Derivation Beyond Regular Languages

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Peter Thiemann[1]

[1]University of Freiburg

14 Oct 2018

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Languages

UNI FREIBURG

Notation

- Σ alphabet (a finite set)
- Σ^* the set of (finite) words over Σ
- $\mathbb{P}(\Sigma^*)$ the set of languages over Σ
- $a, b, c, \ldots \in \Sigma$
- $u, v, w, \ldots \in \Sigma^*$; empty word ε, concatenation $\cdot$
- $L, U, V, \ldots \in \mathbb{P}(\Sigma^*)$

Operations on Languages

UNI FREIBURG

Product and Power

$$U \cdot V = \{u \cdot v \mid u \in U, v \in V\}$$
$$U^0 = \{\varepsilon\}$$
$$U^{n+1} = U \cdot U^n$$

Kleene Closure

$$U^* = \bigcup_{n \geq 0} U^n$$

Finite Automata

A nondeterministic finite automaton (NFA) $\mathcal{M} = (Q, \Sigma, I, \delta, F)$ consists of

- Q finite set of states
- Σ
- $I \subseteq Q$ initial states
- $\delta : Q \times \Sigma \to \mathbb{P}Q$ transition function
- $F \subseteq Q$ accepting states

$\mathcal{M}$ is deterministic (DFA) if $|I| = 1$ and, for all q, a, $|\delta(q, a)| = 1$.

Runs and Languages

Run

A run of $\mathcal{M}$ on $a_1 \dots a_n \in \Sigma^*$ is a sequence $q_0, \dots, q_n \in Q^*$ s.t.

- $q_0 \in I$,
- for all $0 \leq i < n$, $q_{i+1} \in \delta(q, a_i)$.

The run is accepting if $q_n \in F$.

Language of Automaton $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^* \mid \mathcal{M} \text{ has an accepting run on } w\}$$

Regular Languages

Regular Expressions

- Syntax

$$RE_\Sigma \ni r, s ::= \mathbf{0} \mid \mathbf{1} \mid a \mid r + s \mid r.s \mid r^*$$

- Semantics $[\![\cdot]\!] : RE_\Sigma \to \mathbb{P}(\Sigma^*)$

$$\begin{aligned}
[\![\mathbf{0}]\!] &= \emptyset \\
[\![\mathbf{1}]\!] &= \{\varepsilon\} \\
[\![a]\!] &= \{a\} \\
[\![r + s]\!] &= [\![r]\!] \cup [\![s]\!] \\
[\![r.s]\!] &= [\![r]\!] \cdot [\![s]\!] \\
[\![r^*]\!] &= [\![r]\!]^*
\end{aligned}$$

From Regular Expressions to Finite Automata

UNI FREIBURG

Thompson's construction

- RE $\rightarrow$ NFA-ε $\rightarrow$ NFA $\rightarrow$ DFA
- three steps: one compositional, two global automata transformations
- key: transitive closure, powerset construction
- standard technique

From Regular Expressions to Finite Automata

UNI FREIBURG

Thompson's construction

- RE $\rightarrow$ NFA-ε $\rightarrow$ NFA $\rightarrow$ DFA
- three steps: one compositional, two global automata transformations
- key: transitive closure, powerset construction
- standard technique

Brzozowski's construction

- RE $\rightarrow$ DFA
- one compositional step: algebraic flavor
- key: derivative of RE

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Foundations of Derivation

UNI FREIBURG

Definition: Left quotient

$U, V \subseteq \Sigma^*$: $U^{-1}V = \{w \mid \exists u \in U, uw \in V\}$

Foundations of Derivation

UNI FREIBURG

Definition: Left quotient

$U, V \subseteq \Sigma^*$: $U^{-1}V = \{w \mid \exists u \in U, uw \in V\}$

Proposition: Closure of regular languages under quotient

$U, V \subseteq \Sigma^*$ regular $\Rightarrow$ $U^{-1}V$ regular

Foundations of Derivation

UNI FREIBURG

Definition: Left quotient

$U, V \subseteq \Sigma^*$: $U^{-1}V = \{w \mid \exists u \in U, uw \in V\}$

Proposition: Closure of regular languages under quotient

$U, V \subseteq \Sigma^*$ regular $\Rightarrow$ $U^{-1}V$ regular

Corrolary: Closure of regular languages under derivation

- Derivation by $a \in \Sigma$ is the special case where $U = \{a\}$
- $V \subseteq \Sigma^*$ regular $\Rightarrow$ $a^{-1}V$ regular

