# The Expressive Power of One Variable Used Once: The Chomsky Hierarchy and First-Order Monadic Constructor Rewriting

**Jakob Grue Simonsen**
Department of Computer Science, University of Copenhagen, Denmark

## Abstract

We study the implicit computational complexity of constructor term rewriting systems where every function and constructor symbol is unary or nullary. Surprisingly, adding simple and natural constraints to rule formation yields classes of systems that accept exactly the four classes of languages in the Chomsky hierarchy.

**2012 ACM Subject Classification** Theory of computation → Grammars and context-free languages; Theory of computation → Equational logic and rewriting; Theory of computation → Computability

**Keywords and phrases** Constructor term rewriting, Chomsky Hierarchy, Implicit Complexity

**Digital Object Identifier** 10.4230/LIPIcs.FSCD.2021.5

**Acknowledgements** I wish to thank the anonymous referees for diligent comments that have helped improve the presentation of the paper.

## 1 Introduction

A natural means of studying the expressive power of declarative programming languages is via constructor term rewriting systems; In these, the set of symbols are partitioned into *defined symbols* and *constructor symbols*, the former representing function names, and the latter representing data constructors.

The study of *implicit complexity* for a class of rewrite systems is, roughly, the study of the set of problems that can be accepted, decided, or otherwise characterized by the class. Implicit complexity has been studied extensively in functional programming (see – amongst many others – [4, 18, 22]), and in term rewriting [3, 2, 9, 19, 8].

In this paper, we study the implicit complexity of constructor term rewriting systems where all function and constructor symbols are restricted to have arity at most one (*monadic* systems); the rewriting systems are characterized according to the computational complexity of the constructor terms they accept. Unsurprisingly, the most general class of monadic systems accept the entire class of recursively enumerable sets. However, imposing simple and natural restrictions leads to exact characterization of the three other classes in the Chomsky hierarchy [7]: Context-sensitive, context-free, and regular languages. The results hold for the *unrestricted* rewriting relation, that is, we impose no evaluation order, and no typing beyond partitioning into sets of defined symbols and constructor symbols.

The restrictions we impose echo the usual intuition about classes in the Chomsky hierarchy: R.e. languages are accepted by Turing machines (finite state + two stacks), context-free languages by PDAs (finite state + one stack), regular languages by DFAs (finite state + no stacks), and context-sensitive languages by LBAs (finite-state + two stacks with a boundedness condition). The novel bits are that (i) we do not enforce machine-like restrictions on the rewrite relation (e.g., rewriting is not required to be innermost), and (ii) that both the encoding of the stacks and the behaviour of tail vs. general recursion have to be done with some finesse.


© Jakob Grue Simonsen;
licensed under Creative Commons License CC-BY 4.0

Classes:

| Type | Restriction on rules $l \to r$ | Example(s) of rule(s) | Expressive power |
|---|---|---|---|
| Unrestricted | None | $f(\mathtt{c}(x)) \to g(h((\mathtt{a}(\mathtt{d}(x)))))$ | RE |
| Non-length-increasing | $\lvert r\rvert \leq \lvert l\rvert$ | $f(\mathtt{c}(\mathtt{d}(x))) \to g(h(\mathtt{a}(x)))$ | CSL |
| (Strongly) cons-free | No constructor symbols in $r$ | $f(\mathtt{c}(x)) \to g(h(x))$, $f(\mathtt{c}(\mathtt{d}(x))) \to x$ | CFL |
| (Strongly) cons-free & tail recursive | cons-free, and the order of defined symbols in $r$ respect a certain preorder | $f(\mathtt{c}(x)) \to g(x)$, $f(x) \to f(g(h(x)))$ (with $f$ not appearing below $g$ or $h$ in the rhs of any rule with $g$ or $h$ in the lhs) | REG |

**Figure 1** Classes of monadic constructor TRSs and the classes of sets they accept.

Figure 1 gives an overview of the four classes of systems we consider and their relation to the language classes in the Chomsky hierarchy.

### Related work

For characterizing context-free and regular languages, we disallow constructors in the right-hand side of rules; this idea stems from Jones' work on the expressive power of higher-order types in functional programming [18] where a number of complexity classes were characterized in programs with call-by-value semantics and where functions may have arbitrary arity. Similar ideas have since been used in rewriting with less strict constraints on the evaluation order [9, 19], but for symbols with arbitrarily high finite arity. Correspondences between context-free languages and so-called *monadic recursion schemes* – essentially function declarations where all functions and data constructors are unary – were investigated some 40 years ago [14, 11, 10, 12]; the research focused mostly on decidability results, but close correspondences between monadic programs with very limited data construction abilites and context-free languages, was established there. Caron [6] proved undecidability of termination for non-length-increasing TRSs by encoding a certain class of linear bounded automata; we use a very similar approach to show that non-length-increasing constructor TRSs precisely accept the context-sensitive languages. Implicit complexity for term rewriting systems has been investigated in a number of papers; see the references above. Finally, the restriction to unary and nullary symbols means that all results in the paper can be viewed as concerning an especially well-behaved class of *string rewriting*; we refer the reader to [27, 28] for overviews of the correspondence between string rewriting and rewriting with unary symbols.

## 2 Preliminaries

We assume a non-empty alphabet, $A$, of *characters* and consider languages $L \subseteq A^+$ where $A^+$ is the set of non-empty strings of characters from $A$. The *empty* string over any alphabet will be denoted $\epsilon$. We presuppose general familiarity with the Chomsky hierarchy, including the four classes of *recursively enumerable languages* (RE, type-0), *context-sensitive languages* (CSL, type-1), *context-free languages* (CFL, type-2), and *regular languages* (REG, type-3).

Ample introductions can be found in [15, 26]. For (constructor) term rewriting, we refer to [27] for basic definitions; we very briefly recapitulate the most pertinent notions in the below definition.

▶ **Definition 1.** *We assume a denumerably infinite set* Var *of* variables*; given a signature* $\Sigma$ *of symbols with non-negative integer arities, we define the set of terms* $\mathrm{Ter}(\Sigma, \mathrm{Var})$ *over* $\Sigma$ *and* Var *inductively as usual:* $\mathrm{Var} \subseteq \mathrm{Ter}(\Sigma, \mathrm{Var})$ *and if* $s_1, \ldots, s_n \in \mathrm{Ter}(\Sigma, \mathrm{Var})$ *and* $f \in \Sigma$ *has arity* $n$*, then* $f(s_1, \ldots, s_n) \in \mathrm{Ter}(\Sigma, \mathrm{Var})$*.*

*A* rule *is a pair of terms, written* $l \to r$ *such that* $l$ *and* $r$ *are terms with* $l \notin \mathrm{Var}$ *and such that every variable occurring in* $r$ *occurs in* $l$*. A* term rewriting system *(abbreviated* TRS*) is a set of rules.*

*Let* $\Sigma = \mathcal{F} \cup \mathcal{C}$ *where* $\mathcal{F}$ *and* $\mathcal{C}$ *are disjoint sets of* defined symbols *and* constructor *symbols, respectively. A* constructor TRS *is a TRS where each rule* $l \to r$ *satisfies* $l = f(t_1, \ldots, t_n)$ *where* $f \in \mathcal{F}$ *and* $t_1, \ldots, t_n \in \mathrm{Ter}(\mathcal{C}, \mathrm{Var})$*.*

*A TRS is said to be* monadic *if the arity of all function and constructor symbols is at most* 1*. If* $R$ *is monadic and* $l \to r$ *is a rule of* $R$*, we occasionally write* $l(x) \to r(x)$ *where* $x$ *is the unique variable occurring in* $l$ *and* $r$ *(and we extend the notation to the case where there are no variables in* $l$ *or* $r$ *– in which case the choice of* $x$ *does not matter).*

*A* substitution *is a partial map* $\theta : \mathrm{Var} \longrightarrow \mathrm{Ter}(\Sigma, \mathrm{Var})$*. In monadic systems, each term* $s$ *contains at most one variable, and we shall write* $s\theta$ *for the term obtained by replacing the variable* $x$ *in* $s$ *by* $\theta(x)$ *(if* $x \in \mathrm{dom}(\theta)$*).*

