroft, J.E., Ullman, J.D.: Sets accepted by one-way stack automata are context sensitive. Inform. Control 13, 114–133 (1968)
9. Hori, C., Ohtake, K., Misu, T., Kashioka, H., Nakamura, S.: Weighted finite state transducer based statistical dialog management. In: Automatic Speech Recognition & Understanding (ASRU 2009). pp. 490–495. IEEE (2009)
10. Kutrib, M., Wendlandt, M.: On simulation costs of unary limited automata. In: Shallit, J., Okhotin, A. (eds.) Descriptional Complexity of Formal Systems (DCFS 2015). LNCS, vol. 9118, pp. 153–164. Springer (2015)
11. Kutrib, M., Wendlandt, M.: Reversible limited automata. In: Durand-Lose, J., Nagy, B. (eds.) Machines, Computations, and Universality (MCU 2015). LNCS, vol. 9288, pp. 113–128. Springer (2015)
12. Mohri, M., Pereira, F.C.N., Riley, M.: Speech recognition with weighted finite-state transducers. In: Rabiner, L., Juang, F. (eds.) Handbook on speech processing and speech communication, Part E: Speech recognition. Springer (2008)
13. Petrov, S.: Towards Universal Syntactic and Semantic Processing of Natural Language. Invited talk at SLTC 2016, Uppsala University (2014)
14. Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. 25, 897–916 (2014)
15. Pighizzini, G., Pisoni, A.: Limited automata and context-free languages. Fund. Inform. 136, 157–176 (2015)
16. Wagner, K., Wechsung, G.: Computational Complexity. Reidel (1986)

Appendix

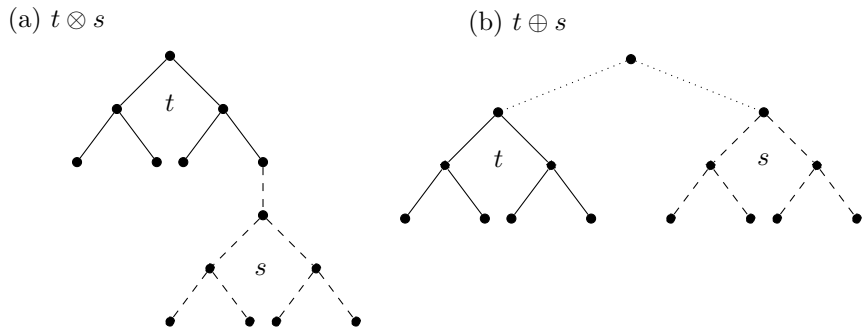

Fig. 5. Every stack tree can be composed from the stack alphabet and the auxiliary symbol $\diamond$ through repeated application of the operators $\otimes$ and $\oplus$.

Proof (of Theorem 2). The inclusion claimed follows for structural reasons. Transduction τ_1 belongs to $\mathscr{T}(\mathsf{ndi\text{-}1DSaT})$ by Example 1. So, it remains to be shown that τ_1 cannot be realized by any PDT.

In contrast to the assertion assume that the transduction τ_1 is realized by the PDT $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, q_0, \bot, F\rangle$. First, we consider input words $a^n\$a^n\$\$$ for n large enough. Clearly, after having read different prefixes $a^{n_1}\$$ and $a^{n_2}\$$ the PDT M must be in different states or have different stack contents. Therefore, the stack height grows with large n. This implies that M runs through a cycle whose length depends only on Q and Γ while processing the prefix $a^n\$$. Moreover, M runs through a cycle while processing the next factor a^n as well. If the stack height grows also in these cycles, M accesses only a constant number of symbols on top of its stack while processing (and verifying) the input $a^n\$a^n$. So, one can deduce a deterministic finite automaton accepting the non-regular language $\{\, a^n\$a^n \mid n \geq 1 \,\}$. We conclude that the stack height shrinks when M runs through cycles on the second input factor a^n.