Foundations of Derivation

UNI FREIBURG

Definition: Left quotient

$U, V \subseteq \Sigma^*$: $U^{-1}V = \{w \mid \exists u \in U, uw \in V\}$

Proposition: Closure of regular languages under quotient

$U, V \subseteq \Sigma^*$ regular $\Rightarrow$ $U^{-1}V$ regular

Corrolary: Closure of regular languages under derivation

- Derivation by $a \in \Sigma$ is the special case where $U = \{a\}$
- $V \subseteq \Sigma^*$ regular $\Rightarrow$ $a^{-1}V$ regular

Proposition: Decidable Nullability

$V \subseteq \Sigma^*$ regular $\Rightarrow$ $\varepsilon \in V$ is decidable

Brzozowski Derivatives

UNI FREIBURG

Derivatives of Regular Expressions

$d : \Sigma \times RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(d_r(a)) = a^{-1}\mathcal{L}(r)$

Brzozowski Derivatives

Derivatives of Regular Expressions

$d : \Sigma \times RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(d_r(a)) = a^{-1}\mathcal{L}(r)$

Nullability

$\mathcal{N} : RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(\mathcal{N}(r)) = \{\varepsilon\} \cap \mathcal{L}(r)$

Brzozowski Derivatives

Derivatives of Regular Expressions

$d : \Sigma \times RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(d_r(a)) = a^{-1}\mathcal{L}(r)$

Nullability

$\mathcal{N} : RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(\mathcal{N}(r)) = \{\varepsilon\} \cap \mathcal{L}(r)$

Smart Concatenation

$$r \odot s = \begin{cases} \mathbf{0} & r = \mathbf{0} \text{ or } s = \mathbf{0} \\ r & s = \mathbf{1} \\ s & r = \mathbf{1} \\ r.s & \text{otherwise} \end{cases}$$

Brzozowski Derivatives

Derivatives of Regular Expressions

$d : \Sigma \times RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(d_r(a)) = a^{-1}\mathcal{L}(r)$

Nullability

$\mathcal{N} : RE_\Sigma \to RE_\Sigma$ such that $\mathcal{L}(\mathcal{N}(r)) = \{\varepsilon\} \cap \mathcal{L}(r)$

Smart Concatenation

$$r \odot s = \begin{cases} \mathbf{0} & r = \mathbf{0} \text{ or } s = \mathbf{0} \\ r & s = \mathbf{1} \\ s & r = \mathbf{1} \\ r.s & \text{otherwise} \end{cases}$$

Smart Alternative

$$r \oplus s = \begin{cases} r & s = \mathbf{0} \\ s & r = \mathbf{0} \\ r & r = s \\ r + s & \text{otherwise} \end{cases}$$

Brzozowski Derivatives

Nullability and Derivative

$$\mathcal{N}(\mathbf{0}) = \mathbf{0} \qquad d(a, \mathbf{0}) = \mathbf{0}$$

$$\mathcal{N}(\mathbf{1}) = \mathbf{1} \qquad d(a, \mathbf{1}) = \mathbf{0}$$

$$\mathcal{N}(a) = \mathbf{0} \qquad d(a, b) = \begin{cases} \mathbf{1} & a = b \\ \mathbf{0} & a \neq b \end{cases}$$

$$\mathcal{N}(r + s) = \mathcal{N}(r) \oplus \mathcal{N}(s) \qquad d(a, r + s) = d(a, r) \oplus d(a, s)$$

$$\mathcal{N}(r.s) = \mathcal{N}(r) \odot \mathcal{N}(s) \qquad d(a, r.s) = (d(a, r) \odot s) \oplus (\mathcal{N}(r) \odot d(a, s))$$

$$\mathcal{N}(r^*) = \mathbf{1} \qquad d(a, r^*) = d(a, r) \odot r^*$$

Brzozowski Derivatives

UNI FREIBURG

Iterated Derivative

$d_{\Sigma^*} : RE_\Sigma \to \mathbb{P}(RE_\Sigma)$ the iterated derivative is defined inductively