*A* context *in a monadic TRS is a term over the variable set* $\mathrm{Var} \cup \{\Box\}$ *where* $\Box \notin \Sigma \cup \mathrm{Var}$*; if* $C$ *is a context and* $w$ *is a term, we denote by* $C[w]$ *the term obtained by replacing the (unique!)* $\Box$ *in* $C$ *by* $w$*. For* $s, t \in \mathrm{Ter}(\Sigma, \mathrm{Var})$*, we write* $s \to t$ *if there is a context* $C$*, a rule* $l \to r$*, and a substitution* $\theta$ *such that* $s = C[l\theta]$ *and* $t = C[r\theta]$*, and we call* $(C, l \to r, \theta)$ *a* redex *in* $s$*; The redex is said to be* contracted *in the* step $s \to t$*. The* position *of a redex is* $1^k$ *where* $k$ *is the number of symbols in* $C$ *(we set* $1^0 = \epsilon$*); we say that the rule* $l \to r$ *is* applied to $s$ at position $p$*. We write* $\to^*$ *for the reflexive, transitive closure of* $\to$ *and* $\to^+$ *as the transitive closure. We call* $s \to^* t$ *a* reduction *or* rewrite sequence*.*

*Two redexes* $(C, l \to r, \theta)$ *and* $(C', l' \to r', \theta')$ *in* $s = C[l\theta] = C'[l'\theta']$ overlap *if a symbol in* $l$ *and a symbol in* $l'$ *share the same position in* $C[l\theta] = C'[l'\theta']$*.*

*A redex* $v$ *at position* $p$ *in* $s$ *is* innermost *if, for any redex* $w$ *at position* $p' > p$*,* $w$ *overlaps* $v$ *(intuitively:* $v$ *is innermost if no other redex occurs "to the right of* $v$*").* The size *of a term* $s$ *in a monadic TRS is defined by induction as:* $|s| = 1$ *if* $s$ *is a variable or a nullary function symbol, and* $|s| = 1 + |s'|$ *if* $s = g(s')$ *where* $g \in \Sigma$*.*

Throughout the paper, we assume that all rewrite systems have a *finite* set of rules.

▶ **Definition 2.** *Let* $A$ *be an alphabet and* $\triangleright$ *a nullary constructor symbol. For every* $a \in A$*, we associate a unary constructor symbol* $\widetilde{a}$*, and we define* $\widetilde{A} = \cup_{a \in A} \{\widetilde{a}\}$*. For any string* $\alpha = a_1 \cdots a_n \in A^+$*, we associate the constructor term* $\widetilde{\alpha} = \widetilde{a}_1(\cdots \widetilde{a}_n(\triangleright))$*, and set* $\widetilde{\epsilon} = \triangleright$*.*

▶ Remark 3. Throughout the paper, *every* term is built from unary or nullary symbols. Hence, there is a natural correspondence between terms and strings: If $f_1, \ldots, f_m$ are unary symbols and $b$ is nullary, then $f_1(f_2(\cdots f_m(b)))$ corresponds to the string of symbols $f_1 f_2 \cdots f_m b$.

▶ **Definition 4.** *Let* $A$ *be an alphabet and let* $R$ *be a constructor TRS with* $\Sigma = \mathcal{F} \cup \mathcal{C}$*, such that* $(\widetilde{A} \cup \{\triangleright\}) \subseteq \mathcal{C}$ *where* $\triangleright$ *is a nullary symbol.* $R$ *is said to* accept $L \subseteq A^+$ *if there is a defined symbol* $f_0 \in \mathcal{F}$ *– the "start function" – such that for every* $\alpha \in A^+$*, there is a reduction* $f_0(\widetilde{\alpha}) \to^* \triangleright$ *iff* $\alpha \in L$*.*

▶ Remark 5. The use of $\triangleright$ in $f_0(\widetilde{\alpha}) \to^* \triangleright$ can be replaced by a fresh constructor (instead of the "nil" constructor $\triangleright$), or a nullary defined symbol when characterizing the classes RE or CSL. For CFL and RE, we consider systems where rules cannot contain any constructors in the right-hand side; there, acceptance by $\triangleright$–the last constructor in the representation $\tilde{\alpha}$ of any string $\alpha \in A^+$–is completely natural (it could wlog. be replaced by introducing rules of the form $f(\triangleright) \to h$ with $h$ a nullary symbol in $\mathcal{F}$, but there seems to be no good reason to do so).

▶ **Definition 6.** *Let $R$ be a monadic constructor TRS with alphabet $\Sigma = \mathcal{F} \cup \mathcal{C}$. $R$ is said to be* tail recursive *if there is a preorder $\leq$ on $\mathcal{F}$ such that for every rule $f(w) \to r$ in $R$ and every occurrence of a defined symbol $g \in \mathcal{F}$ in $r$, either (i) $f > g$, or (ii) $f \geq g$ and the occurrence of $g$ is at position $\epsilon$.*

The reason for requiring $\leq$ to be (only) a preorder as in [8] (rather than a partial order as in, e.g. [18]) is that recursion should be limited to tail calls (so, in rewriting terms, at the root of the rhs), but that the tail call does not need to be the same defined symbol as in the left-hand-side, merely a symbol having the same rank in the $\leq$-order.

## 3 Recursively enumerable languages: General monadic systems

For each of the class of languages we consider, we first remind the reader of their associated class of accepting machine; for recursively enumerable languages, these are Turing machines.

▶ **Definition 7.** *A (one-tape, non-deterministic)* Turing machine *is a tuple $(Q, A, \Gamma, \delta, q_0, q_h)$ where $Q$ is a set of states, $A$ is the input alphabet (which does not contain blanks), $\Gamma$ is the tape alphabet (with $A \subseteq \Gamma$ and a designated symbol $\square \in \Gamma$ representing "blank"), $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}) \times \{L, R\})$ is the transition function, $q_0 \in Q$ is the start state, and $q_h \in Q$ is the accept state.*

We write $\delta(q, a) \to (q', b, H)$ if $(q', b, H) \in \delta(a, b)$; note that several such transition rules may exist for each $(q, a)$. On a transition rule $\delta(q, a) \to (q', b, H)$, the machine is said to *transition*, when reading symbol $a$ in state $q$, to state $q'$, writing symbol $b$ (or not writing anything when $b = \epsilon$), and moving either left or right on the tape, according to whether $H = L$ or $H = R$.

As usual, we define Turing machine configurations as a (tape contents, tape head position, state)-triple:

▶ **Definition 8.** *A* configuration *of a machine $M = (Q, \Gamma, A, \delta, q_0, q_h)$ is a triple $(T, n, q)$ where $T \in \Gamma^+$ is the current content of the tape (disregarding the infinite strings of blanks to the left and right of the portion of the tape that contains the input and the set of cells scanned by $M$ so far; $T$ is assumed to have length at least 1, possibly consisting of a single blank symbol), $n$ is an integer where $1 \leq n \leq |T|$ (the position of the tape head in $T$), and $q \in Q$. $M$* transitions in one step *on configuration $(T, n, q)$ to configuration $(T', n', q')$* on transition *$\delta(q, b) \to (q', b', H)$ if the $n$th element of $T$ is $b$ and $(T', n', q')$ represents the machine state after the corresponding move. We say that "$A$ transitions to $B$" if $A$ reduces to $B$ by a sequence of $\geq 0$ steps. $M$* accepts *input $\alpha$ if it transitions from $(\alpha, 1, q_0)$ to a configuration $(T, n, q_h)$ – we also say that $M$ transitions to $q_h$ on input $\alpha$. A language $L \subseteq A^+$ is* recursively enumerable *if there is a Turing machine that, for each $\alpha \in A^+$, accepts $\alpha$ iff $\alpha \in L$.*

Huet and Lankford proved that monadic TRSs can simulate Turing machines [16]. For completeness, we re-prove Huet and Lankford's result in a new setting, giving a new proof of simulation of Turing machines by constructor TRSs with unary function and constructor

$M = (Q, A, \Gamma, \delta, q_0, q_h)$

Sets of function and constructor symbols:

$\mathcal{F} = \{f_q : q \in Q\} \cup \{f_b : b \in \Gamma\} \cup \{f_0, f_\triangleleft\}$
$\mathcal{C} = \{\mathtt{b} : b \in \Gamma\} \cup \{\mathtt{c}_q : q \in Q\} \cup \{\triangleright\}$

Note: $f_\square \in \mathcal{F}$ and $\square \in \mathcal{C}$. If $b \in A$, then $\mathtt{b} = \tilde{b}$. The constructors $\mathtt{c}_q$ are fresh symbols designed to "store" the state of $M$ on left-moves.

