Partial Derivatives for Context-Free Languages
From μ-Regular Expressions to Pushdown Automata

Peter Thiemann(✉)

University of Freiburg, Freiburg, Germany
thiemann@informatik.uni-freiburg.de

Abstract. We extend Antimirov's partial derivatives from regular expressions to μ-regular expressions that describe context-free languages. We prove the correctness of partial derivatives as well as the finiteness of the set of iterated partial derivatives. The latter are used as pushdown symbols in our construction of a nondeterministic pushdown automaton, which generalizes Antimirov's NFA construction.

Keywords: Automata and logic · Regular languages · Context-free languages · Pushdown automata · Derivatives

1 Introduction

Brzozowski derivatives [5] and Antimirov's partial derivatives [4] are well-known tools to transform regular expressions to finite automata and to define algorithms for equivalence and containment of regular languages [3,10]. Both automata constructions rely on the finiteness of the set of iterated derivatives. Brzozowski derivatives need to be considered up to similarity (commutativity, associativity, and idempotence for union) to obtain finiteness. Derivatives had quite some impact on the study of algorithms for regular languages on finite words and trees [6,15].

There are many studies of derivative structures for enhancements of regular expressions. While Brzozowski's original work covered extended regular expressions, partial derivatives were originally limited to simple expressions without intersection and complement. It is a significant effort to define partial derivatives for extended regular expressions [6]. Many further operators have been considered, among them shuffle operators [16], multi-tilde-bar expressions [7], expressions with multiplicities [12], approximate regular expressions [9], and many more. There have been a number of approaches to develop general frameworks for derivation: Caron and coworkers [8] abstract over the support for creating derivations, Thiemann [17] develops criteria for derivable language operators.

Recently, there has been practical interest in the study of derivatives and partial derivatives. Owens and coworkers [14] report a functional implementation with some extensions (e.g., character classes) to handle large character sets, which is partially rediscovering work on the FIRE library [18]. Might and coworkers [1,13] push beyond regular languages by implementing parsing for context-free languages using derivatives and demonstrate its feasibility in practice.

J. Esparza and A.S. Murawski (Eds.): FOSSACS 2017, LNCS 10203, pp. 248–264, 2017.
DOI: 10.1007/978-3-662-54458-7_15

Winter and coworkers [19] study context-free languages in a coalgebraic setting. They use a notion of derivative to give three equivalent characterizations of context-free languages by grammars in weak Greibach normal form, behavioral differential equations, and guarded μ-regular expressions.

In this work, we focus on using derivatives for parsing of context-free languages. While Might and coworkers explore algorithmic issues, we investigate the correctness of context-free parsing with derivatives. To this end, we develop the theory of derivatives for μ-regular expressions, which extend regular expressions with a least fixed point operator. Our results are relevant for context-free parsing because μ-regular expressions are equivalent to context-free grammars in generating power. Compared to the work of Winter and coworkers [19], we do not require recursion to be guarded (i.e., we admit left recursion) and we focus on establishing the connection to pushdown automata. Unguarded recursion forces us to consider derivation by ε, which corresponds to an unfolding of a left-recursive μ-expression. Guarded expressions always admit a proper derivation by a symbol.

Our theory is the proper generalization of Antimirov's theory of partial derivatives to μ-regular expressions: our derivative function corresponds *exactly* to the transition function of the nondeterministic pushdown automaton that recognizes the same language. The pendant of Antimirov's finiteness result yields the finiteness of the set of pushdown symbols of this automaton.

2 Preliminaries

We write $\mathbb{N}$ for the set of natural numbers, $\mathbb{B} = \{\mathbf{ff}, \mathbf{tt}\}$ for the set of booleans, and $X \uplus Y$ for the disjoint union of sets X and Y. We consider total maps $m : X \to Y$ as sets of pairs in the usual way, so that $m \subseteq X \times Y$ and $\emptyset$ denotes the empty mapping. For $x_0 \in X$ and $y_0 \in Y$, the *map update* of m is defined as $m[y_0/x_0](x) = y_0$ if $x = x_0$ and $m[y_0/x_0](x) = m(x)$ if $x \neq x_0$.

For conciseness, we fix a finite set of symbols, Σ, as the underlying *alphabet*. We write Σ^* for the set of finite words over Σ, $\varepsilon \in \Sigma^*$ stands for the empty word, and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For $u, v \in \Sigma^*$, we write $|u| \in \mathbb{N}$ for the length of u and $u \cdot v$ (or just uv) for the concatenation of words.

Given languages $U, V, W \subseteq \Sigma^*$, concatenation extends to languages as usual: $U \cdot V = \{u \cdot v \mid u \in U, v \in V\}$. The Kleene closure is defined as the smallest set $U^* \subseteq \Sigma^*$ such that $U^* = \{\varepsilon\} \cup U \cdot U^*$. We write the *left quotient* as $U \backslash W = \{v \mid v \in \Sigma^*, \exists u \in U : uv \in W\}$. For a singleton language $U = \{u\}$, we write $u \backslash W$ for the left quotient.

Definition 1. *A (nondeterministic) finite automaton (NFA) is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *where* Q *is a finite set of states,* Σ *an alphabet,* $\delta \subseteq Q \times \Sigma \times Q$ *the transition relation,* $q_0 \in Q$ *the initial state, and* $F \subseteq Q$ *the set of final states.*

Let $n \in \mathbb{N}$. *A* run *of* $\mathcal{A}$ *on* $w = a_0 \dots a_{n-1} \in \Sigma^*$ *is a sequence* $q_0 \dots q_n \in Q^*$ *such that, for all* $0 \leq i < n$, $(q_i, a_i, q_{i+1}) \in \delta$. *The run is* accepting *if* $q_n \in F$. *The language recognized by* $\mathcal{A}$ *is* $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists\, \text{accepting run of } \mathcal{A} \text{ on } w\}$.

Definition 2. *A (nondeterministic) pushdown automaton (PDA) is a tuple* $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ *where* Q *is a finite set of states,* Σ *the input alphabet,* Γ *the pushdown alphabet (a finite set),* $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ *is the transition relation,* $q_0 \in Q$ *is the initial state,* $Z_0 \in \Gamma$ *is the bottom symbol.*

A configuration of $\mathcal{P}$ *is a tuple* $c \in Q \times \Sigma^* \times \Gamma^*$ *of the current state, the rest of the input, and the current contents of the pushdown.*

The transition relation δ *gives rise to a binary stepping relation* $\vdash$ *on configurations defined by (for all* $q, q' \in Q$*,* $\alpha \in \Sigma \cup \{\varepsilon\}$*,* $Z \in \Gamma$*,* $\gamma, \gamma' \in \Gamma^*$*,* $v \in \Sigma^*$*):*

$$\frac{(q, \alpha, Z, q', \gamma') \in \delta}{(q, \alpha v, Z\gamma) \vdash (q', v, \gamma'\gamma)}$$

The language of the PDA is $\mathcal{L}(\mathcal{P}) = \{v \in \Sigma^* \mid \exists q \in Q : (q_0, v, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$ *where* $\vdash^*$ *is the reflexive transitive closure of* $\vdash$*.*

3 μ-Regular Expressions

Regular expressions can be extended with a least fixed point operator μ to extend their scope to context-free languages [11].

Definition 3. *The set* $\mathbf{R}(\Sigma, X)$ *of* μ*-regular pre-expressions over alphabet* Σ *and set of variables* X *is defined as the smallest set such that*

- $\mathbf{0} \in \mathbf{R}(\Sigma, X)$,
- $\mathbf{1} \in \mathbf{R}(\Sigma, X)$,
- $a \in \Sigma$ *implies* $a \in \mathbf{R}(\Sigma, X)$,
- $r, s \in \mathbf{R}(\Sigma, X)$ *implies* $r \cdot s \in \mathbf{R}(\Sigma, X)$,
- $r, s \in \mathbf{R}(\Sigma, X)$ *implies* $r + s \in \mathbf{R}(\Sigma, X)$,
- $r \in \mathbf{R}(\Sigma, X)$ *implies* $r^* \in \mathbf{R}(\Sigma, X)$,
- $x \in X$ *implies* $x \in \mathbf{R}(\Sigma, X)$,
- $r \in \mathbf{R}(\Sigma, X \cup \{x\})$ *implies* $\mu x.r \in \mathbf{R}(\Sigma, X)$.