- $r \in d_{\Sigma^*}(r)$
- if $s \in d_{\Sigma^*}(r)$ and $a \in \Sigma$, then $d(a, s) \in d_{\Sigma^*}(r)$

Brzozowski's Results

UNI FREIBURG

Representation theorem

$$\mathcal{L}(r) = \mathcal{L}(\mathcal{N}(r)) \cup \bigcup_{a \in \Sigma} \{a\} \cdot \mathcal{L}(d_a(r))$$

Brzozowski's Results

UNI FREIBURG

Representation theorem

$$\mathcal{L}(r) = \mathcal{L}(\mathcal{N}(r)) \cup \bigcup_{a \in \Sigma} \{a\} \cdot \mathcal{L}(d_a(r))$$

Finiteness theorem

For each $r \in R_\Sigma$, the set $d_{\Sigma^*}(r)/_\approx$ is finite.
($\approx$ is a similarity relation on R_Σ, i.e., associativity and idempotence of $+$)

Brzozowski's Results

Representation theorem

$$\mathcal{L}(r) = \mathcal{L}(\mathcal{N}(r)) \cup \bigcup_{a \in \Sigma} \{a\} \cdot \mathcal{L}(d_a(r))$$

Finiteness theorem

For each $r \in R_\Sigma$, the set $d_{\Sigma^*}(r)/_\approx$ is finite.
($\approx$ is a similarity relation on R_Σ, i.e., associativity and idempotence of $+$)

Theorem: RE $\rightarrow$ DFA

Let $\mathcal{Q} = d_{\Sigma^*}(r)/_\approx$ and $F = \{s \in \mathcal{Q} \mid \mathcal{N}(s) = \mathbf{1}\}$.
Then $\mathcal{A} = (\mathcal{Q}, \Sigma, d, \{r\}, F)$ is a DFA with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(r)$.

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Partial Derivatives

UNI FREIBURG

- Due to Antimirov (1996)
- Direct translation RE $\rightarrow$ NFA
- Simpler proofs
- Introduces the important notion of linear factors

Linear Factors

Definition

The set of linear factors of a regular expression is defined by a function $\mathrm{LF} : RE_\Sigma \to \mathbb{P}(\Sigma \times RE_\Sigma)$

$$
\begin{aligned}
\mathrm{LF}(\mathbf{0}) &= \emptyset \\
\mathrm{LF}(\mathbf{1}) &= \emptyset \\
\mathrm{LF}(a) &= \{\langle a, \mathbf{1}\rangle\} \\
\mathrm{LF}(r + s) &= \mathrm{LF}(r) \cup \mathrm{LF}(s) \\
\mathrm{LF}(r.s) &= \mathrm{LF}(r) \odot s \cup \mathrm{LF}(s) \odot \mathcal{N}(r) \\
\mathrm{LF}(r^*) &= \mathrm{LF}(r) \odot r^*
\end{aligned}
$$

where

$$
L \odot s = \{\langle a, r \odot s\rangle \mid \langle a, r\rangle \in L, s \neq \mathbf{0}\}
$$

Properties of Linear Factors

UNI FREIBURG

Definition

- $\mathcal{L}(\langle a, r \rangle) = \{a\} \cdot \mathcal{L}(r)$
- $\mathcal{L}(L) = \bigcup \{\{a\} \cdot \mathcal{L}(r) \mid \langle a, r \rangle \in L\}$

Properties of Linear Factors

Definition

- $\mathcal{L}(\langle a, r\rangle) = \{a\} \cdot \mathcal{L}(r)$
- $\mathcal{L}(L) = \bigcup \{\{a\} \cdot \mathcal{L}(r) \mid \langle a, r\rangle \in L\}$

Lemma (Representation)

$\mathcal{L}(r) = L(\mathcal{N}(r)) \cup \mathcal{L}(\mathrm{LF}(r))$

Partial Derivatives

Definition

Let $r \in RE_\Sigma$, $a \in \Sigma$, $w \in \Sigma^*$.