Now assume that the stack height is not bounded by some constant after having processed $a^n\$a^n\$$, that is, it depends on n. It follows that for input words $a^n\$a^n\$\$$ the PDT M in the end either accepts with an arbitrary stack height or it runs through a cycle with λ-moves that empties the stack, possibly up to a constant number of symbols. So, an input $a^{n+i}\$a^n\$\$$, where i is a multiple of all cycle lengths, is accepted as well. We conclude that the stack height is bounded by a constant that depends only on Q and Γ after processing words of the form $a^n\$a^n\$$. Then there must be two different numbers n_1 and n_2 so that M is in the same state with the same stack content after having processed $a^{n_1}\$a^{n_1}\$$ and $a^{n_2}\$a^{n_2}\$$. If the output emitted so far contains at least two \$ we obtain a contradiction by extending the input by different suffixes. So, the outputs produced so far contain at most one \$, say $a^{n_1}\$a^{j_1}$ and $a^{n_2}\$a^{j_2}$. Continuing both computations with the same suffix $a^m\$$ emits the same output suffix u. So, even if $j_1 = j_2$ at most one of $a^{n_1}\$a^{j_1}u$ and $a^{n_2}\$a^{j_2}u$ is a correct output. The contradiction concludes the proof. □

Proof (of Lemma 7). In contrast to the assertion assume that transduction τ_3 is realized by some ndi-1DSaT M. In order to obtain a contradiction, we show that in this case a non-context-free language is accepted by a deterministic pushdown automaton M'.

We consider the situation when M has processed an input prefix $a^n\$$, for n large enough. Up to that time nothing can have been written on the output tape. Otherwise, assume M has already written some word that necessarily is of the form bv or $\$v$, for some $v \in \Delta^*$. Then either the accepting computation on input $a^n\$\$$ produces a pair $(a^n\$\$, bvu)$ or the accepting computation on input $a^n\$b\$$, produces a pair $(a^n\$\$, \$vu)$, for some $u \in \Delta^*$, belonging to the transduction realized, but not to τ_3. So, on input $a^n\$\$$ transducer M starts to write its output not before having read the second $\$$. Since M is non-digging, it has to write $\$a^n\$a^n\$$ on the tape with the stack head on top of the stack. This writing has to be performed by reading the sole input symbol $\$$ left and by λ-moves.

When processing the input prefix a^n, the stack head may move into the stack and back. During such excursions of the stack head the stack content cannot be changed, but M may read some input symbols a. The number of a's read may be a constant or depend on the stack content, but it cannot happen that M runs into a loop consuming all a's. Otherwise, the stack content would be the same for all longer prefixes which gives a contradiction.

As in the previous proof we use finitely many stack tables that list the behavior of M when its stack head is moved into the stack. For every state it is listed in which state the stack head returns to the top provided that nothing, only a's, or exactly one $\$$ are read during the excursion. If M does not return its stack head to the top under these conditions, the corresponding entry in the table is set to undefined. Additionally, it is listed if M halts successfully with the stack head in the stack while reading nothing in the excursion.

Now the pushdown automaton M' is constructed as follows. On input prefixes of the form a^n basically it simulates M. In addition a stack table is maintained as part of the state. Whenever M moves the stack head into the stack, M' uses the table as follows. If M returns its stack head to the top without reading any input symbol during the excursion, M' reads nothing as well. If M returns its stack head while reading one or more a's, M' reads precisely one a. In both cases M' enters the state listed in the table. In all other cases M' halts. Further, if M pushes some symbols, M' simulates this step by pushing the same symbols together with its current table and entering the new state of M. The new table dependent on the old one and the symbols pushed is computed and stored in the state. Whenever M pops a symbol, M' simulates this step by popping as well, entering the new state of M, and storing the table popped as new table in the state. Since M does not write while processing the prefix, M' simulates M except for the input read.