The set $\mathbf{R}(\Sigma)$ *of* μ*-regular expressions over* Σ *is defined as* $\mathbf{R}(\Sigma) := \mathbf{R}(\Sigma, \emptyset)$.

As customary, we consider the elements of $\mathbf{R}(\Sigma, X)$ as abstract syntax trees and freely use parentheses to disambiguate. We further assume that $*$ has higher precedence than $\cdot$, which has higher precedence than $+$. The μx-operator binds the recursion variable x with lowest precedence: its scope extends as far to the right as possible. A variable x occurs free if there is no enclosing μx-operator. A *closed* expression has no free variables.

Definition 4. *The language denoted by a* μ*-regular pre-expression is defined inductively by* $\mathcal{L} : \mathbf{R}(\Sigma, X) \times (X \to \wp(\Sigma^*)) \to \wp(\Sigma^*)$*. Let* $\eta \in X \to \wp(\Sigma^*)$ *be a mapping from variables to languages.*

- $\mathcal{L}(\mathbf{0}, \eta) = \{\}$.
- $\mathcal{L}(\mathbf{1}, \eta) = \{\varepsilon\}$.
- $\mathcal{L}(a, \eta) = \{a\}$ *(singleton letter word) for each* $a \in \Sigma$.
- $\mathcal{L}(r \cdot s, \eta) = \mathcal{L}(r, \eta) \cdot \mathcal{L}(s, \eta)$.
- $\mathcal{L}(r + s, \eta) = \mathcal{L}(r, \eta) \cup \mathcal{L}(s, \eta)$.
- $\mathcal{L}(r^*, \eta) = (\mathcal{L}(r, \eta))^*$.
- $\mathcal{L}(x, \eta) = \eta(x)$.
- $\mathcal{L}(\mu x.r, \eta) = \mathsf{lfp}\ L.\mathcal{L}(r, \eta[x \mapsto L])$.

For an expression $r \in \mathbf{R}(\Sigma)$*, we write* $\mathcal{L}(r) := \mathcal{L}(r, \emptyset)$.

Here, lfp is the *least fixed point operator* on the complete lattice $\wp(\Sigma^*)$ (ordered by set inclusion). Its application in the definition yields the smallest set $L \subseteq \Sigma^*$ such that $L = \mathcal{L}(r, \eta[x \mapsto L])$. This fixed point exists by Tarski's theorem because $\mathcal{L}$ is a monotone function, which is captured precisely in the following lemma.

Lemma 5. *For each finite set* X*,* $\eta \in X \to \wp(\Sigma)$*,* $r \in \mathbf{R}(\Sigma, X \cup \{x\})$*, the function* $L \mapsto \mathcal{L}(r, \eta[x \mapsto L])$ *is monotone on* $\wp(\Sigma^*)$*. That is, if* $L \subseteq L'$*, then* $\mathcal{L}(r, \eta[x \mapsto L]) \subseteq \mathcal{L}(r, \eta[x \mapsto L'])$.

According to Leiss [11], it is a folkore theorem that the languages generated by μ-regular expressions are exactly the context-free languages.

Theorem 6. $L \subseteq \Sigma^*$ *is context-free if and only if there exists a* μ*-regular expression* $r \in \mathbf{R}(\Sigma)$ *such that* $L = \mathcal{L}(r)$.

Subsequently we will deal syntactically with fixed points. To this end, we define properties of expressions and substitutions to make substitution application well-defined.

Definition 7. *Let* $\mathcal{X}$ *be the universe of variables occurring in expressions equipped with a strict partial order* $\prec$.

An expression is order-respecting *if each subexpression of the form* $\mu x.r$ *has only free variables which are strictly before* x*:* $\forall y \in fv(\mu x.r), y \prec x$.

A mapping $\sigma : X \to \mathbf{R}(\Sigma, X)$ *is* order-closed *if* $\forall x \in X$*,* $\sigma(x)$ *is order-respecting and* $\forall y \in fv(\sigma(x))$*,* $y \prec x$ *and* $y \in dom(\sigma)$.

A variable ordering for an expression always exists: assume that all binders bind different variables and take the topological sort of the subexpression containment.

We define the application $\sigma \bullet r$ of an order-closed mapping σ to an order-respecting expression r by starting to substitute a maximal free variable by its image and repeat this process until all variables are eliminated.

Definition 8. *Let* $X \subseteq \mathcal{X}$ *a finite set of variables,* $r \in \mathbf{R}(\Sigma, X)$ *order-respecting, and* $\sigma : X \to \mathbf{R}(\Sigma, X)$ *be order-closed.*

The application $\sigma \bullet r \in \mathbf{R}(\Sigma, X)$ *yields an expression that is defined by substituting for the free variables in* r *in descending order.*

$$\sigma \bullet r = \begin{cases} r & fv(r) = \emptyset \\ \sigma \bullet r[\sigma(x)/x] & x \in \max(fv(r)) \text{ is a maximal element} \end{cases}$$

Application is well-defined because the variables x are drawn from the finite set X and the substitution step for x only introduces new variables that are strictly smaller than x due to order-closedness. The outcome does not depend on the choice of the maximal variable because the unfolding of a maximal variable cannot contain one of the other maximal variables. Furthermore, all intermediate expressions (and thus the result) are order-respecting.

4 Partial Derivatives

Antimirov [4] introduced partial derivatives to study the syntactic transformation from regular expressions to nondeterministic and deterministic finite automata. A partial derivative $\partial_a(r)$ with respect to an input symbol a maps an expression r to a set of expressions such that their union denotes the left quotient of $\mathcal{L}(r)$. Antimirov's definition corresponds to the left part of Fig. 1. We write $\mathbf{R}_o(\Sigma)$ for the set of ordinary regular expressions that neither contain the μ-operator nor any variables. We extend $\cdot$ to a function $(\cdot) : \wp(\mathbf{R}(\Sigma, X)) \times \mathbf{R}(\Sigma, X) \to \wp(\mathbf{R}(\Sigma, X))$ on sets of expressions R defined pointwise by

$$R \cdot s = \{r \cdot s \mid r \in R\}.$$

The definition of partial derivatives relies on nullability, which is tested by a function $\mathcal{N} : \mathbf{R}_o(\Sigma) \to \mathbb{B}$. The right side of the figure corresponds to Antimirov's definition.

Lemma 9. *For all $r \in \mathbf{R}_o(\Sigma)$, $\mathcal{N}(r)$ iff $\varepsilon \in \mathcal{L}(r)$.*

Theorem 10 (Correctness [4]). *For all $r \in \mathbf{R}_o(\Sigma)$, $a \in \Sigma$, $\mathcal{L}(\partial_a(r)) = a\backslash\mathcal{L}(r)$.*

Here we adopt the convention that if R is a set of expressions, then $\mathcal{L}(R)$ denotes the union of the languages of all expressions: $\mathcal{L}(R) = \bigcup\{\mathcal{L}(r) \mid r \in R\}$.