Rewrite rules induced by transition rules of $M$:

| (L/R)-move | rewrite rules (for each $q \in Q$, $b \in \Gamma$) |
|---|---|
| $\delta(q,b) \to (q',b',R)$ | $f_q(\mathtt{b}(x)) \to f_{b'}(f_{q'}(x))$ |
| $\delta(q,b) \to (q',\epsilon,R)$ | $f_q(\mathtt{b}(x)) \to f_b(f_{q'}(x))$ |
| $\delta(q,b) \to (q',b',L)$ | $f_q(\mathtt{b}(x)) \to \mathtt{c}_{q'}(\mathtt{b}'(x))$ |
| $\delta(q,b) \to (q',\epsilon,L)$ | $f_q(\mathtt{b}(x)) \to \mathtt{c}_{q'}(\mathtt{b}(x))$ |

Housekeeping rules for endmarkers:

| rewrite rules (for each $q \in Q \setminus \{q_h\}$) |
|---|
| $f_q(\triangleright) \to f_q(\square(\triangleright))$ |
| $f_\triangleleft(\mathtt{c}_q(x)) \to f_\triangleleft(f_q(\square(x)))$ |

Propagation and start rules:

| rewrite rules (for each $q \in Q$, $b \in \Gamma$) |
|---|
| $f_{q_h}(x) \to \mathtt{c}_{q_h}(x)$ |
| $f_\triangleleft(\mathtt{c}_{q_h}(x)) \to \triangleright$ |
| $f_b(\mathtt{c}_q(x)) \to f_q(\mathtt{b}(x))$ |
| $f_0(x) \to f_\triangleleft(f_{q_0}(x))$ |

**Figure 2** Basic encoding $\Delta^1(M)$ of a Turing machine $M$: The part of the tape to the left of the single read/write head is represented by a string of defined symbols $f$, and the part to the right by a string of constructor symbols $\mathtt{c}$.

symbols. The constructor TRS simulating a given Turing machine is given in Figure 2; the construction is similar to that given in [27], but uses only unary function and unary and nullary constructor symbols. The simulation serves as illustration of an observation we shall exploit throughout the paper: A term $f_1(\cdots f_m(\mathtt{c}_1(\cdots \mathtt{c}_n(\triangleright))))$ may be viewed as composed of a "call stack" of defined symbols and a "memory stack" of constructor symbols; intuitively, we thus obtain the expressive power of the (Turing-complete) class of 2-counter machines [23].

The following is tedious, but not hard, to prove:

▶ **Lemma 9.** *Let $\alpha \in A^+$. Then, $f_0(\widetilde{\alpha}) \to^* \triangleright$ iff $M$ transitions to $q_h$ on input $\alpha$.*

We then have:

▶ **Theorem 10.** *Let $L \subseteq A^+$. The following are equivalent: (i) $L$ is recursively enumerable, (ii) $L$ is accepted by a monadic constructor TRS.*

**Proof.** If $R$ is a monadic constructor TRS accepting $L$, we may construct a (non-deterministic) Turing machine accepting $L$ by encoding the finitely many rules of $R$ and non-deterministically applying the rules to input $\alpha$. If this simulation reaches $\triangleright$, the machine halts. It therefore suffices to prove that every recursively enumerable language is accepted by a monadic constructor TRS. By standard results (see e.g. [25, Thm. 17.2]), $L$ is recursively enumerable iff it is accepted by a non-deterministic Turing machine. Lemma 9 then furnishes that, for each $\alpha \in A^+$, we have $f_0(\widetilde{\alpha}) \to^* \triangleright$ iff $M$ accepts $\alpha$, as desired. ◀

## 4 Context-sensitive languages: Non-length-increasing rules

The class of context-sensitive languages is characterized by its class of acceptors: the linearly bounded automata. The following definition is standard.

▶ **Definition 11.** *A non-deterministic Turing (multi-tape) machine $M$ accepts $L \subseteq A^+$ in non-deterministic linear space if there is a $k$ such that all computation branches halts on all inputs and any computation scans at most $k \cdot |x|$ distinct cells on each of its tapes. The set of languages acceptable by such machines is called* NLINSPACE.

▶ **Proposition 12.** *Let $L \subseteq A^+$ be a language accepted by an $m$-tape Turing machine in space $O(n)$. Then there is a* one-tape Turing machine *with input alphabet $A$ (but possibly a much larger tape alphabet) that accepts $L$ in space $\leq n$ on all inputs.*

**Proof.** Standard exercise in linear space reduction, see e.g. [17, Prop. 21.1.5] for the reduction to one-tape machines (at the cost of an input-independent constant factor of more space use), and [24] for the technique of getting rid of constant space factors on one-tape machines. ◀

By Proposition 12 we may restrict attention to machines that accept their input using no more space than that originally allocated to the input: The linear bounded automata. To ensure that linear bounded automata do not exceed their tape allowance, we make the provision that inputs are always bookended by special *stoppers* ◀ and ▶. For example, if $A = \{0, 1\}$ the string 10010 will be fed to the automaton as ◀ 10010 ▶.

▶ **Definition 13.** *A* linear bounded automaton *(LBA) over alphabet $A$ is a one-tape Turing machine $(Q, A, \Gamma, \delta, q_0, q_h)$ with input alphabet $A' = A \cup \{◀, ▶\}$ and where ◀ and ▶ are the* left *and* right stoppers*, respectively, and such that: (i) ◀, ▶$\notin \Gamma$, (ii) for every rule $\delta(q, b) \to (q', b', H)$, we have $b' \notin \{◀, ▶\}$ (i.e. stoppers are not written on the tape), (iii) for every rule $\delta(q, ◀) \to (q', b', H)$, we have $b' = \epsilon$ and $H = R$ (i.e. the left stopper is not overwritten, and the tape head cannot move left of the left stopper), (iv) for every rule $\delta(q, ▶) \to (q', b', H)$, we have $b' = \epsilon$ and $H = L$ (the right stopper is not overwritten and the tape head cannot move to the right of the stopper endmarker). An LBA is said to* accept *input $\alpha \in A^+$ if its underlying Turing machine accepts the string ◀ $\alpha$ ▶.*

Thus: A linear bounded automaton can *only* use the space that its input originally occupies: Space *exactly* $n$ where $n$ is the size of the input. The following proposition makes this precise:

▶ **Proposition 14.** *Let $M$ be an LBA. If $(◀\ b_1 \cdots b_m\ ▶, n, q)$ is a configuration of $M$ and $b_1, \ldots, b_m \in \Gamma \backslash \{◀, ▶\}$, and $M$ transitions to configuration $(T', n', q')$, then $T' =◀\ b'_1 \cdots b'_m\ ▶$ for $b'_1, \ldots, b'_m \in \Gamma \setminus \{◀, ▶\}$.*

**Proof.** By the assumptions on the form of the rules of the LBA in Definition 13, neither of the symbols ◀ and ▶ can be overwritten by $M$, nor can any symbol be overwritten by ◀ or ▶. By the same assumptions on the form of rules, $M$ cannot move to the left of a ◀, nor to the right of a ▶. ◀

▶ **Theorem 15.** *Let $L \subseteq A^+$. The following are equivalent: (i) $L$ is accepted by an LBA, (ii) $L$ is context-sensitive, (iii) $L \in$* NLINSPACE.

**Proof.** Standard textbook exercise, see e.g. [21, Exerc. 6.29], or [25, Thm. 24.3]. For the original proof, see [20]. ◀

For every $(a, b, d) \in A^3$:

$M_{abd} = (Q_{abd}, A \cup \{\blacktriangleleft, \blacktriangleright\}, \Gamma, \delta_{abd}, q_0^{\text{abd}}, q_h)$

(where $M_{abd}$ is given by Proposition 16)

$\mathcal{F}_{\text{abd}} = \{f_q : q \in Q_{abd}\} \cup \{f_b : b \in \Gamma\} \cup \{f_\triangleleft\}$

$\mathcal{C}_{\text{abd}} = \{\mathtt{b} : b \in \Gamma\} \cup \{\mathtt{c}_q : q \in Q_{\text{abd}}\} \cup \{\triangleright\}$

Note that as $A \cup \{\blacktriangleleft, \blacktriangleright\} \subseteq \Gamma$, we have $f_\blacktriangleleft, f_\blacktriangleright \in \mathcal{F}_{\text{abd}}$, and $\blacktriangleleft, \blacktriangleright \in \mathcal{C}_{\text{abd}}$.