Next, we consider the infinitely many numbers $N = \{n_1, n_2, \dots\}$ so that M has its stack head on top of the stack after processing a^{n_i}, $i \geq 1$, and its next step is not a λ-move. These numbers exist since M has to push symbols from time to time to distinguish the number of a's read so far. For each of this numbers, M' reads a unique fixed number of a's, say, $f(n_i)$ many a's, where f is a function. On input $a^{n_i}\$\$$ the simulation of M' continues as follows. If M reads the $\$$ keeping the stack head at the top, M' simulates this step directly. Otherwise it simulates

the step and uses its table in order to know in which state to continue. Recall that the table provides information for the both cases that M will read nothing or the sole remaining input symbol \$ during an excursion. Subsequently, M starts to emit symbols with λ-moves, where at some time the last \$ may be read. These steps are simulated by M' directly or with the help of the table, whereby push and pops are simulated as before. However, M' does not write anything. Instead its reads the symbols written by M from the input. Since altogether M writes $\$a^{n_i}\$a^{n_i}\$$ and halts successfully, M' reads $\$a^{n_i}\$a^{n_i}\$$ and halts successfully.

Since any computation of M on $a^{n_i}\$\$$ yields an accepting computation of M' on $a^{f(n_i)}\$a^{n_i}\$a^{n_i}\$$, and any accepting computation of M' on $a^{f(n_i)}\$a^{n_i}\$a^{n_i}\$$ is based on a successful computation of M on $a^{n_i}\$\$$ that emits $\$a^{n_i}\$a^{n_i}\$$, we conclude that M accepts the language $L = \{\, a^{f(n_i)}\$a^{n_i}\$a^{n_i}\$ \mid n_i \in N \,\}$. A simple pumping argument shows that L is not context free. □

Example 18. The transduction τ_4 is realized by the non-erasing stack transducer

$$M = \langle \{q_0, q_1, \ldots, q_5, q_+\}, \{a, b, \$\}, \{A, B, \$, \bot\}, \{a, b, \$\}, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, \{q_+\}\rangle,$$

where the transition functions are as follows.

(1) $\delta_{\text{ext}}(q_0, a, \bot) = (q_0, A\bot, \lambda)$
(2) $\delta_{\text{ext}}(q_0, a, A) = (q_0, AA, \lambda)$
(3) $\delta_{\text{ext}}(q_0, \$, A) = (q_1, -1, \lambda)$

(4) $\delta_{\text{int}}(q_1, a, A) = (q_1, -1, \lambda)$
(5) $\delta_{\text{int}}(q_1, \$, \bot) = (q_2, 1, \lambda)$
(6) $\delta_{\text{int}}(q_2, \lambda, A) = (q_1, +1, \lambda)$

(7) $\delta_{\text{ext}}(q_2, \lambda, A) = (q_3, \$, \lambda)$
(8) $\delta_{\text{ext}}(q_3, a, X) = (q_3, AX, \lambda)$
(9) $\delta_{\text{ext}}(q_3, b, X) = (q_3, BX, \lambda)$
(10) $\delta_{\text{ext}}(q_3, \$, X) = (q_4, X, \lambda)$
(11) $\delta_{\text{ext}}(q_4, \lambda, A) = (q_4, -1, a)$

(12) $\delta_{\text{int}}(q_4, \lambda, A) = (q_4, -1, a)$
(13) $\delta_{\text{int}}(q_4, \lambda, B) = (q_4, -1, b)$
(14) $\delta_{\text{int}}(q_4, \lambda, \$) = (q_4, -1, \$)$
(15) $\delta_{\text{int}}(q_4, \lambda, \bot) = (q_5, +1, \$)$
(16) $\delta_{\text{int}}(q_5, \lambda, A) = (q_5, +1, a)$
(17) $\delta_{\text{int}}(q_5, \lambda, \$) = (q_+, 0, \$)$

■

Proof (of Theorem 9). By Example 18, transduction τ_4 belongs to $\mathscr{T}(\mathsf{1DNESaT})$. The inclusion itself follows for structural reasons. It remains to be shown that τ_4 does not belong to the family $\mathscr{T}(\mathsf{1DCSaT})$. In contrast to the assertion assume τ_4 is realized by the 1DCSaT $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F\rangle$.