- The partial derivative $\partial_a(r) \subseteq RE_\Sigma$ of r by a is defined by

$$\partial_a(r) = \{r' \mid \langle a, r' \rangle \in \mathrm{LF}(r)\}$$

- The partial derivative $\partial_w(r) \subseteq RE_\Sigma$ of r by w is defined inductively by

$$\partial_\varepsilon(r) = \{r\} \qquad \partial_{aw}(r) = \partial_w(\partial_a(r))$$

- The iterated partial derivative $\partial_{\Sigma^*}(r) \subseteq RE_\Sigma$ is defined by

$$\partial_{\Sigma^*}(r) = \bigcup\{\partial_w(r) \mid w \in \Sigma^*\}$$

Properties of Linear Factors

UNI FREIBURG

Linear factors automatically perform the simplifications needed to obtain finiteness of iterated derivatives.

Definition

Let $S(r) \subseteq RE_\Sigma$ defined by $s \in S(r)$ if $s = s_1 \dots s_n$ such that

- each s_i is a subterm of r
- if $1 \leq i < n$, then s_i *occurs strictly below* s_{i+1} in r

(if $n = 0$ then $s = \mathbf{1}$)

Properties of Linear Factors

Linear factors automatically perform the simplifications needed to obtain finiteness of iterated derivatives.

Definition

Let $S(r) \subseteq RE_\Sigma$ defined by $s \in S(r)$ if $s = s_1 \dots s_n$ such that

- each s_i is a subterm of r
- if $1 \leq i < n$, then s_i *occurs strictly below* s_{i+1} in r

(if $n = 0$ then $s = \mathbf{1}$)

Proposition

$S(r)$ is finite

Finiteness

UNI FREIBURG

Lemma

If $\langle a, s \rangle \in \mathrm{LF}(r)$, then $s \in S(r)$.

Finiteness

UNI FREIBURG

Lemma

If $\langle a, s \rangle \in \text{LF}(r)$, then $s \in S(r)$.

Lemma

If $s \in S(r)$, then $\partial_a(s) \subseteq S(r)$

Finiteness

UNI FREIBURG

Lemma

If $\langle a, s \rangle \in \mathrm{LF}(r)$, then $s \in S(r)$.

Lemma

If $s \in S(r)$, then $\partial_a(s) \subseteq S(r)$

Finiteness Theorem for Partial Derivatives

$\partial_{\Sigma^*}(r) \subseteq S(r)$ and thus finite

Finiteness

UNI FREIBURG

Lemma

If $\langle a, s\rangle \in \mathrm{LF}(r)$, then $s \in S(r)$.

Lemma

If $s \in S(r)$, then $\partial_a(s) \subseteq S(r)$

Finiteness Theorem for Partial Derivatives

$\partial_{\Sigma^*}(r) \subseteq S(r)$ and thus finite

Theorem: RE $\rightarrow$ NFA

Let $\mathcal{Q} = \partial_{\Sigma^*}(r)$ and $F = \{s \in \mathcal{Q} \mid \mathcal{N}(s) = \mathbf{1}\}$.
Then $\mathcal{A} = (\mathcal{Q}, \Sigma, \partial, \{r\}, F)$ is a NFA with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(r)$.

Alternative Finiteness Proof (Broda et al 2011)

UNI FREIBURG

Definition (Approximate Iterated Derivatives)

Specification $\delta^+ : RE_\Sigma \to \mathbb{P}(RE_\Sigma)$

$$
\begin{aligned}
\delta^+(\mathbf{0}) &= \emptyset \\
\delta^+(\mathbf{1}) &= \emptyset \\
\delta^+(a) &= \{\mathbf{1}\} \\
\delta^+(r+s) &= \delta^+(r) \cup \delta^+(s) \\
\delta^+(r.s) &= \delta^+(r) \odot s \cup \delta^+(s) \\
\delta^+(r^*) &= \delta^+(r) \odot r^*
\end{aligned}
$$

Remark

An over approximation of the set of iterated derivatives!