Rules induced by transition rules of $M_{abd}$:

| (L/R)-move | rewrite rules ($q \in Q_{\text{abd}}$, $b \in \Gamma$) |
|---|---|
| $\delta(q, b) \to (q', b', R)$ | $f_q(\mathtt{b}(x)) \to f_{b'}(f_{q'}(x))$ |
| $\delta(q, b) \to (q', \epsilon, R)$ | $f_q(\mathtt{b}(x)) \to f_b(f_{q'}(x))$ |
| $\delta(q, b) \to (q', b', L)$ | $f_q(\mathtt{b}(x)) \to \mathtt{c}_{q'}(\mathtt{b}'(x))$ |
| $\delta(q, b) \to (q', \epsilon, L)$ | $f_q(\mathtt{b}(x)) \to \mathtt{c}_{q'}(\mathtt{b}(x))$ |

Propagation rules:

| rewrite rules ($q \in Q_{\text{abd}}$, $b \in \Gamma$) |
|---|
| $f_{q_h}(x) \to \mathtt{c}_{q_h}(x)$ |
| $f_\triangleleft(\mathtt{c}_{q_h}(x)) \to \triangleright$ |
| $f_b(\mathtt{c}_q(x)) \to f_q(\mathtt{b}(x))$ |

**Figure 3** Non-length-increasing constructor TRS defined from an LBA $M_{\text{abd}}$. Observe that $f_0 \notin \mathcal{F}_{\text{abd}}$ and that the constructor TRS will not accept any strings on its own. .

Due to our convention that constructor TRSs must start their computations on terms on the form $f_0(\widetilde{\alpha})$, we encounter the problem that non-length-increasingness prevents us from setting up the simulation of the LBA tape and state: We would need a rule of the form $f_0(x) \to f_\triangleleft(f_{q_0}(\cdots))$. The problem is solved by the following proposition:

▶ **Proposition 16.** *Let LBA $M$ accept the language $L \subseteq A^+$ and let $(a, b, d) \in A^3$. Then there exists an LBA $M_{abd}$ with input and tape alphabets identical to those of $M$ that accepts the language $L' = \{\beta \in A^* : abd \cdot \beta \in L\}$.*

**Proof.** If the input to $M$ has size $n \geq 3$, we may encode all possible configurations of the leftmost 3 cells of the tape of $M$ in $|\Gamma|^3$ states. If $M$ has $|Q|$ states, we may construct an LBA $M_{abd}$ with $(4|\Gamma|^3) \times |Q|$ states that encodes any changes to the leftmost 3 cells in its states (the factor 4 is used by $M_{abd}$ to keep track of where the tape head is (either of the first three "cells" encoded by the states, or to their right), and only uses $n - 3$ tape cells (where it simply simulates $M$) . ◀

For each LBA $M$ and $(a, b, d) \in A^3$, we define a non-length-increasing constructor TRS $\Delta_{LBA}^{\text{abd}}(M)$ by the translation given in Figure 3 – effectively the same translation as that in Figure 2, except for the absence of a start rule and the addition of rules for stopper fitting. For each LBA $M$, we define a corresponding non-length-increasing system $\Delta_{LBA}(M)$ by taking the union of all rules from all of the $|\Gamma|^3$ LBAs $M_{\text{abd}}$ and adding rules to start the computation. The resulting constructor TRS is shown in Figure 4.

▶ **Proposition 17.** *If $M$ is an LBA, then $\Delta_{LBA}(M)$ is a non-length-increasing monadic constructor TRS.*

**Proof.** Observe that every rule of $M$ is translated by $\Delta_{LBA}(\cdot)$, whence $\Delta_{LBA}(M)$ is defined for all $M$. Furthermore, every rule of $\Delta_{LBA}(M)$ is non-length-increasing, and the general result follows. ◀

$$\mathcal{F} = \left( \bigcup_{(a,b,d)\in A^3} \mathcal{F}_{\text{abd}} \right) \cup \{f_0\} \quad \mathcal{C} = \bigcup_{(a,b,d)\in A^3} \mathcal{C}_{\text{abd}}$$

The rules of $\Delta_{LBA}(M)$ are the union of $\bigcup_{(a,b,c)\in A^3} R_{\text{abd}}$ with the set of rules below:

Start rules and stopper rules:

| rewrite rules (for each $a, b, d \in A$, not necessarily distinct) | |
|---|---|
| $f_0(\widetilde{a}(\triangleright)) \to \triangleright$ | if $M$ accepts $a$ |
| $f_0(\widetilde{a}(\widetilde{b}(\triangleright))) \to \triangleright$ | if $M$ accepts $ab$ |
| $f_0(\widetilde{a}(\widetilde{b}(\widetilde{d}(x)))) \to f_{\triangleleft}(f_{q_0^{\text{abd}}}(\blacktriangleleft\,(e(x))))$ | |
| $e(\widetilde{a}(x)) \to \widetilde{a}(e(x))$ | |
| $e(\triangleright) \to \blacktriangleright\,(\triangleright)$ | |

**Figure 4** Encoding $\Delta_{LBA}$(M) of an LBA $M$ as a non-length-increasing system.

Again, the following is tedious, but not hard, to prove:

▶ **Lemma 18.** *Let $\alpha \in A^+$. Then LBA $M$ transitions to the halting state on input $\blacktriangleleft\,\alpha\,\blacktriangleright$ iff $f_0(\widetilde{\alpha}) \to^*_{\Delta_{LBA}(M)} \triangleright$.*

We can now prove the main result of the section:

▶ **Theorem 19.** *Let $L \subseteq A^+$. The following are equivalent: (i) $L$ is context-sensitive, (ii) $L$ is accepted by a monadic non-length-increasing constructor TRS.*

**Proof.** If $L$ is context-sensitive, it is accepted by an LBA $M$ by Theorem 15. Then, Lemma 18 furnishes that $\Delta_{LBA}(M)$ accepts $L$, and by Proposition 17, $\Delta_{LBA}(M)$ is a monadic non-length-increasing constructor TRS. Conversely, if $L$ is accepted by a monadic non-length-increasing constructor TRS $R$ over alphabet $\Sigma$, we can define a non-deterministic Turing machine with tape alphabet $\Sigma$ that runs in linear space and accepts $L$: In every rule $l \to r$, both $l$ and $r$ are terms over unary and nullary symbols, hence essentially strings. As $|l| \geq |r|$, a rewrite step corresponds to replacing a substring by a substring of at most the same size. Thus, we may simply encode the rules of $R$ in the states $M$. The current state of the term $f_1(f_2(\cdots f_m(b)))$ is encoded in $m+1$ symbols $f_1 f_2 \cdots f_m b$ on the Turing machine's tape, and application of a rule is simply done by replacing the symbols on the relevant tape cells. Choosing what rule to apply and where to apply it is selected non-deterministically by $M$. As $|l| \geq |r|$, the number of tape cells used will never increase, whence the machine runs in linear space, and Theorem 15 furnishes that $L$ is context-sensitive. ◀

## 5 Context-Free Languages: (Strongly) cons-free systems

We now treat context-free languages; we first need their corresponding notion of accepting machine.

▶ **Definition 20.** *A* pushdown automaton *(PDA) is a tuple $(Q, A, \Gamma, \delta, q_0, Z_0)$ where $Q$ is a finite set of* states*, $A$ is a finite set of* input symbols*, $\Gamma$ is a finite* stack alphabet*, $q_0 \in Q$ is the* start state*, $Z_0 \in \Gamma$ is the* start stack symbol*, and $\delta$ is a relation consisting of a finite number of transition rules of the form $\delta(q, a, X) \to (p, \gamma)$ where $q \in Q$, $a \in A \cup \{\epsilon\}$, $X \in \Gamma$, $p \in Q$, and $\gamma \in \Gamma^*$.*

The definition of PDA above has no final states, and will thus accept by empty stack (and empty input), as is common in the literature [26]. We make the convention that the bottom of the stack is written to the left and the top to the right; hence, symbols are pushed and popped to the right.