First we note that M cannot write anything to the output tape before having read the last \$ from the input. Otherwise the input can always be extended such that the last symbol before the \$ does not match the first symbol already written. Therefore, the writing has to be performed entirely by λ-moves at the end of the input.

Since M is checking, the stack content cannot be changed after the stack head has moved into the stack. Assume that there is some $n_0 \geq 1$ such that M moves its stack head into the stack at some step while processing the input prefix

$a^{n_0}\$a^{n_0}\$$. In particular, this implies that for all $v \in \{a,b\}^*$ the successful computation on input $a^{n_0}\$a^{n_0}\$v\$$ starts to read v with the same stack content $s \in \Gamma^*$, the same stack head position $1 \leq p \leq |s|$, and the same state, and that the stack content does not change anymore. Since nothing is written until that time step, one can construct a finite-state transducer M' from M that realizes the transduction $\{\, (v\$, v^R\$) \mid v \in \{a,b\}^* \,\}$. To this end, the fixed stack content s as well as the stack head position is maintained as part of the states of M'. Now, while reading the input $v\$$, transducer M' simulates the behavior of M directly until the first $\$$ has been written. From now on, the simulation continues but nothing is written anymore.

Since it is well known that the transduction $\{\, (v\$, v^R\$) \mid v \in \{a,b\}^* \,\}$ cannot be realized by any finite-state transducer, we obtain a contradiction to the assumption that there is some $n_0 \geq 1$ such that M moves its stack head into the stack at some step while processing the input prefix $a^{n_0}\$a^{n_0}\$$.

Next, for all $n \geq 1$ we consider the input $a^n\$a^n\$\$$, where we know from above that M does not move the stack head into the stack before having read the second $\$$. Notice that M can only push a constant number of symbols in a consecutive sequence of λ-moves. Otherwise it would run into an infinite loop. The idea is to simulate M by so-called 1-*limited automata*. In general, a k-limited automaton is a linear bounded automaton that may rewrite each tape cell only in the first k visits, where k is a fixed constant. Afterwards the cells can still be visited any number of times, but without rewriting their contents [7, 10, 11, 14, 15]. The class of 1-limited automata captures the regular languages [16]. The idea of the construction of a 1-limited automaton M' that accepts the language $\{\, a^n\$a^n\$\$ \mid n \geq 1 \,\}$ is as follows. Automaton M' simulates M directly on the input prefix $a^n\$a^n\$$ on which M only pushes symbols. The symbols pushed by M are written on the current tape square by M'. If nothing is pushed, a special empty symbol is written. In this way, M' rewrites the visited part of its tape by the stack content pushed by M. After processing the input prefix, M' continues to simulate M directly but ignoring the output. The sole remaining input symbol is handled by M' in its states. When M moves its stack head, M' moves as well to find the stack symbol that M sees. Finally, M' accepts if M halts successfully. So, the 1-limited automaton M' accepts a non-regular language, a contradiction. We conclude that τ_4 does not belong to the family $\mathscr{T}(\mathsf{1DCSaT})$. □

Proof (of Theorem 12). By Example 11, transduction τ_5 belongs to $\mathscr{T}(\mathsf{1DSaT})$. The inclusion itself follows for structural reasons. It remains to be shown that τ_5 does not belong to the family $\mathscr{T}(\mathsf{1DNESaT})$. In contrast to the assertion assume τ_5 is realized by the $\mathsf{1DNESaT}$ $M = \langle Q, \Sigma, \Gamma, \Delta, \delta_{\text{ext}}, \delta_{\text{int}}, q_0, \bot, F \rangle$. As in the previous proof, we first note that M cannot write anything to the output tape before having read the last $\$$ from the input. Otherwise the input can always be extended such that the last symbol before the $\$$ does not match the first symbol already written. Therefore, the writing has to be performed entirely by λ-moves at the end of the input.