Alternative Finiteness Proof (Broda et al 2011)

UNI FREIBURG

Lemma

For all r, $\delta^+(r)$ is finite

Alternative Finiteness Proof (Broda et al 2011)

UNI FREIBURG

Lemma

For all r, $\delta^+(r)$ is finite

Lemma

- $\partial_a(r) \subseteq \delta^+(r)$
- $s \in \delta^+(r)$ implies $\partial_a(s) \subseteq \delta^+(r)$

Alternative Finiteness Proof (Broda et al 2011)

UNI FREIBURG

Lemma

For all r, $\delta^+(r)$ is finite

Lemma

- $\partial_a(r) \subseteq \delta^+(r)$
- $s \in \delta^+(r)$ implies $\partial_a(s) \subseteq \delta^+(r)$

Finiteness Theorem for Partial Derivatives

$\partial_{\Sigma^*}(r) \subseteq \delta^+(r)$ and thus finite

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Applications

Definition: Regular Expression Equivalence and Containment

Let $r, s \in RE_\Sigma$

- Equivalence $r \sim s$ if $\mathcal{L}(r) = \mathcal{L}(s)$
- Containment $r \lesssim s$ if $\mathcal{L}(r) \subseteq \mathcal{L}(s)$

Applications

Definition: Regular Expression Equivalence and Containment

Let $r, s \in RE_\Sigma$

- Equivalence $r \sim s$ if $\mathcal{L}(r) = \mathcal{L}(s)$
- Containment $r \lesssim s$ if $\mathcal{L}(r) \subseteq \mathcal{L}(s)$

Equivalence and Containment are Decidable

- Standard proof for containment $r \lesssim s$
  - Construct DFA for $\mathcal{L}(r) \setminus \mathcal{L}(s)$
  - Check for emptiness
- Equivalence and containment are equally powerful
  - $r \sim s$ iff $r \lesssim s$ and $s \lesssim r$
  - Observe that $r \lesssim s$ iff $r + s \sim s$

Derivative-based Equivalence Test

UNI FREIBURG

Coinductive Definition of Equivalence

$$\frac{\mathcal{N}(r) = \mathcal{N}(s) \qquad (\forall a \in \Sigma)\ \Sigma\partial_a(r) \sim \Sigma\partial_a(s)}{r \sim s}$$

Derivative-based Equivalence Test

UNI FREIBURG

Coinductive Definition of Equivalence

$$\frac{\mathcal{N}(r) = \mathcal{N}(s) \qquad (\forall a \in \Sigma)\ \Sigma\partial_a(r) \sim \Sigma\partial_a(s)}{r \sim s}$$

Remarks

- Establishes bisimulation between r and s
- Labeled transition system: $r \xrightarrow{a} \Sigma\partial_a(r)$
- Analogous to equivalence of finite automata

Implementation

UNI FREIBURG

Implementation

```
collect R [] = True
collect R ((r, s) : W) =
  if (r,s) in R then
    collect R W
  else if N(r) == N(s) then
    collect ((r, s) : R)
            (map (\a -> (∂a r, ∂a s)) Σ ++ W)
  else False

equiv r s =
  collect [] [(r, s)]
```

Derivative-based Equivalence Test

UNI FREIBURG

Properties of collect

- attempts to construct a bisimulation in R
- returns True iff $r \sim s$ (prove this!)
- termination guaranteed by finiteness of iterated derivatives
- straightforward extension with counterexample on failure

Extension of collect with Counterexample

```
collect R [] = Yes
collect R ((w, r, s) : W) =
  if (r,s) in R then
    collect R W
  else if N(r) == N(s) then
    collect ((r, s) : R)
            (map (\a -> (a:w, ∂a r, ∂a s)) Σ ++ W)
  else No w

equiv r s =
  collect [] [(ε, r, s)]
```

Derivative-Based Containment

Coinductive Definition of Containment (cf. Antimirov)

$$\frac{\mathcal{N}(r) \Rightarrow \mathcal{N}(s) \qquad (\forall a \in \Sigma)\ \Sigma\partial_a(r) \lesssim \Sigma\partial_a(s)}{r \lesssim s}$$

Derivative-Based Containment

Coinductive Definition of Containment (cf. Antimirov)

$$\frac{\mathcal{N}(r) \Rightarrow \mathcal{N}(s) \qquad (\forall a \in \Sigma)\ \Sigma\partial_a(r) \lesssim \Sigma\partial_a(s)}{r \lesssim s}$$

Remarks

- Establishes a simulation between r and s
- Implementation very similar to equivalence
- Finiteness follows for the same reason