Theorem 11 (Expansion). *For $r \in \mathbf{R}_o(\Sigma)$, $\mathcal{L}(r) = \{\varepsilon \mid \mathcal{N}(r)\} \cup \bigcup_{a\in\Sigma} a \cdot \mathcal{L}(\partial_a(r))$.*

$$\begin{aligned}
\partial_a(\mathbf{0}) &= \{\} & \mathcal{N}(\mathbf{0}) &= \mathbf{ff}\\
\partial_a(\mathbf{1}) &= \{\} & \mathcal{N}(\mathbf{1}) &= \mathbf{tt}\\
\partial_a(b) &= \{\mathbf{1} \mid a = b\} & \mathcal{N}(a) &= \mathbf{ff}\\
\partial_a(r + s) &= \partial_a(r) \cup \partial_a(s) & \mathcal{N}(r + s) &= \mathcal{N}(r) \vee \mathcal{N}(s)\\
\partial_a(r \cdot s) &= \partial_a(r) \cdot s \cup \{s' \mid \mathcal{N}(r), s' \in \partial_a(s)\} & \mathcal{N}(r \cdot s) &= \mathcal{N}(r) \wedge \mathcal{N}(s)\\
\partial_a(r^*) &= \partial_a(r) \cdot r^* & \mathcal{N}(r^*) &= \mathbf{tt}
\end{aligned}$$

Fig. 1. Antimirov's definition of partial derivatives and nullability

Partial derivatives give rise to a nondeterministic finite automaton.

Theorem 12 (Finiteness [4]). *Let $r \in \mathbf{R}_o(\Sigma)$ be a regular expression. Define partial derivatives by words by $\partial_\varepsilon(r) = \{r\}$ and $\partial_{aw}(r) = \bigcup\{\partial_w(s) \mid s \in \partial_a(r)\}$ and by a language L by $\partial_L(r) = \bigcup\{\partial_w(r) \mid w \in L\}$.*

The set $\partial_{\Sigma^*}(r)$ is finite.

Theorem 13 (Nondeterministic finite automaton construction [4]). *Let $r \in \mathbf{R}_o(\Sigma)$ be a regular expression and define $Q = \partial_{\Sigma^*}(r)$, $\delta : Q \times \Sigma \to \wp(Q)$ by $(q, a, q') \in \delta$ iff $q' \in \partial_a(q)$. Let further $q_0 = r$ and $F = \{q \in Q \mid \mathcal{N}(q)\}$.*

Then $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is a NFA such that $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$.

The plan is to extend these results to μ-regular expressions. We start with the extension of the nullability function.

5 Nullability

Figure 2 extends nullability to μ-regular expressions. To cater for recursion, the $\mathcal{N}$ function obtains as a further argument a nullability environment ν of type $X \to \mathbb{B}$. With this extension, an expression $\mu x.r$ is deemed nullable if its body r is nullable. Furthermore, the least fixed point operator feeds back the nullability of the body to the free occurrences of the recursion variables. This fixed point is computed on the two-element Boolean lattice $\mathbb{B}$ ordered by $\mathbf{ff} \sqsubseteq \mathbf{tt}$ with disjunction $(\vee) : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ as the least upper bound operation. Thus, the case for a free variable x obtains its nullability information from the nullability environment.

Lemma 14. *For each $r \in \mathbf{R}(\Sigma, X)$, $\mathcal{N}(r)$ is a monotone function from $X \to \mathbb{B}$ (ordered pointwise) to $\mathbb{B}$.*

To prepare for the correctness proof of $\mathcal{N}$, we first simplify the case for the fixed point. It turns out that one iteration is sufficient to obtain the fixed point. This fact is also a consequence of a standard result, namely that the number

$$
\begin{aligned}
\mathcal{N}(\mathbf{0})\nu &= \mathbf{ff} \\
\mathcal{N}(\mathbf{1})\nu &= \mathbf{tt} \\
\mathcal{N}(a)\nu &= \mathbf{ff} \\
\mathcal{N}(r + s)\nu &= \mathcal{N}(r)\nu \vee \mathcal{N}(s)\nu \\
\mathcal{N}(r \cdot s)\nu &= \mathcal{N}(r)\nu \wedge \mathcal{N}(s)\nu \\
\mathcal{N}(r^*)\nu &= \mathbf{tt} \\
\mathcal{N}(\mu x.r)\nu &= \mathsf{lfp}\ b.\mathcal{N}(r)\nu[x \mapsto b] \\
\mathcal{N}(x)\nu &= \nu(x)
\end{aligned}
$$

Fig. 2. Nullability of μ-regular expressions

of iterations needed to compute the fixed point of a monotone function on a lattice is bounded by the height of the lattice. In this case, the Boolean lattice has height one.

Lemma 15. *Let X be a set of variables, $r \in \mathbf{R}(\Sigma, X \cup \{x\})$, $\eta : X \to \wp(\Sigma^*)$, and $L \subseteq \Sigma^*$ such that $\varepsilon \notin L$. If $\varepsilon \notin \mathcal{L}(r, \eta[x \mapsto \emptyset])$, then $\varepsilon \notin \mathcal{L}(r, \eta[x \mapsto L])$.*

Lemma 16. *For all $r \in \mathbf{R}(\Sigma, X)$, for all $\nu : X \to \mathbb{B}$,*

$$\mathit{lfp}\ b.\mathcal{N}(r)\nu[x \mapsto b] = \mathcal{N}(r)\nu[x \mapsto \mathbf{ff}].$$

For the statement of the correctness, we need to define what it means for a nullability environment to agree with a language environment.

Definition 17. *Nullability environment $\nu : X \to \mathbb{B}$* agrees *with language environment $\eta : X \to \wp(\Sigma^*)$, written $\eta \models \nu$, if for all $x \in X$, $\varepsilon \in \eta(x)$ iff $\nu(x)$.*

Lemma 18 (Correctness of $\mathcal{N}$). *For all X, $r \in \mathbf{R}(\Sigma, X)$, $\eta \in X \to \wp(\Sigma^*)$, $\nu \in X \to \mathbb{B}$, such that $\eta \models \nu$, it holds that $\varepsilon \in \mathcal{L}(r, \eta)$ iff $\mathcal{N}(r)\nu$.*

6 Derivation

The derivative for μ-regular expressions has a **different type** than for ordinary regular expressions: A partial derivative is a set of **non-empty sequences** (i.e., stack fragments) of regular expressions. The idea is that deriving a recursion operator $\mu x.r$ pushes the current context on the stack and starts afresh with the derivation of r. In other words, the derivative function for μ-regular expressions has the same signature as the transition function for a nondeterministic PDA.

To distinguish operations on stacks from operations on words over Σ, we write ":" (read "push") for the concatenation operator on stacks. We also use this operator for pattern matching parts of a stack. We write $[\,]$ for the empty stack, $[r_1, \ldots, r_n]$ for a stack with n elements, and $\overline{\mathbf{r}}$ for any stack of expressions. We extend the concatenation operator for regular expressions to non-empty stacks by having it operate on the **last** (bottom) element of a stack.

$$\begin{aligned}
\partial_\alpha^{\sigma,\nu}(\mathbf{0}) &= \{\} \\
\partial_\alpha^{\sigma,\nu}(\mathbf{1}) &= \{\} \\
\partial_\alpha^{\sigma,\nu}(b) &= \{[\mathbf{1}] \mid \alpha = b \in \Sigma\} \\
\partial_\alpha^{\sigma,\nu}(r + s) &= \partial_\alpha^{\sigma,\nu}(r) \cup \partial_\alpha^{\sigma,\nu}(s) \\
\partial_\alpha^{\sigma,\nu}(r \cdot s) &= \partial_\alpha^{\sigma,\nu}(r) \cdot (\sigma \bullet s) \cup \{\overline{\mathbf{s}} \mid \mathcal{N}(r)\nu, \overline{\mathbf{s}} \in \partial_\alpha^{\sigma,\nu}(s)\} \\
\partial_\alpha^{\sigma,\nu}(r^*) &= \partial_\alpha^{\sigma,\nu}(r) \cdot (\sigma \bullet r^*) \\
\partial_\alpha^{\sigma,\nu}(\mu x.r) &= \partial_\alpha^{\sigma[\mu x.r/x],\nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]}(r) : [\mathbf{1}] \\
\partial_\alpha^{\sigma,\nu}(x) &= \{[\sigma \bullet x] \mid \alpha = \varepsilon\}
\end{aligned}$$