As we shall only consider one-state PDAs in this paper; the below definition of acceptance has been specialized to that case (for the general case, see any standard textbook, e.g. [26]):

▶ **Definition 21.** *A one-state PDA is said to* accept *input* $\alpha \in A^+$ *if* $\alpha = a_1 \cdots a_m$ *where each* $a_i \in A \cup \{\epsilon\}$ *and there is a sequence of strings* $s_1, \ldots, s_m$ *from* $\Gamma^*$ *such that: (i)* $s_0 = Z_0$*, (ii) for* $i = 0, \ldots, m-1$*, there is a rule* $\delta(a_{i+1}, Z) \to Z'$ *where* $s_i = tZ$ *and* $s_{i+1} = tZ'$ *for some* $Z, Z' \in \Gamma \cup \{\epsilon\}$ *with* $Z \neq \epsilon$*, and* $t \in \Gamma^*$ *(that is, the PDA moves according to the stack and next input symbol)*[1]*, (iii)* $a_m = \epsilon$ *and* $s_m = \epsilon$ *(that is, empty input and empty stack are reached at the end). Otherwise, the PDA is said to* reject *the input.*

The following proposition is standard; see for example [13] for a proof.

▶ **Proposition 22.** *If* $L \subseteq A^*$ *is accepted by a PDA, it is accepted by a one-state PDA* $(\{q_0\}, A, \Gamma', \delta, q_0, Z_0)$ *(where we assume acceptance by empty stack).*

The following theorem is standard (see e.g. [26, Thm. 2.12])

▶ **Theorem 23.** *A language* $L \subseteq A^+$ *is context-free iff it is accepted by a PDA with input alphabet* $A$*.*

By Proposition 22 and Theorem 23, a language is thus context-free iff it is accepted by a one-state PDA.

As with the language classes RE and CSL, we shall prove that a particular class of monadic rewrite systems corresponds to CFL; this class consists of the (strongly) cons-free systems:

▶ **Definition 24.** *A constructor TRS is said to be (strongly)* cons-free *if, for every rule* $l \to r$ *there are no constructor symbols in* $r$*.*

▶ Remark 25. Cons-freeness has been used for multiple characterizations of complexity classes (see, e.g., [18, 5, 19, 8]). The gist is that, during rewriting, no new constructor terms can be built; thus, the definition of cons-freeness is usually less restrictive than the *strong* cons-freeness of Definition 24 [19, 8][2], but we believe that the restriction to the very simple notion of strong cons-freeness is cleaner and simpler to work with here.

▶ Remark 26. As pointed out by a referee, there are likely simpler grammar-based proofs that the class of strongly cons-free constructor TRSs characterizes CFL compared to the one we give using PDAs. However, the proof via PDAs shed light on the intuition that rewriting in monadic constructor TRSs essentially consist of manipulation of up to two stacks – and that for (strongly) cons-free systems, the manipulation is essentially a single "general" stack and a "restricted" stack that can only be decremented, exactly as in a PDA.

The following proposition shows that we may disregard nullary defined symbols in the remainder of the paper:

[1] As the PDA has only a single state, we have suppressed the state in the notation of the rule $\delta(a_{i+1}, Z) \to Z'$.

[2] For example, cons-freeness of a rule $l \to r$ in [8] is defined as the requirement that every subterm of the form $\mathsf{c}(s)$ in $r$ (where $\mathsf{c} \in \mathcal{C}$) either occurs in $l$, or is a ground constructor term.

$M = (\{q_0\}, A, \Gamma, \delta, q_0, Z_0)$

$\mathcal{F} = \{f_Z : Z \in \Gamma\} \cup \{f_0\}$ and $\mathcal{C} = \{\widetilde{a} : a \in A\} \cup \{\triangleright\}$

Rewrite rules induced by transition rules in $\delta$:

| transition rule in $\delta$ | rule of $R_m$ |
|---|---|
| $\delta(a, Z) \to Z_1 \cdots Z_m$ | $f_Z(\widetilde{a}(x)) \to f_{Z_1}(\cdots f_{Z_m}(x))$ |
| $\delta(a, Z) \to \epsilon$ | $f_Z(\widetilde{a}(x)) \to x$ |
| $\delta(\epsilon, Z) \to Z_1 \cdots Z_m$ | $f_Z(x) \to f_{Z_1}(\cdots f_{Z_m}(x))$ |
| $\delta(\epsilon, Z) \to \epsilon$ | $f_Z(x) \to x$ |

Start rule:

| rewrite rules |
|---|
| $f_0(x) \to f_{Z_0}(x)$ |

**Figure 5** Rules of the cons-free system $R_M$ induced by the PDA $M$. As $M$ has only one state, the state argument has been omitted from $\delta$.

▶ **Proposition 27.** *Let $R$ be a cons-free, monadic constructor TRS that accepts $L \subseteq A^+$, and let $R'$ be the monadic, cons-free constructor TRS obtained by omitting all rules in $R$ that contain a nullary defined symbol. Then $R'$ accepts $L$.*

To greatly simplify our proofs for context-free and regular languages, we introduce *normal* systems:

▶ **Definition 28.** *A rule $l \to r$ is* normal *if $l$ contains at most one constructor symbol, and that constructor symbol is unary, that is either $l = f(\mathsf{c}(x))$, or $l = f(x)$ (for some $f \in \mathcal{F}$ and $\mathsf{c} \in \mathcal{C}$). A constructor TRS $R$ is* normal *if every rule is normal.*

The following lemma shows that we can transform a set of rules with "large" left-hand sides into (a larger set of) normal rules that accept the same language:

▶ **Lemma 29.** *If $L \subseteq A^+$ is accepted by a monadic, cons-free constructor TRS $R$, then $L$ is accepted by a monadic, cons-free, normal constructor TRS $R'$ with $\mathcal{C} = \widetilde{A} \cup \{\triangleright\}$. If $R$ is tail recursive, then $R'$ may be chosen to be tail recursive as well.*

For each one-state PDA, we define a cons-free constructor TRS $R_M$ as given in Figure 5.

In Figure 5, the presence of transition rules of the form $\delta(\epsilon, Z) \to r$ force us to let $R_M$ contain rules of the form $f(x) \to r'$. By the definition of TRSs, application of such a rule may occur anywhere in a term. However, as we want to simulate the PDA stack by a string of defined symbols, applying a rule $f(x) \to r'$ corresponds to removing a symbol in the middle of the stack rather than popping it off the top. Hence, we are forced to require that redexes in $R_M$ are contracted only at places corresponding to the top of the stack – which is the case if the redexes are *innermost*. This is also sufficient, as we shall see shortly.

▶ **Definition 30.** *Let $p$ be a non-negative integer. A ground term $s$ has a* border position at $p$ *if $s = f_1(\cdots(f_p(t))\cdots)$ where $p \geq 1$, $f_1, \ldots, f_p \in \mathcal{F}$ and $t$ is a ground constructor term.*

The following proposition is proved by induction on the length of the involved rewrite sequence:

▶ **Proposition 31.** *Let $R$ be a monadic, cons-free constructor TRS. If $t$ is a ground term with a border position such that $t \longrightarrow^* \triangleright$, then every term in the rewrite sequence, except the last, has a border position, and an innermost redex at the border position.*

Even if $R$ contains overlapping redexes, innermost rewite steps can be retracted across non-innermost ones (and efficiently so, as monadic systems cannot make more than a single copy of each subterm):

▶ **Proposition 32.** *Let $R$ be a monadic, cons-free constructor TRS, let $s$ be a term containing a redex at a border position, and let $m \geq 0$. If $s \to^k t'$ by non-innermost steps, and $t' \to_{IM} t$, then there is a term $s'$ such that $s \to_{IM} s' \to^k t$, where $\to_{IM}$ is innermost reduction.*

**Proof.** As $R$ is a constructor TRS, every redex at a border position is innermost, whence $s$ contains an innermost redex. As every non-innermost redex cannot overlap an innermost redex, all innermost redexes in $s$ are preserved across any non-innermost reduction, and remain innermost. Consider the redex $u$ contracted in the step $t' \to t$; as the innermost redex at border position in $s$ is preserved across $s \to^k t'$, it overlaps with $u$. But as the left-hand sides of all redexes in $R$ are of the form $f(w)$ where $w$ is a constructor term, no redexes created in the reduction $s \to^k t'$ can overlap with the descendants of redexes at innermost position in $s$. Hence, $u$ is the descendant of an innermost redex $u'$ in $s$. Furthermore, contracting an innermost redex cannot destroy any redexes except those that overlap with it (and are thus, by definition, also innermost), and thus we may contract $u'$ to obtain the step $s \to_{\mathrm{IM}} s'$, followed by mimicking the steps in $s \to^k t'$ starting from the term $s'$ (all of which can be performed, as $u'$ does not overlap with any non-innermost redex). Thus, $s' \to^k t$, concluding the proof. ◀