Example

Show that $\mathbf{1} + a^* b \lesssim a^* b^*$

$$\dfrac{\mathbf{1} \Rightarrow \mathbf{1} \qquad \dfrac{\mathbf{0} \Rightarrow \mathbf{1} \qquad \dfrac{\text{coind.}}{a^* b \lesssim a^* b^*} \qquad \dfrac{\mathbf{1} \Rightarrow \mathbf{1}}{\mathbf{1} \lesssim b^*}}{a^* b \lesssim a^* b^*} \qquad \dfrac{\mathbf{1} \Rightarrow \mathbf{1}}{\mathbf{1} \lesssim b^*}}{\mathbf{1} + a^* b \lesssim a^* b^*}$$

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Regular Expressions Are Types

Parse Trees Are Values

$$\vdash_r \textsc{Eps} : \mathbf{1} \qquad \vdash_r \textsc{Sym}\ a : a \qquad \vdash_r \textsc{Empty} : r^*$$

$$\frac{\vdash_r p : r \qquad \vdash_r q : s}{\vdash_r \textsc{Seq}\ p\ q : r.s} \qquad \frac{\vdash_r p : r \qquad \vdash_r q : r^*}{\vdash_r \textsc{Cons}\ p\ q : r^*}$$

$$\frac{\vdash_r p : r}{\vdash_r \textsc{Inl}\ p : r + s} \qquad \frac{\vdash_r q : s}{\vdash_r \textsc{Inr}\ q : r + s}$$

Regular Expressions Are Types

Parse Trees Are Values

$$\vdash_r \textsc{Eps} : \mathbf{1} \qquad \vdash_r \textsc{Sym}\ a : a \qquad \vdash_r \textsc{Empty} : r^*$$

$$\frac{\vdash_r p : r \qquad \vdash_r q : s}{\vdash_r \textsc{Seq}\ p\ q : r.s} \qquad \frac{\vdash_r p : r \qquad \vdash_r q : r^*}{\vdash_r \textsc{Cons}\ p\ q : r^*}$$

$$\frac{\vdash_r p : r}{\vdash_r \textsc{Inl}\ p : r + s} \qquad \frac{\vdash_r q : s}{\vdash_r \textsc{Inr}\ q : r + s}$$

Relations Between Types Yield Functions on Parse Trees

- Equivalence Proof $r \sim s \Rightarrow$ Isomorphism $r \leftrightarrow s$
- Containment Proof $r \lesssim s \Rightarrow$ Embedding $r \to s$

Example (cont)

Embedding for $\mathbf{1} + a^* b \lesssim a^* b^*$

- $\textsc{Inl}\ \textsc{Eps} : \mathbf{1} + a^* b \quad \mapsto \quad \textsc{Seq}\ \textsc{Empty}\ \textsc{Empty} : a^* b^*$

Example (cont)

Embedding for $\mathbf{1} + a^* b \lesssim a^* b^*$

- $\textsc{Inl Eps} : \mathbf{1} + a^* b \quad \mapsto \quad \textsc{Seq Empty Empty} : a^* b^*$
- $\textsc{Inr (Seq Empty (Sym } b)) \quad \mapsto$
  $\textsc{Seq Empty (Cons (Sym } b) \textsc{ Empty})$

Example (cont)

Embedding for $\mathbf{1} + a^*b \lesssim a^*b^*$

- $\textsc{Inl Eps} : \mathbf{1} + a^*b \quad \mapsto \quad \textsc{Seq Empty Empty} : a^*b^*$
- $\textsc{Inr (Seq Empty (Sym } b\textsc{))} \quad \mapsto$
  $\textsc{Seq Empty (Cons (Sym } b\textsc{) Empty)}$
- $\textsc{Inr (Seq (Cons (Sym } a\textsc{) Empty) (Sym } b\textsc{))} \quad \mapsto$
  $\textsc{Seq (Cons (Sym } a\textsc{) Empty) (Cons (Sym } b\textsc{) Empty)}$

Example (cont)