So, on input prefix $a^m\$a^n\$v$ the non-erasing transducer M reads input symbols, pushes symbols to the stack, and may move the stack head into the stack. Next we turn to show that M never reads more than $|Q|$ many consecutive symbols a or more than $|Q|$ many symbols from v without pushing. We distinguish

three similar cases and contrarily assume first that M reads $c > |Q|$ symbols from the prefix a^m without pushing. The computation is as follows, where the configuration with state q_1 is reached by a push operation and then there are no further push operations until the configuration with state q_2 is reached:

$$(q_0, a^m, \bot, \lambda, 1) \vdash^+ (q_1, a^{m-j}, u_1, \lambda, 1) \vdash^+ (q_2, a^{m-j-c}, u_1, \lambda, p_2).$$

Now we adjust the input to $a^{j+i}\$a^j\$v\$$, where $0 \leq i < c$, obtain the computation

$$(q_0, a^{j+i}\$a^j\$v\$, \bot, \lambda, 1) \vdash^+ (q_1, a^i\$a^j\$v\$, u_1, \lambda, 1) \vdash^+ (q_{2,i}, \$a^j\$v\$, u_1, \lambda, 1)$$

and consider the configuration that is reached just before the next push operation necessarily takes place if the input is long enough:

$$(q_1, a^i\$a^j\$v\$, u_1, \lambda, 1) \vdash^+ (q_{2,i}, \$a^j\$v\$, u_1, \lambda, 1) \vdash^+ (q_{3,i}, w_{3,i}, u_1, \lambda, 1).$$

where the next step is a push operation and $w_{3,i}$ is a non-empty suffix of $\$a^j\$v\$$. Since $0 \leq i < c$ and $c > |Q|$, there are at least two values for i, say i_1 and i_2, so that $q_{3,i_1} = q_{3,i_2} = q_3$ are equal. So, we have the computations

$$(q_0, a^{j+i_1}\$a^j\$v\$, u_1, \lambda, 1) \vdash^+ (q_3, w_{3,i_1}, u_1, \lambda, 1)$$

and

$$(q_0, a^{j+i_2}\$a^j\$v\$, u_1, \lambda, 1) \vdash^+ (q_3, w_{3,i_2}, u_1, \lambda, 1).$$

If $w_{3,i_1} = w_{3,i_2}$ the contradiction follows since the output for both different inputs would be the same. On the other hand, if $w_{3,i_1} \neq w_{3,i_2}$ let w_{3,i_1} be the longer one, that is, $w_{3,i_1} = w'w_{3,i_2}$, for some non-empty w. Continuing from configuration $(q_3, w_{3,i_1}, u_1, \lambda, 1)$ transducer M will emit $v^R\$a^{i_1}\$$ and continuing from configuration $(q_3, w_{3,i_2}, u_1, \lambda, 1)$ will generate the output $v^R\$a^{i_2}\$$. But now removing the factor w' from the input implies that on an input beginning with $a^{j+i_1}\$$ the output $v^R\$a^{i_2}\$$ is generated, a contradiction.

For the two remaining cases that M reads $c > |Q|$ symbols from the factor a^n or from the suffix v without pushing, contradictions are obtained along the same line. It may happen that M reads some \$ without pushing, but at most c symbols before and c symbols after the \$. Otherwise one of the cases from above applies. In total, there is a constant c_0 such that M never reads more than c_0 symbols successively without pushing, in particular with the stack head inside the stack.

Next, M is simulated by a *checking* stack transducer M'. To this end, again finitely many stack tables are used that list the behavior of M when its stack head is moved into the stack. For every state it is listed in which state the stack head returns to the top and which factors of the input is read during the excursion. At the end of the input, M' can simulate M directly by moving the stack head into the stack if necessary. Note, that at the end of the input M can only push a constant number of symbols without running into an infinite loop.

Now a contradiction follows by Corollary 10 saying that τ_5 does not belong to $\mathscr{T}(\mathsf{1DCSaT})$. □