Fig. 3. Partial derivatives of μ-regular expressions for $\alpha \in \Sigma \cup \{\varepsilon\}$

Definition 19. *Let $(M, (\cdot), \mathbf{1})$ be a monoid. We lift the monoid operation to non-empty stacks $(\cdot) \in M^+ \times M \to M^+$ for $\overline{a} \in M^*$ and $a, b \in M$ by*

$$(\overline{a} : [a]) \cdot b = (\overline{a} : [a \cdot b]).$$

We further lift it pointwise to sets $A \subseteq M^+$ to obtain $(\cdot) \in \wp(M^+) \times M \to \wp(M^+)$:

$$A \cdot b = \{\overline{a} \cdot b \mid \overline{a} \in A\}.$$

We use this definition for $M = \mathbf{R}(\Sigma, X)$ and also extend the push operation $(:)$ pointwise to sets of stacks.

$$\begin{aligned}(:) &\in \wp(\mathbf{R}(\Sigma, X)^+) \times \mathbf{R}(\Sigma, X)^+ \to \wp(\mathbf{R}(\Sigma, X)^+) \\ R : \overline{\mathbf{s}} &= \{\overline{\mathbf{r}} : \overline{\mathbf{s}} \mid \overline{\mathbf{r}} \in R\}\end{aligned}$$

Most of the time, the second argument will be a singleton stack $[s]$.

Before we discuss the intricacies of the full definition in Fig. 3, let's first consider a naive extension of the derivative function in Fig. 1 to μ-regular expressions and analyze its problems:

$$\partial_a(\mu x.r) = \partial_a(r[\mu x.r/x]) : [\mathbf{1}] \qquad \text{(naive unrolling: to be revised)}$$

Taking the derivative of a recursive definition means to apply the derivative to the unrolled definition. At the same time, we push an empty context on the stack so that the context of the recursion does not become a direct part of the derivative. This proposed definition makes sure that the partial derivative $\partial_a(r)$ is only ever applied to closed expressions $r \in \mathbf{R}(\Sigma)$. Hence, the case of a free recursion variable x would not occur during the computation of $\partial_a(r)$.

Example 20. The "naive unrolling" definition of the partial derivative has a problem. While it can be shown to be (partially) correct, it is not well-defined for all arguments. Consider the left-recursive expression $r = \mu x.\mathbf{1} + x \cdot a$, which is equivalent to a^*. Computing its partial derivative according to "naive unrolling" reveals that it depends on itself, so that $\partial_a(r)$ would be undefined.

$$\begin{aligned}\underline{\partial_a(r)} &= \partial_a(\mathbf{1} + r \cdot a) : [\mathbf{1}] \\ &= (\{\} \cup \partial_a(r \cdot a)) : [\mathbf{1}] \\ &= (\partial_a(r) \cdot a \cup \partial_a(a)) : [\mathbf{1}] \\ &= (\underline{\partial_a(r)} \cdot a \cup \{[\mathbf{1}]\}) : [\mathbf{1}]\end{aligned}$$

We remark that the expression r corresponds to a left-recursive grammar, where the naive construction of a top-down parser using the method of recursive descent also runs into problems [2]. There would be no problem with the right-recursive equivalent $r' = \mu x.\mathbf{1} + a \cdot x$ where the naive unrolling yields $\partial_a(r') = \{[r', \mathbf{1}]\}$. Indeed, the work by Winter and others [19] only allows guarded uses of the recursion operator, which rules out expressions like r from the start and which enables them to use the "naive unrolling" definition of the derivative.

For that reason, the derivative must not simply unroll recursions as they are encountered. Our definition distinguishes between left-recursive occurrences of a recursion variable, which must not be unrolled, and guarded occurrences, which can be unrolled safely. The derivative function remembers deferred unrollings in a substitution σ and applies them only when it is safe.

These observations lead to the signature of the definition of partial derivative in Fig. 3. Its type is

$$\partial : (\Sigma \cup \{\varepsilon\}) \times (X \to \mathbf{R}(\Sigma, X)) \times (X \to \mathbb{B}) \times \mathbf{R}(\Sigma, X) \to \wp(\mathbf{R}(\Sigma)^+)$$

and we write it as $\partial_\alpha^{\sigma,\nu}(r)$. It takes a symbol or an empty string $\alpha \in \Sigma \cup \{\varepsilon\}$ to derive, a substitution $\sigma : X \to \mathbf{R}(\Sigma, X)$ that maps free recursion variables to expressions (i.e., their unrollings), a nullability function $\nu : X \to \mathbb{B}$ that maps free recursion variables to their nullability, and the regular expression $r \in \mathbf{R}(\Sigma, X)$ to derive as arguments and returns the partial derivatives as a set of non-empty stacks of expressions.

Let's examine how the revised definition guarantees well-definedness. Example 20 demonstrates that left recursion is the cause for non-termination of the naive definition. The problem is that the naive definition indiscriminately substitutes all occurrences of x by its unfolding and propagates the derivative into the unfolding. However, this substitution is only safe in guarded positions (i.e., behind at least one terminal symbol in the unfolding). To avoid substitution in unguarded positions, the definition in Fig. 3 reifies this substitution as an additional argument σ and takes care to only apply it in guarded positions.

To introduce this recursion, the derived symbol α ranges over $\Sigma \cup \{\varepsilon\}$ in Fig. 3. For $\alpha = \varepsilon$, the derivative function unfolds one step of left recursion.

Example 21. Recall $r = \mu x.\mathbf{1} + x \cdot a$ from Example 20. Observe that $\mathcal{N}(r)\emptyset = \mathcal{N}(\mathbf{1} + x \cdot a)[\mathbf{ff}/x] = \mathbf{tt}$.

$$\begin{aligned}
\partial_a^{\emptyset,\emptyset}(r) &= (\partial_a^{[r/x],[\mathbf{tt}/x]}(\mathbf{1} + x \cdot a)) : [\mathbf{1}] \\
&= (\{\} \cup \partial_a^{[r/x],[\mathbf{tt}/x]}(x \cdot a)) : [\mathbf{1}] \\
&= (\{[\mathbf{1}]\}) : [\mathbf{1}] \\
&= \{[\mathbf{1}, \mathbf{1}]\}
\end{aligned}$$

The spontaneous derivative unfolds one level of left recursion.

$$\begin{aligned}
\partial_\varepsilon^{\emptyset,\emptyset}(r) &= (\partial_\varepsilon^{[r/x],[\mathbf{tt}/x]}(\mathbf{1} + x \cdot a)) : [\mathbf{1}] \\
&= (\{\} \cup \partial_\varepsilon^{[r/x],[\mathbf{tt}/x]}(x \cdot a)) : [\mathbf{1}] \\
&= (\{[r \cdot a]\}) : [\mathbf{1}] \\
&= \{[r \cdot a, \mathbf{1}]\}
\end{aligned}$$

Thus, the spontaneous derivative corresponds to ε-transitions of the PDA that is to be constructed.

7 Correctness

To argue about the correctness of our derivative operation, we define the membership of a word $w \in \Sigma^*$ in the language of an order-respecting expression $r \in \mathbf{R}(\Sigma, X)$ under an order-closed mapping $\sigma : X \to \mathbf{R}(\Sigma, X)$ inductively by the judgment $\sigma \vdash w \in r$ in Fig. 4 along with $\sigma \vdash w \in \overline{\mathbf{r}}$ for an expression stack $\overline{\mathbf{r}}$ and $\sigma \vdash w \in R$ for a set of such stacks $R \subseteq \mathbf{R}(\Sigma, X)^*$. This inductive definition mirrors the previous fixed point definition of the language of an expression.