▶ **Lemma 33.** *Let $R$ be a monadic, cons-free, normal constructor TRS. If $s = f_0(\tilde{\alpha}) \to^* \rhd$, then $s \longrightarrow^*_{IM} \rhd$.*

**Proof.** By Proposition 31, every term in $s \longrightarrow^* \rhd$, except the last, contains an innermost redex at a border position. Divide $s \longrightarrow^* \rhd$ into subsequences, each of the form $s' \to^*_{\mathrm{IM}} s'' \to^k t' \to^+_{\mathrm{IM}} t''$ where $s'' \to^k t'$ consists solely of non-innermost steps for some $k \geq 1$. Observe that this is always possible because the last step of $s \longrightarrow^* \rhd$ must be innermost as $R$ is cons-free and $\rhd$ is a constructor. By repeated application of Proposition 32, we obtain $s' \to^+_{\mathrm{IM}} s''' \to^k t''$ for some term $s'''$. Hence, a straightforward induction on the length of $s \longrightarrow^* \rhd$ shows that all innermost steps can be retracted across non-innermost steps, resulting in a reduction $s \to^*_{\mathrm{IM}} t''' \to^* \rhd$ of length no more than the original where $t''' \to^* \rhd$ contains no innermost steps. But as the last step of any reduction $s \to^* \rhd$ must be innermost, the length of $t''' \to^* \rhd$ is zero, and thus $s \to^*_{\mathrm{IM}} \rhd$, as desired. ◀

As with our previous simulation results, the following result is tedious to prove, but not difficult:

▶ **Lemma 34.** *Let $M$ be a one-state PDA accepting language $L \subseteq A^+$. Then $R_M$ accepts $L$ by innermost evaluation.*

We now show how to simulate any cons-free constructor TRS by a one-state PDA. We consider only normal systems, as this suffices by Lemma 29. For any normal, monadic, cons-free constructor TRS with $\mathcal{C} = \widetilde{A} \cup \{\rhd\}$, we define a one-state PDA as shown in Figure 6.

Again, the following is tedious, but fairly straightforward:

▶ **Lemma 35.** *Let $L \subseteq A^+$ be accepted by innermost reduction by a normal, cons-free, monadic constructor TRS $R$. Then $\mathrm{PDA}_R$ accepts $L$.*

We thus have:

▶ **Theorem 36.** *The following are equivalent for a language $L \subseteq A^+$: (i) $L$ is context-free, (ii) $L$ is accepted by a monadic cons-free constructor TRS.*

$\mathrm{PDA}_R = (\{q_0\}, A, \{Z_f : f \in \mathcal{F}\}, \delta, q_0, Z_{f_0})$

| rule of $R$ | transition rule in $\delta$ |
|---|---|
| $f(\mathsf{c}(x)) \to f_1(\cdots f_m(x))$ | $\delta(\mathsf{c}, Z_f) \to Z_{f_1} \cdots Z_{f_m}$ |
| $f(x) \to f_1(\cdots f_m(x))$ | $\delta(\epsilon, Z_f) \to Z_{f_1} \cdots Z_{f_m}$ |
| $f(\mathsf{c}(x)) \to x$ | $\delta(\mathsf{c}, Z_f) \to \epsilon$ |
| $f(x) \to x$ | $\delta(\epsilon, Z_f) \to \epsilon$ |

**Figure 6** Definition of the pushdown automaton $\mathrm{PDA}_R$ from a normal, monadic, cons-free constructor TRS $R$ with signature $\mathcal{F} \cup \mathcal{C} = \mathcal{F} \cup (\tilde{A} \cup \{\triangleright\})$.

**Proof.** If $L$ is context-free, Theorem 23 yields that $L$ is accepted by a PDA $M$, which we may assume by Proposition 22, has exactly one state. Lemma 34 yields that $R_M$ accepts $L$ by innermost reduction, and Lemma 33 shows that the elements of $A^*$ accepted by $R_M$ are exactly those accepted by innermost reduction. Clearly, $R_M$ is a monadic, cons-free constructor TRS.

Conversely, if $L$ is accepted by a monadic, cons-free constructor TRS $R$, Lemma 33 yields that $R$ accepts $L$ by innermost reduction, and by Lemma 29 we may assume wlog. that $R$ is normal. Lemma 35 now shows that $\mathrm{PDA}_R$ accepts $L$, whence $L$ is context-free by Theorem 23. ◀

## 6 Regular languages: tail recursive cons-free systems

We shall now consider the class of *regular* languages. We assume the reader to be familiar with the fact that a language is regular iff it is accepted by an NFA iff it is accepted by a DFA. To fix notation, we give the following definition:

▶ **Definition 37.** *A* non-deterministic finite automaton *(NFA) is a tuple* $(Q, A, \delta, q_0, Q_{\mathrm{accept}})$ *such that* $Q$ *is a non-empty set of* states*,* $A$ *is the* input alphabet*,* $\delta$ *is a set of transition rules on one of the forms* $\delta(q, a) \to q'$ *or* $\delta(q, \epsilon) \to q'$ *where* $q, q' \in Q$ *and* $a \in A$*,* $q_0 \in Q$ *is the start state, and* $Q_{\mathrm{accept}} \subseteq Q$ *is the set of accept states. Furthermore, for any* $q \in Q$ *and any* $a \in A$*, there is at least one transition of the form* $\delta(q, a) \to q'$*.* A deterministic finite automaton *(DFA) is an NFA such that there are* no *transitions of the form* $\delta(q, \epsilon) \to q'$*, and if there is a transition of the form* $\delta(q, a) \to q'$*, then there is no transition* $\delta(q, a) \to q''$ *with* $q' \neq q''$*.*

The class REG is characterized by the monadic constructor TRSs that are both cons-free and *one-call* (see Definition 40).

In tail-recursive functional programming, the height of the call stack is bounded above by a constant; a similar result holds here for *innermost* reduction:

▶ **Proposition 38.** *Let* $R$ *be a monadic, normal, cons-free, tail-recursive constructor TRS. Then there is a constant* $c$ *such that for any* $\alpha \in A^*$ *and any innermost reduction* $f_0(\tilde{\alpha}) \to^*_{IM} \triangleright$*, the number of defined symbols in any term of the reduction is at most* $c$*.*

**Proof.** By Proposition 31, any term in the reduction $f_0(\tilde{\alpha}) \to^*_{\mathrm{IM}} \triangleright$ contains an innermost redex at border position. Hence, the position of any rewrite step in a term $t$ in the reduction will occur at the *rightmost* element of $\mathcal{F}$ in $t$. Thus, redex contraction in innermost reduction will always occur at the rightmost element of $\mathcal{F}$ in $t$. Let $f \in \mathcal{F}$ be such an element, and let $f(\mathsf{c}(x)) \to f_1(\cdots f_m(x))$ be the rule of a redex at that position (if there is no variable in the

right-hand side of the rule, the supposition that $R$ is cons-free entails that no future steps will be able to produce $\triangleright$, a contradiction). As $R$ is tail recursive, we have $f > f_2, \ldots, f_m$, and $f \geq f_1$.