Lemma 22. *For all $r \in \mathbf{R}(\Sigma)$ and $w \in \Sigma^*$. $\emptyset \vdash w \in r$ iff $w \in \mathcal{L}(r)$.*

It is straightforward to prove the following derived rule.

$$\boxed{\sigma \vdash w \in r} \qquad \sigma \vdash \varepsilon \in \mathbf{1} \qquad \sigma \vdash a \in a \qquad \frac{\sigma \vdash w \in r}{\sigma \vdash w \in r + s} \qquad \frac{\sigma \vdash w \in s}{\sigma \vdash w \in r + s}$$

$$\frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in s}{\sigma \vdash vw \in r \cdot s} \qquad \sigma \vdash \varepsilon \in r^* \qquad \frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in r^*}{\sigma \vdash vw \in r^*}$$

$$\frac{\sigma[\mu x.r/x] \vdash w \in r}{\sigma \vdash w \in \mu x.r} \qquad \frac{\sigma \vdash w \in \mu x.r}{\sigma[\mu x.r/x] \vdash w \in x}$$

$$\boxed{\sigma \vdash w \in \overline{\mathbf{r}}} \qquad \sigma \vdash \varepsilon \in [] \qquad \frac{\sigma \vdash v \in r \qquad \sigma \vdash w \in \overline{\mathbf{r}}}{\sigma \vdash vw \in [r] : \overline{\mathbf{r}}}$$

$$\boxed{\sigma \vdash w \in R} \qquad \frac{\sigma \vdash w \in \overline{\mathbf{r}} \qquad \overline{\mathbf{r}} \in R}{\sigma \vdash w \in R}$$

Fig. 4. Membership in a μ-regular expression, a stack of expressions, and a set of stacks

Lemma 23. *If $R \subseteq S \subseteq \mathbf{R}(\Sigma, X)^*$, then $\sigma \vdash w \in R$ implies $\sigma \vdash w \in S$.*

Lemma 24. *Let $r \in \mathbf{R}(\Sigma, X)$ and $\sigma : X \to \mathbf{R}(\Sigma, X)$ be order-respecting. If $\sigma \vdash w \in r$, then $\emptyset \vdash w \in \sigma \bullet r$.*

The derivation closure $\tilde{\partial}_a(\overline{\mathbf{r}})$ of a non-empty closed stack of expressions is defined by the union of the partial derivatives after taking an arbitrary number of ε-steps. It is our main tool in proving the correctness of the derivative.

Definition 25. *For $a \in \Sigma$, the derivation closure $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}})$ is inductively defined as the smallest set of stacks such that*

1. $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}}) \supseteq \partial_a^{\sigma,\nu}(r) : \overline{\mathbf{r}}$ *and*
2. $\tilde{\partial}_a^{\sigma,\nu}(r : \overline{\mathbf{r}}) \supseteq \bigcup\{\tilde{\partial}_a^{\sigma,\nu}(\overline{\mathbf{s}} : \overline{\mathbf{r}}) \mid \overline{\mathbf{s}} \in \partial_\varepsilon^{\sigma,\nu}(r)\}$.

Lemma 26 (Unfolding). *Let $r \in \mathbf{R}(\Sigma, X)$ an order-respecting expression, $\sigma : X \to \mathbf{R}(\Sigma, X)$ order-closed with $\sigma(x) = \mu x.s_x$, $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$.*

$$\sigma \vdash w \in r \Leftarrow \emptyset \vdash w \in \partial_\varepsilon^{\sigma,\nu}(r)$$

Theorem 27 (Correctness). *Let $r \in \mathbf{R}(\Sigma, X)$ an order-respecting expression, $\sigma : X \to \mathbf{R}(\Sigma, X)$ order-closed with $\sigma(x) = \mu x.s_x$, $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$.*

$$\sigma \vdash aw \in r \Leftrightarrow \emptyset \vdash w \in \tilde{\partial}_a^{\sigma,\nu}([r])$$

Proof. The direction from left to right is proved by induction on $\sigma \vdash aw \in r$.

We demonstrate the right-to-left direction.

Suppose that $\Delta = \emptyset \vdash w \in \tilde{\partial}_a^{\sigma,\nu}([r])$ and show that $\sigma \vdash aw \in r$.

The proof is by induction on the size of the derivation of Δ. Inversion yields that there is some $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}([r])$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}$. Now there are two cases.

Case $w = \varepsilon$ and $\overline{\mathbf{r}} = []$ so that the empty-sequence-rule $\emptyset \vdash \varepsilon \in []$ applies. But this case cannot happen because $\overline{\mathbf{r}} \neq []$.

Case $\emptyset \vdash vw \in [s] : \overline{\mathbf{r}}$ because $\emptyset \vdash v \in s$ and $\emptyset \vdash w \in \overline{\mathbf{r}}$.

These two cases boil down to $w = w_1 \ldots w_n$, $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$, for some $n \geq 1$, and $\emptyset \vdash w_1 \ldots w_n \in [r_1, \ldots, r_n]$ because $\emptyset \vdash w_i \in r_i$.

We perform an inner induction on r.

Case $\mathbf{0}$, $\mathbf{1}$, $b \neq a$: contradictory because $\tilde{\partial}_a^{\sigma,\nu}([r]) = \emptyset$.

Case a: $\tilde{\partial}_a^{\sigma,\nu}([a]) = \{[\mathbf{1}]\}$ so that $w = \varepsilon$. Clearly, $\sigma \vdash a \in a$.

Case $r + s$: We can show that $\tilde{\partial}_a^{\sigma,\nu}(r + s) = \tilde{\partial}_a^{\sigma,\nu}(r) \cup \tilde{\partial}_a^{\sigma,\nu}(s)$. Assuming that $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r)$, induction on r yields $\sigma \vdash aw \in r$ and the $+$-rule yields $\sigma \vdash aw \in r+s$. Analogously for s.

Case $r \cdot s$: We can show that $\tilde{\partial}_a^{\sigma,\nu}(r \cdot s) = \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet s) \cup \{\overline{\mathbf{s}} \mid \mathcal{N}(r)\nu, \overline{\mathbf{s}} \in \tilde{\partial}_a^{\sigma,\nu}(s)\}$. There are two cases.

Subcase $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet s)$. Hence, $\overline{\mathbf{r}} = [r_1, \ldots, r_n \cdot (\sigma \bullet s)]$ so that $w = w_1 \ldots w_n w_{n+1}$ and $\emptyset \vdash w_1 \in r_1$, $\ldots$, $\emptyset \vdash w_n \in r_n$, and $\emptyset \vdash w_{n+1} \in (\sigma \bullet s)$. Now, $\overline{\mathbf{r}}' = [r_1, \ldots, r_n] \in \tilde{\partial}_a^{\sigma,\nu}(r)$ and thus $\emptyset \vdash w_1 \ldots w_n \in \tilde{\partial}_a^{\sigma,\nu}(r)$. By induction, $\sigma \vdash aw_1 \ldots w_n \in r$. Because $\sigma \bullet s$ is closed, we also have $\emptyset \vdash w_{n+1} \in (\sigma \bullet s)$ and thus by Lemma 24 $\sigma \vdash w_{n+1} \in s$. Taken together $\sigma \vdash aw_1 \ldots w_n w_{n+1} \in r \cdot s$.

Subcase $\mathcal{N}(r)\nu$ and $\overline{\mathbf{r}} \in \partial_a^{\sigma,\nu}(s)$. Hence, $\sigma \vdash \varepsilon \in r$, by induction $\sigma \vdash aw \in s$, and the concatenation rule yields $\sigma \vdash aw \in r \cdot s$.

Case r^*. Because $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\sigma,\nu}(r) \cdot (\sigma \bullet r^*)$, it must be that $\overline{\mathbf{r}} = [r_1, \ldots, r_n \cdot (\sigma \bullet r^*)]$ and $w = w_1 \ldots w_n w_{n+1}$ so that $\emptyset \vdash w_1 \in r_1, \ldots, \emptyset \vdash w_n \in r_n$, and $\emptyset \vdash w_{n+1} \in (\sigma \bullet r^*)$. Proceed as in the first subcase for concatenation.