Let $l$ be the maximum number of occurrences of symbols from $\mathcal{F}$ in any right-hand side among rules of $R$. Any totally ordered chain $f_1 > f_2 > \cdots > f_m$ in $\mathcal{F}$ has length at most $|\mathcal{F}|$, and thus, the maximal number of defined symbols in any term in $f_0(\tilde{\alpha}) \to_{\mathrm{IM}}^* \triangleright$ is at most $c \triangleq 1 + l \cdot |\mathcal{F}|$. ◀

▶ **Example 39.** The assumption that $f_0(\tilde{\alpha}) \to_{\mathrm{IM}}^* \triangleright$ in Proposition 38 cannot be omitted (that is, the presence of $\triangleright$ as the final term is crucial). Consider the following constructor TRS:

$$R = \left\{ \begin{array}{rcll} f_0(x) & \to & f_0(g(x)) & \\ f_0(x) & \to & f_0(h(x)) & \\ f_0(x) & \to & g(x) & \\ g(\tilde{a}(x)) & \to & x & \text{for all } a \in A \end{array} \right\}$$

Observe that $R$ is tail recursive and accepts $A^+$ (because $f_0(\tilde{\alpha}) \to^* f_0(g^{|\alpha|-1}(\tilde{\alpha})) \to g^{|\alpha|}(\tilde{\alpha}) \to^* \triangleright$. But the number of elements of $\mathcal{F}$ in terms occurring in reductions starting from $f_0(\tilde{\alpha})$ is unbounded, as witnessed by $f_0(\tilde{\alpha}) \to f_0(g(\tilde{\alpha})) \to \cdots$ and $f_0(\tilde{\alpha}) \to f_0(h(\tilde{\alpha})) \to f_0(h(h(\tilde{\alpha}))) \to \cdots$; in particular, the latter reduction shows that there are infinite reductions with an innermost redex at the *root* of every term, and where the number of elements of $\mathcal{F}$ in the terms has no upper bound.

We now define *one-call* systems:

▶ **Definition 40.** *A monadic constructor TRS is said to be* one-call *if, for every rule $l \to r$, the right-hand side $r$ contains at most one element of $\mathcal{F}$.*

The following lemma shows that instead of tail recursion, we could instead have considered one-call systems:

▶ **Lemma 41.** *Let $R$ be a monadic, cons-free, tail-recursive constructor TRS accepting language $L \subseteq A^+$. Then, there is a one-call, normal, monadic, cons-free constructor TRS that accepts $L$.*

**Proof.** By Lemma 29, we may assume wlog. that $R$ is normal. By Lemma 33, for every $\alpha \in A^+$, if $f_0(\tilde{\alpha}) \to^* \triangleright$, then $f_0(\tilde{\alpha}) \to_{\mathrm{IM}}^* \triangleright$. By Proposition 38, there is a constant $c$ such that for every $\alpha \in A^+$, for every reduction of the form $f_0(\tilde{\alpha}) \to_{\mathrm{IM}}^* \triangleright$, the number of elements of $\mathcal{F}$ in any term of the reduction is at most $c$.

We now construct a one-call (and normal, monadic, cons-free) constructor TRS $R'$ that accepts $L$. $R'$ will have a new set of defined symbols $\mathcal{F}'$ and use the same set of constructors $\mathcal{C}$ as $R$. For every integer $k$ with $0 < k \leq c$ and every $(f_1, \ldots, f_k) \in \mathcal{F}^k$, create a defined symbol $g_{f_1 \cdots f_k} \in \mathcal{F}'$. As $R$ is normal and cons-free, every rule of $R$ is on one of the forms $f(\mathsf{c}(x)) \to r$ or $f(x) \to r$. For each symbol $g_{f_1 \cdots f_k} \in \mathcal{F}'$, and each rule $l \to r$ of $R$ such that the root symbol of $l$ is $f_k$, create a rule of $R'$ as follows:

- $g_{f_1 \cdots f_k}(\mathsf{c}(s)) \to g_{f_1 \cdots f_{k-1} h_1 \cdots h_m}(s)$ if $l \to r = f_k(\mathsf{c}(x)) \to h_1(\cdots h_m(s))$ (where $s = x$ or $s \in \mathcal{F}$).
- $g_{f_1 \cdots f_k}(x) \to g_{f_1 \cdots f_{k-1} h_1 \cdots h_m}(s)$ if $l \to r = f_k(x) \to h_1(\cdots h_m(s))$ (where $s = x$ or $s \in \mathcal{F}$).

Define $S$ to be the resulting TRS. By construction, $S$ is one-call, monadic, and cons-free.

$\mathrm{NFA}_R = (Q, A, \delta, \{q_{f_0}\}, \{q_h\})$ where $Q = \{q_f : f \in \mathcal{F}\} \cup \{q_h\}$

Transition rules in $\delta$:

| rule(s) of $R$ | transition rule in $\delta$ |
|---|---|
| $f(\mathsf{c}(x)) \to g(x)$ | $\delta(q_f, c) \to q_g$ |
| $f(x) \to g(x)$ | $\delta(q_f, \epsilon) \to q_g$ |
| $f(\mathsf{c}(x)) \to x$ | $\delta(q_f, c) \to q_h$ |
| $f(x) \to x$, | $\delta(q_f, \epsilon) \to q_h$ |

**Figure 7** The NFA–$\mathrm{NFA}_R$–defined from a normal, monadic cons-free, one-call constructor TRS $R$.

We claim that, for each $\alpha \in A^+$, we have $f_0(\tilde{\alpha}) \to_R^* \rhd$ iff $f_0(\tilde{\alpha}) \to_S^* \rhd$.

If $f_0(\tilde{\alpha}) \to_{\mathrm{IM}}^* \rhd$, write the reduction as $t_0 = f_0(\tilde{\alpha}) \to_{\mathrm{IM}} t_1 \to_{\mathrm{IM}} \to \cdots \to \rhd = t_n$. Observe that each term $t_i = f_1(\cdots f_k(\mathsf{c}))$ (where $\mathsf{c}$ is a constructor term) in the reduction can be mimicked in $S$ by a term of the form $g_{f_1 \cdots f_k}(\mathsf{c})$.

By Proposition 31, every term of $f_0(\tilde{\alpha}) \to_{R,\mathrm{IM}}^* \rhd$, except the last, contains at least one innermost redex at border position, and hence, the step $t_i \to t_{i+1}$ must be $f_1(\cdots f_k(\mathsf{c})) \to f_1(\cdots f_{k-1}(h_1(\cdots h_m(\cdots))))$ using some rule $f_k(\mathsf{c}(x)) \to h_1(\cdots h_m(s))$ or $f_k(x) \to h_1(\cdots h_m(s))$ (for some $m \geq 0$). Hence, the step can clearly be mimicked by application of a rule in $S$, and we have $f_0(\tilde{\alpha}) \to_S^* \rhd$.

Conversely, if $f_0(\tilde{\alpha}) \to_S^* \rhd$, by construction of $S$, every term in the reduction is of the form $g_{f_1 \cdots f_k}(\mathsf{c})$ for some constructor term $\mathsf{c}$. For each such term, there is a step $g_{f_1 \cdots f_k}(\mathsf{c}) \to g_{f_1 \cdots f_{k-1} h_1 \cdots h_m}(s'\{x \mapsto \mathsf{c}\})$ iff there is a rule $f_k(s) \to h_1(\cdots h_m(s'))$ in $R$, and hence $f_1(\cdots f_k(\mathsf{c})) \to_R f_1(\cdots f_{k-1}(h_1(\cdots h_m(s'\{x \mapsto \mathsf{c}\}))))$.

Thus, every step of $f_0(\tilde{\alpha}) \to_S^* \rhd$ can be mimicked by an innermost step in $R$, whence $f_0(\tilde{\alpha}) \to_R^* \rhd$, as desired. Hence, $S$ accepts $L$, and by construction, $S$ is normal, monadic, and one-call. ◀

▶ **Lemma 42.** *Let $R$ be a normal, monadic, cons-free, one-call constructor TRS deciding language $L \subseteq A^+$. Then, the NFA $\mathrm{NFA}_R$ (see Fig. 7) accepts $L$.*

**Proof.** Recall from basic automata theory that we may wlog. assume that an NFA only accepts if it is in an accepting state when all of its input has been consumed. Denote by $L(\mathrm{NFA}_R)$ the language accepted by $\mathrm{NFA}_R$. By construction of $\mathrm{NFA}_R$, any run of $\mathrm{NFA}_R$ clearly mimicks reductions of $R$: every rewrite step is mimicked by exactly one transition in $\mathrm{NFA}_R$, and conversely, any transition in $\mathrm{NFA}_R$ can be mimicked by a rewrite step in $R$. If $f_0(\tilde{\alpha}) \to^* \rhd$, there is in particular a run of $\mathrm{NFA}_R$ ending in $q_h$ with the entire input $\alpha$ having been consumed in the run, and hence $L \subseteq L(\mathrm{NFA}_R)$. Conversely, if $\alpha \in L(\mathrm{NFA}_R$, there is a run of $\mathrm{NFA}_R$ on input $\alpha$ that (i) consumes all the input, and (ii) ends in $q_h$, and hence there is a rewrite sequence starting from $f_0(\tilde{\alpha})$ that ends with one of the two rewrite steps $f(\mathsf{c}(\rhd)) \to \rhd$ or $f(\rhd) \to \rhd$, whence $L(\mathrm{NFA}_R) \subseteq L$. ◀

By the equivalence of DFAs and NFAs, it suffices to simulate DFAs by rewriting systems. In Figure 8 we show how to obtain such a system.