Case $\mu x.r$. As usual, let $\hat{\sigma} = \sigma[\mu x.r/x]$ and $\hat{\nu} = \nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]$. Again, $\tilde{\partial}_a^{\sigma,\nu}(\mu x.r) = \tilde{\partial}_a^{\hat{\sigma},\hat{\mu}}(r) : [\mathbf{1}]$. Hence, $\overline{\mathbf{r}} = \overline{\mathbf{r}}' : [\mathbf{1}]$ for some $\overline{\mathbf{r}}' \in \tilde{\partial}_a^{\hat{\sigma},\hat{\mu}}(r)$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}'$. Induction yields that $\hat{\sigma} \vdash aw \in r$ and application of the μ-rule yields $\sigma \vdash aw \in \mu x.r$.

Case x. Then $\tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x) = \tilde{\partial}_a^{\sigma,\nu}(\mu x.r)$ if $\hat{\sigma} = \sigma[\mu x.r/x]$ and $\hat{\nu} = \nu[\mathcal{N}(r)\nu[\mathbf{ff}/x]/x]$.

Now $\emptyset \vdash w \in \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x)$ iff exists $\overline{\mathbf{r}} \in \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(x) = \tilde{\partial}_a^{\sigma,\nu}(\mu x.r)$ such that $\emptyset \vdash w \in \overline{\mathbf{r}}$. But $\tilde{\partial}_a^{\sigma,\nu}(\mu x.r) = \tilde{\partial}_a^{\hat{\sigma},\hat{\nu}}(r) : [\mathbf{1}]$ so that $\overline{\mathbf{r}} = \overline{\mathbf{r}}' : [\mathbf{1}]$ and $\emptyset \vdash w \in \overline{\mathbf{r}}'$ with a smaller derivation tree. Thus, induction yields that $\hat{\sigma} \vdash aw \in r$, application of the μ-rule yields $\sigma \vdash aw \in \mu x.r$, and application of the variable rule yields $\hat{\sigma} \vdash aw \in x$, as desired. □

8 Finiteness

In analogy to Antimirov's finite automaton construction, we aim to use the set of iterated derivatives as a building block for a pushdown automaton. In our construction, derivatives end up as pushdown symbols rather than states: the top of the pushdown plays the role of the state. It remains to prove that this set is finite to obtain a proper PDA.

Our finiteness argument is based on an analysis of the syntactical form of the derivatives. It turns out that a derivative is, roughly, a concatenation of a strictly descending sequence of certain subexpressions of the initial expression. As this ordering is finite, we obtain a finite bound on the syntactically possible derivatives.

We start with an analysis of the output of $\partial_\alpha^{\sigma,\nu}(r)$. The elements in the stack of a partial derivative are vectors of the form $((h \cdot s_1) \cdot s_2) \cdots s_k$ that we abbreviate $h \cdot \vec{s}$, where the s_i are arbitrary expressions and h is either $\mathbf{1}$ or $\mu x.r$ where $\mu x.r$ is closed.

It turns out that the vectors produced by derivation are always strictly ascending chains in the subterm ordering of the original expression, say t. We first define this ordering, then we define the structure of these vectors in Definition 31.

Definition 28. *Let $r \in \mathbf{R}(\Sigma)$ be a closed expression. We define the* addressing function $A_r : \mathbb{N}^* \hookrightarrow \mathbf{R}(\Sigma, X)$ *by induction on r.*

$$
\begin{aligned}
A_{\mathbf{0}} &= \{(\varepsilon, \mathbf{0})\} & A_{r+s} &= \{(\varepsilon, r+s)\} \cup 1.A_r \cup 2.A_s \\
A_{\mathbf{1}} &= \{(\varepsilon, \mathbf{1})\} & A_{r \cdot s} &= \{(\varepsilon, r \cdot s)\} \cup 1.A_r \cup 2.A_s \\
A_a &= \{(\varepsilon, a)\} & A_{r^*} &= \{(\varepsilon, r^*)\} \cup 1.A_r \\
A_x &= \{(\varepsilon, x)\} & A_{\mu x.r} &= \{(\varepsilon, \mu x.r)\} \cup 1.A_r
\end{aligned}
$$

Here $i.A$ modifies the function A by prepending i to each element of A's domain:

$$
(i.A)(w) = \begin{cases} A(w') & w = iw' \text{ and } A(w') \text{ defined} \\ \text{undefined} & \text{otherwise.} \end{cases}
$$

It is well known that $dom(A_r)$ is prefix-closed and assigns a unique $w \in \mathbb{N}^*$ to each occurrence of a subexpression in r. Let $r_1 = A_r(w_1)$ and $r_2 = A_r(w_2)$ be subexpression occurrences of r. We say that r_1 *occurs before* r_2 *in* r if $w_1 \preceq w_2$ in the lexicographic order on $\mathbb{N}^*$:

$$\varepsilon \preceq w \qquad \frac{i < j}{iv \preceq jw} \qquad \frac{v \preceq w}{iv \preceq iw}$$

We write $w_1 \prec w_2$ if $w_1 \preceq w_2$ and $w_1 \neq w_2$, in which case we say that r_1 occurs strictly before r_2.

Lemma 29. *For each closed expression $r \in \mathbf{R}(\Sigma)$, the strict lexicographic ordering $\prec$ on $dom(A_r)$ has no infinite chains.*

Definition 30. *Let $t \in \mathbf{R}(\Sigma)$ be a closed expression such that each variable occurring in t is bound exactly once. The* unfolding substitution σ_t *is defined by induction on t.*

$$\begin{aligned} \sigma_{\mathbf{0}} &= [] & \sigma_{r+s} &= \sigma_r \cup \sigma_s \\ \sigma_{\mathbf{1}} &= [] & \sigma_{r \cdot s} &= \sigma_r \cup \sigma_s \\ \sigma_a &= [] & \sigma_{r^*} &= \sigma_r \\ \sigma_x &= [] & \sigma_{\mu x.r} &= [\mu x.r/x] \cup \sigma_r \end{aligned}$$

Definition 31. *A vector $\vec{s} = (s_1 \cdot s_2) \cdots s_k$ is* t-sorted *if for all $1 \leq i < j \leq k$: s_i and s_j are subexpressions of t and s_i occurs strictly before s_j, which means that there are $w_1, \ldots, w_k \in \mathbb{N}^*$ such that $s_i = A_t(w_i)$ and $w_i \prec w_{i+1}$, for $1 \leq i < k$.*

For a t-sorted vector $\vec{s} = (s_1 \cdot s_2) \cdots s_k$ define two forms of expressions:

top: $\sigma_t \bullet (\mathbf{1} \cdot \vec{s})$.
rec: $\sigma_t \bullet ((\mu x.s) \cdot \vec{s})$ *where $\mu x.s$ is a subexpression of t and either $\mu x.s$ or an occurrence of x is strictly before s_i, for all $1 \leq i \leq k$.*

A stack $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$ (for $n \geq 1$) has form $\mathbf{top}^+$ *if $r_1, \ldots, r_n$ have form* **top**.

A stack $\overline{\mathbf{r}} = [r_1, \ldots, r_n]$ (for $n \geq 1$) has form $\mathbf{rec}.\mathbf{top}^*$ *if r_1 has form* **rec** *and $r_2, \ldots, r_n$ have form* **top**.

Next, we show that all derivatives and partial derivatives of subexpressions of a closed expression t have indeed one of the forms $\mathbf{top}^+$ or $\mathbf{rec}.\mathbf{top}^*$.

Lemma 32 (Classification of derivatives). *Suppose that $t \in \mathbf{R}(\Sigma)$ is a closed expression, $r \in \mathbf{R}(\Sigma, X)$ is a subexpression of t, $\sigma : X \to \mathbf{R}(\Sigma, X)$ is order-closed with $\sigma(x) = \mu x.s$ (for $x \in X$ and $\mu x.s$ a subterm of t), and $\nu : X \to \mathbb{B}$ such that $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$. If $\overline{\mathbf{r}} = [r_1, \ldots, r_n] \in \partial_a^{\sigma,\nu}(r)$, then $n \geq 1$ and $\overline{\mathbf{r}}$ has form $\mathbf{top}^+$ and each $r_i = h_i \cdot \vec{s_i}$ for some t-sorted $\vec{s_i}$ which is before r.*

Lemma 33 (Classification of spontaneous derivatives). *Suppose that* $t \in \mathbf{R}(\Sigma)$ *is a closed expression,* $r \in \mathbf{R}(\Sigma, X)$ *is a subexpression of* t*,* $\sigma : X \to \mathbf{R}(\Sigma, X)$ *is order-closed with* $\sigma(x) = \mu x.s$ *(for* $x \in X$ *and* $\mu x.s$ *a subterm of* t*), and* $\nu : X \to \mathbb{B}$ *such that* $\nu(x) = \mathcal{N}(\sigma \bullet x)\emptyset$*. If* $\bar{\mathbf{r}} = [r_1, \ldots, r_n] \in \partial_{\varepsilon}^{\sigma,\nu}(r)$*, then* $n \geq 1$ *and* $\bar{\mathbf{r}}$ *has form* $\mathbf{rec}.\mathbf{top}^*$ *and each* $r_i = h_i \cdot \vec{s_i}$ *for some* t*-sorted* $\vec{s_i}$ *which is before* r*.*

Lemma 34 (Classification of derivatives of vectors). *Let* $t \in R(\Sigma)$ *be a closed expression and* t_0 *be closed of form* $\mathbf{top}$ *or form* $\mathbf{rec}$ *with respect to* t*. Then the elements of* $\partial_a^{\emptyset,\emptyset}(t_0)$ *are stacks of the form* $\mathbf{top}^+$ *as in Lemma 32 and the elements of* $\partial_{\varepsilon}^{\emptyset,\emptyset}(t_0)$ *are stacks of the form* $\mathbf{rec}.\mathbf{top}^*$*.*

We define the set of iterated partial derivatives as the expressions that may show up in the stack of a partial derivative. This set will serve as the basis for defining the set of pushdown symbols of a PDA.

Definition 35 (Iterated Partial Derivatives). *Let* $t \in \mathbf{R}(\Sigma)$ *be a closed expression. Define* $\Delta(t)$*, the set of iterated partial derivatives of* t*, as the smallest set such that*

- $\mathbf{1} \cdot t \in \Delta(t)$*;*
- *if* $r \in \Delta(t)$ *and* $[t_1, \ldots, t_n] \in \partial_a^{\emptyset,\emptyset}(r)$*, then* $t_j \in \Delta(t)$*, for all* $1 \leq j \leq n$*; and*
- *if* $r \in \Delta(t)$ *and* $[t_1, \ldots, t_n] \in \partial_{\varepsilon}^{\emptyset,\emptyset}(r)$*, then* $t_j \in \Delta(t)$*, for all* $1 \leq j \leq n$*.*

Lemma 36 (Closure). *Let* $t \in \mathbf{R}(\Sigma)$ *be a closed expression. Then all elements of* $\Delta(t)$ *either have form* $\mathbf{top}$ *or* $\mathbf{rec}$ *with respect to* t*.*

Proof. Follows from Lemmas 32, 33, and 34.

Lemma 37 (Finiteness). *Let* $t \in \mathbf{R}(\Sigma)$ *be closed. Then* $\Delta(t)$ *is finite.*

Proof. By construction, the elements of $\Delta(t)$ are all closed and have either form $\mathbf{top}$ or form $\mathbf{rec}$, which is a vector of the form $\sigma_t \bullet (h \cdot \vec{s})$ where $\vec{s}$ is t-sorted. As t is a finite expression and a t-sorted vector is strictly decreasing, there are only finitely many candidates for $\vec{s}$ (by Lemma 29).

The head h of the vector is either $\mathbf{1}$ or it is a subexpression of t of the form $\mu x.s_x$. Hence, there are only finitely many choices for h.

Thus $\Delta(t)$ is a subset of a finite set and hence finite. □

9 Automaton Construction

Given that the derivative for a closed μ-regular expression gives rise to a finite set of iterated partial derivatives, we use that set as the pushdown alphabet to construct a nondeterministic pushdown automaton that recognizes the same language. This construction is straightforward as its transition function corresponds exactly to the derivative and the spontaneous derivative function.

Definition 38. *Suppose that $t \in \mathbf{R}(\Sigma)$ is closed. Define the PDA $\mathcal{UA}(t) = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ by a singleton set $Q = \{q\}$, $\Gamma = \Delta(t)$, $q_0 = q$, $Z_0 = \mathbf{1} \cdot r$, and $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ as the smallest relation such that*

- $(q, a, s, q, \overline{\mathbf{s}}) \in \delta$ *if* $\overline{\mathbf{s}} \in \partial_a^{\emptyset,\emptyset}(s)$, *for all* $s \in \Gamma$, $\overline{\mathbf{s}} \in \Gamma^*$, $a \in \Sigma$;
- $(q, \varepsilon, s, q, \overline{\mathbf{s}}) \in \delta$ *if* $\overline{\mathbf{s}} \in \partial_\varepsilon^{\emptyset,\emptyset}(s)$, *for all* $s \in \Gamma$, $\overline{\mathbf{s}} \in \Gamma^*$;
- $(q, \varepsilon, s, q, \varepsilon)$, *for all* $s \in \Gamma$ *with* $\mathcal{N}(s)\emptyset$.

Theorem 39 (Automaton correctness). *For all closed expressions $t \in R(\Sigma)$, $\mathcal{L}(t) = \mathcal{L}(\mathcal{UA}(t))$.*

Proof. Let $\mathcal{UA}(t) = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$. We prove a generalized statement from which the original statement follows trivially: for all $\overline{\mathbf{r}} \in \Delta(t)^*$, $\emptyset \vdash w \in \overline{\mathbf{r}}$ iff $(q, \overline{\mathbf{r}}, w) \vdash^* (q, \varepsilon, \varepsilon)$. The proof in the left-to-right direction is by induction on the derivation of $\emptyset \vdash w \in \overline{\mathbf{r}}$.

Case $\emptyset \vdash \varepsilon \in []$. Immediate.

Case $\emptyset \vdash w \in \overline{\mathbf{r}}$ because $w = w_1 w_2$, $\overline{\mathbf{r}} = [r] : \overline{\mathbf{r}}'$, $\emptyset \vdash w_1 \in r$, and $\emptyset \vdash w_2 \in \overline{\mathbf{r}}'$. By induction, we find that $(q, [r], w_1) \vdash^* (q, [], \varepsilon)$. By a standard argument that means $(q, [r] : \overline{\mathbf{r}}', w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}', w_2)$. By the second inductive hypothesis, we find that $(q, \overline{\mathbf{r}}', w_2) \vdash^* (q, [], \varepsilon)$. Taken together, we obtain the desired result.

Now we consider the derivation of $\emptyset \vdash w \in r$ by performing a case analysis on w and using Lemma 27.

Case ε. In this case, $\emptyset \vdash \varepsilon \in r$ iff $\mathcal{N}(r_i)\emptyset$ iff $(q, \varepsilon, r, q, \varepsilon) \in \delta$ so that $(q, [r], \varepsilon) \vdash^+ (q, \varepsilon, \varepsilon)$.