▶ **Lemma 43.** *If $M = (Q, A, \delta, Q_{\mathrm{accept}})$ is a DFA accepting language $L \subseteq A^+$, then $R_M^{\mathrm{DFA}}$ (see Fig. 8) accepts $L$.*

$M = (Q, A, \delta, q_0, Q_{\text{accept}})$ $\quad \mathcal{F} = \{f_q : q \in Q\}$ $\quad \mathcal{C} = \widetilde{A} \cup \{\triangleright\}$

Rules:

| transition in $\delta$ | rewrite rule |
|---|---|
| $\delta(q, a) \to q'$ | $f_q(\tilde{a}(x)) \to f_{q'}(x)$ |

Accepting run:

| Rule (for every $q \in Q_{\text{accept}}$) |
|---|
| $f_q(x) \to x$ |

(recall that DFAs do not have $\epsilon$-transitions)

**Figure 8** Monadic cons-free, tail recursive constructor TRS $R_M^{\text{DFA}}$ induced by a DFA $M$.

**Proof.** As the DFA is deterministic, there are no $\epsilon$-transitions, and for every $(q, a) \in Q \times A$, there is at most one transition $\delta(q, a) \to q'$. Thus, the constructor TRS $R_M^{\text{DFA}}$ is monadic, cons-free and one-call. Furthermore, if $q_0$ is the start state, set $f_0 = f_{q_0}$. We claim that for any $\alpha \in A^+$, we have $f_0(\widetilde{\alpha}) \to^* \triangleright$ iff there is an accepting run of the automaton on input $\alpha$ starting in $q_0$. To see this, note that there is a transition on string $b_1 b_2 \cdots b_k$ from state $q$ to state $q' \notin Q_{\text{accept}}$ iff there is a rule $\delta(q, b_1) \to q'$ iff $f_q(\widetilde{b}_1(\widetilde{b}_2 \cdots \widetilde{b}_k(\triangleright))) \to f_{q'}(\widetilde{b}_2 \cdots \widetilde{b}_k(\triangleright)))$. Thus, $M$ reaches an accepting state after emptying the input iff $f_0(\widetilde{\alpha}) \to^* f_q(\triangleright)$ where $q \in Q_{\text{accept}}$; and $f_q(\triangleright) \to \triangleright$ iff $q \in Q_{\text{accept}}$. Hence, the DFA accepts string $\alpha$ iff the above system accepts string $\alpha$, and the result follows. ◀

We thus have the final result of the paper:

▶ **Theorem 44.** *The following are equivalent for a language $L \subseteq A^+$: (i) $L$ is regular, (ii) $L$ is accepted by a one-call, monadic, cons-free constructor TRS, (iii) $L$ is accepted by a tail recursive, monadic, cons-free constructor TRS.*

**Proof.** If $L$ is regular, it is accepted by a DFA, hence by Lemma 43 accepted by a monadic, cons-free, one-call constructor TRS. Conversely, if $L$ is accepted by a monadic cons-free, one-call constructor TRS, Lemma 29 shows that we may wlog. assume that $R$ is normal and one-call. Lemma 42 then shows that there is an NFA accepting $L$, whence $L$ is regular. Finally, observe that a one-call TRS is always tail-recursive (by relating all defined symbols in the weak component of the ordering), and that Lemma 41 shows that any language accepted by a tail-recursive monadic, cons-free constructor TRS is also accepted by a one-call monadic, cons-free constructor TRS. ◀

## 7 Conclusion and future work

While we have characterized the original 4 language classes in the Chomsky hierarchy, it is clear that similar characterizations *should* exist for other classes, e.g., the visibly pushdown languages [1], or for deterministic context-free languages (where it is natural to conjecture that *non-overlapping* (strongly) cons-free constructor TRSs suffice). However, the proofs of the correspondences asserted in this paper followed from intuition about the (set of) *stacks* maintained by the restricted computational models traditionally used to characterize the classes; it is unclear whether this intuition can be used for more esoteric classes of languages.

On a different note, while the restriction to monadic systems plays well with the Chomsky hierarchy, it seems to be less amenable to characterizations of the usual complexity classes of interest in implicit complexity theory, e.g. PTIME, and it would be interesting to find *natural* constraints on monadic systems that allowed characterization of these classes in a liberal rewriting setting (i.e., no typing beyond what is strictly necessary, and with no restrictions on the evaluation order).

Finally, it should be investigated whether *strong* cons-freeness can be relaxed to more lenient versions of cons-freeness, but for the reasons noted in the paper, this may not give as short and clean a characterization as for strongly cons-free systems.

## References


1 R. Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.

2 M. Avanzini, N. Eguchi, and G. Moser. A path order for rewrite systems that compute exponential time functions. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011*, volume 10 of *LIPIcs*, pages 123–138. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

3 M. Avanzini, G. Moser, and A. Schnabl. Automated implicit computational complexity analysis. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR '08)*, volume 5195 of *Lecture Notes in Computer Science*, pages 132–138. Springer-Verlag, 2008.

4 S. Bellantoni and S.A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

5 Guillaume Bonfante. Some programming languages for logspace and ptime. In Michael Johnson and Varmo Vene, editors, *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Kuressaare, Estonia, July 5-8, 2006, Proceedings*, volume 4019 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2006.

6 A.-C. Caron. Linear bounded automata and rewrite systems: Influence of initial configurations on decision properties. In Samson Abramsky and T. S. E. Maibaum, editors, *TAPSOFT, Vol.1*, volume 493 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 1991. doi:10.1007/3-540-53982-4_5.

7 N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

8 L. Czajka. Term rewriting characterisation of LOGSPACE for finite and infinite data. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

9 D. de Carvalho and J. G. Simonsen. An implicit characterization of the polynomial-time decidable sets by cons-free rewriting. In G. Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014*, volume 8560 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2014.

10 E. P. Friedman. Equivalence problems for deterministic context-free languages and monadic recursion schemes. *Journal of Computer and Systems Sciences*, 14(3):344–359, 1977.

11 E. P. Friedman. Simple context-free languages and free monadic recursion schemes. *Mathematical Systems Theory*, 11(1):9–28, 1977.

12 E. P. Friedman and Sheila A. Greibach. Monadic recursion schemes: The effect of constants. *Journal of Computer and Systems Sciences*, 18(3):254–266, 1979.

13 J. Goldstine, J. K. Price, and D. Wotschke. On reducing the number of states in a pda. *Mathematical Systems Theory*, 15(4):315–321, 1982.

14 S. A. Greibach. *Theory of Program Structures: Schemes, Semantics, Verification*, volume 36 of *Lecture Notes in Computer Science*. Springer-Verlag, 1975.

15 J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education International Inc., 2 edition, 2003.

16 G. Huet and D.S. Lankford. On the uniform halting problem for term rewriting systems. Rapport Laboria 283, IRIA, 1978.

17 N. D. Jones. *Computability and Complexity from a Programming Perspective*. The MIT Press, 1997.

**18** N.D. Jones. The expressive power of higher-order types, or: Life without cons. *Journal of Functional Programming*, 11(1):55–94, 2001.

**19** C. Kop and J. G. Simonsen. Complexity hierarchies and higher-order cons-free term rewriting. *Log. Methods Comput. Sci.*, 13(3), 2017.

**20** S. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, 1964.

**21** H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

**22** J.-Y. Marion. Analysing the implicit complexity of programs. *Information and Computation*, 183(1):2–18, 2003.

**23** M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *The Annals of Mathematics*, 74(3):437–455, 1961.

**24** C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

**25** E. Rich. *Automata, Computability and Complexity: Theory and Applications*. Pearson Prentice Hall, 2008.

**26** M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition, 2006.

**27** Terese, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

**28** R. Thiemann, H. Zantema, J. Giesl, and P. Schneider-Kamp. Adding constants to string rewriting. *Appl. Algebra Eng. Commun. Comput.*, 19(1):27–38, 2008. doi:10.1007/s00200-008-0060-6.