Case aw. In this case $\emptyset \vdash aw \in r$. By Lemma 27, $\emptyset \vdash aw \in r$ is equivalent to $\emptyset \vdash w \in \tilde{\partial}_a^{\emptyset,\emptyset}([r])$ and we perform a subsidiary induction on its definition. That is, either $\exists \overline{\mathbf{s}} \in \partial_a^{\emptyset,\emptyset}(r)$ such that $\emptyset \vdash w \in \overline{\mathbf{s}}$. In that case, $\mathcal{UA}(t)$ has a transition $(q, r, aw) \vdash (q, \overline{\mathbf{s}}, w)$ by definition of δ. By induction we know that $(q, \overline{\mathbf{s}}, w) \vdash^+ (q, \varepsilon, \varepsilon)$.

Alternatively, $\exists \overline{\mathbf{s}} \in \partial_\varepsilon^{\emptyset,\emptyset}(r)$ such that $\emptyset \vdash aw \in \overline{\mathbf{s}}$. In this case, $(q, r, aw) \vdash (q, \overline{\mathbf{s}}, aw)$ is a transition and by induction we have $(q, \overline{\mathbf{s}}, aw) \vdash^+ (q, \varepsilon, \varepsilon)$.

Right-to-left direction. By induction on the length of $(q, \overline{\mathbf{r}}, w) \vdash^* (q, [], \varepsilon)$.

Case length 0: it must be $\overline{\mathbf{r}} = []$ and $w = \varepsilon$. Obviously, $\emptyset \vdash \varepsilon \in []$.

Case length > 0: Thus the first configuration must have the form $(q, [s] : \overline{\mathbf{r}}, w)$. There are three possibilities.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w')$ if $w = aw'$ and $\overline{\mathbf{s}} \in \partial_a(s)$. We split the run of the automaton at the point where $\overline{\mathbf{s}}$ is first consumed: let $w' = w_1 w_2$ such that $(q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}, w_2) \vdash^* (q, [], \varepsilon)$. Hence, there is also a shorter run on w_1: $(q, \overline{\mathbf{s}}, w_1) \vdash^* (q, [], \varepsilon)$. Induction yields $\emptyset \vdash w_1 \in \overline{\mathbf{s}}$. By Lemma 27, we also have a derivation $\emptyset \vdash aw_1 \in s$. By induction on the $\overline{\mathbf{r}}$ run, we obtain $\emptyset \vdash w_2 \in \overline{\mathbf{r}}$ and applying the stack rule yields $\emptyset \vdash aw_1 w_2 \in [s] : \overline{\mathbf{r}}$ or in other words $\emptyset \vdash w \in [s] : \overline{\mathbf{r}}$.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w)$ if $\overline{\mathbf{s}} \in \partial_\varepsilon(s)$. We split the run of the automaton at the point where $\overline{\mathbf{s}}$ is first consumed: let $w' = w_1 w_2$ such that $(q, \overline{\mathbf{s}} : \overline{\mathbf{r}}, w_1 w_2) \vdash^* (q, \overline{\mathbf{r}}, w_2) \vdash^* (q, [], \varepsilon)$. Hence there is also a shorter run on w_1: $(q, \overline{\mathbf{s}}, w_1) \vdash^* (q, [], \varepsilon)$. By induction, we have a derivation $\emptyset \vdash w_1 \in \overline{\mathbf{s}}$, which yields $\emptyset \vdash w_1 \in s$ by Lemma 26, and a derivation $\emptyset \vdash w_2 \in \overline{\mathbf{r}}$, which we can combine to $\emptyset \vdash w_1 w_2 \in [s] : \overline{\mathbf{r}}$ as desired.

Subcase $(q, [s] : \overline{\mathbf{r}}, w) \vdash (q, \overline{\mathbf{r}}, w)$ if $\mathcal{N}(s)\emptyset$. By induction, $\emptyset \vdash w \in \overline{\mathbf{r}}$. As $\mathcal{N}(s)\emptyset$, it must be that $\emptyset \vdash \varepsilon \in s$. Hence, $\emptyset \vdash w \in [s] : \overline{\mathbf{r}}$. □

If all recursion operators in an expression t are guarded, in the sense that they consume some input before entering a recursive call, then all ε-transitions in the constructed automaton pop the stack. In fact, when restricting to guarded expressions, the spontaneous derivative function is not needed at all, which explains the simplicity of the derivative in the work of Winter and coworkers [19].

Acknowledgments. The thoughtful comments of the anonymous reviewers helped improve the presentation of this paper.

References

1. Adams, M.D., Hollenbeck, C., Might, M.: On the complexity and performance of parsing with derivatives. In: PLDI, pp. 224–236. ACM (2016)
2. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers. Principles, Techniques, and Tools. Addison-Wesley, Boston (2007)
3. Antimirov, V.: Rewriting regular inequalities. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 965, pp. 116–125. Springer, Heidelberg (1995). doi:10.1007/3-540-60249-6_44
4. Antimirov, V.M.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155**(2), 291–319 (1996)
5. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
6. Caron, P., Champarnaud, J.-M., Mignot, L.: Partial derivatives of an extended regular expression. In: Dediu, A.-H., Inenaga, S., Martín-Vide, C. (eds.) LATA 2011. LNCS, vol. 6638, pp. 179–191. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21254-3_13
7. Caron, P., Champarnaud, J.-M., Mignot, L.: Multi-tilde-bar derivatives. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 321–328. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31606-7_28
8. Caron, P., Champarnaud, J., Mignot, L.: A general framework for the derivation of regular expressions. RAIRO - Theor. Inf. Appl. **48**(3), 281–305 (2014)
9. Champarnaud, J.-M., Jeanne, H., Mignot, L.: Approximate regular expressions and their derivatives. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 179–191. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28332-1_16
10. Grabmayer, C.: Using proofs by coinduction to find "Traditional" Proofs. In: Fiadeiro, J.L., Harman, N., Roggenbach, M., Rutten, J. (eds.) CALCO 2005. LNCS, vol. 3629, pp. 175–193. Springer, Heidelberg (2005). doi:10.1007/11548133_12

11. Leiß, H.: Towards Kleene algebra with recursion. In: Börger, E., Jäger, G., Kleine Büning, H., Richter, M.M. (eds.) CSL 1991. LNCS, vol. 626, pp. 242–256. Springer, Heidelberg (1992). doi:10.1007/BFb0023771
12. Lombardy, S., Sakarovitch, J.: Derivatives of rational expressions with multiplicity. Theor. Comput. Sci. **332**(1–3), 141–177 (2005)
13. Might, M., Darais, D., Spiewak, D.: Parsing with derivatives: a functional pearl. In: Proceedings of ICFP 2011, pp. 189–195. ACM (2011)
14. Owens, S., Reppy, J., Turon, A.: Regular-expression derivatives reexamined. J. Funct. Program. **19**(2), 173–190 (2009)
15. Roşu, G., Viswanathan, M.: Testing extended regular language membership incrementally by rewriting. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 499–514. Springer, Heidelberg (2003). doi:10.1007/3-540-44881-0_35
16. Sulzmann, M., Thiemann, P.: Derivatives for regular shuffle expressions. In: Dediu, A.-H., Formenti, E., Martín-Vide, C., Truthe, B. (eds.) LATA 2015. LNCS, vol. 8977, pp. 275–286. Springer, Cham (2015). doi:10.1007/978-3-319-15579-1_21
17. Thiemann, P.: Derivatives for enhanced regular expressions. In: Han, Y.-S., Salomaa, K. (eds.) CIAA 2016. LNCS, vol. 9705, pp. 285–297. Springer, Cham (2016). doi:10.1007/978-3-319-40946-7_24
18. Watson, B.W.: FIRE lite: FAs and REs in C++. In: Raymond, D., Wood, D., Yu, S. (eds.) WIA 1996. LNCS, vol. 1260, pp. 167–188. Springer, Heidelberg (1997). doi:10.1007/3-540-63174-7_14
19. Winter, J., Bonsangue, M.M., Rutten, J.: Context-free languages, coalgebraically. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 359–376. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22944-2_25