Embedding for $\mathbf{1} + a^*b \lesssim a^*b^*$

- $\textsc{Inl Eps} : \mathbf{1} + a^*b \quad \mapsto \quad \textsc{Seq Empty Empty} : a^*b^*$
- $\textsc{Inr (Seq Empty (Sym } b\textsc{))} \quad \mapsto$
  $\textsc{Seq Empty (Cons (Sym } b\textsc{) Empty)}$
- $\textsc{Inr (Seq (Cons (Sym } a\textsc{) Empty) (Sym } b\textsc{))} \quad \mapsto$
  $\textsc{Seq (Cons (Sym } a\textsc{) Empty) (Cons (Sym } b\textsc{) Empty)}$

Goal

Derive functions between parse tree that witness containment

1. Embedding and Projection from Derivation
2. Coinductive Embedding

Step 1: Embedding and Projection from Derivation

UNI FREIBURG

- Derivation of r by a yields a sublanguage of r:

$$\{a\} \cdot [\![d(a, r)]\!] \subseteq [\![r]\!]$$

Step 1: Embedding and Projection from Derivation

UNI FREIBURG

- Derivation of r by a yields a sublanguage of r:

$$\{a\} \cdot [\![d(a, r)]\!] \subseteq [\![r]\!]$$

- Hence, there is an embedding and a projection

$$a.d(a, r) \to r \qquad\qquad r \to a.d(a, r)$$

Step 1: Embedding and Projection from Derivation

UNI FREIBURG

- Derivation of r by a yields a sublanguage of r:

$$\{a\} \cdot [\![d(a, r)]\!] \subseteq [\![r]\!]$$

- Hence, there is an embedding and a projection

$$a.d(a, r) \to r \qquad\qquad r \to a.d(a, r)$$

- Reformulate derivation to construct "embedding" and "projection" functions

$$\vdash_a t : d(a, r) \to r \qquad\qquad \vdash_a^- t : r \to d(a, r)$$

(leaving the a argument implicit)

Auxiliary: Parse Tree for Empty Word

- Every nullable expression has a parse tree for the empty word
- $mkE(\cdot) : \{r \mid \mathcal{N}(r)\} \to r$

$$mkE(\mathbf{1}) = \textsc{Eps} \qquad \frac{\mathcal{N}(r)}{mkE(r + s) = \textsc{Inl}\ mkE(r)}$$

$$\frac{\mathcal{N}(s)}{mkE(r + s) = \textsc{Inr}\ mkE(s)}$$

$$mkE(r.s) = \textsc{Seq}\ mkE(r)\ mkE(s) \qquad mkE(r^*) = \textsc{Empty}$$

Auxiliary: Get First Symbol from Parse Tree

UNI FREIBURG

- $fi(\cdot) : r \to \mathit{Maybe}\ \Sigma$
- $fi(p) = \mathit{Nothing}$: p is parse tree for empty word
- $fi(p) = \mathit{Just}\ a$: $a \in \Sigma$ is first symbol of parsed word

$$fi(\textsc{Eps}) = \mathit{Nothing} \qquad fi(\textsc{Sym}\ a) = \mathit{Just}\ a$$

$$\frac{fi(p) = m}{fi(\textsc{Inl}\ p) = m} \qquad \frac{fi(p) = m}{fi(\textsc{Inr}\ p) = m} \qquad \frac{fi(p) = \mathit{Just}\ a}{fi(\textsc{Seq}\ p\ q) = \mathit{Just}\ a}$$

$$\frac{fi(p) = \mathit{Nothing} \qquad fi(q) = m}{fi(\textsc{Seq}\ p\ q) = m} \qquad fi(\textsc{Empty}) = \mathit{Nothing}$$

$$\frac{fi(p) = \mathit{Just}\ a}{fi(\textsc{Cons}\ p\ q) = \mathit{Just}\ a} \qquad \frac{fi(p) = \mathit{Nothing} \qquad fi(q) = m}{fi(\textsc{Cons}\ p\ q) = m}$$

Embedding and Projection from Derivation

$d(a, \mathbf{0}) = \mathbf{0}$

$$\vdash_a \lambda y.abort\ y : \mathbf{0} \to \mathbf{0} \qquad \vdash_a^- \lambda y.abort\ y : \mathbf{0} \to \mathbf{0}$$

$d(a, \mathbf{1}) = \mathbf{0}$

$$\vdash_a \lambda y.abort\ y : \mathbf{0} \to \mathbf{1} \qquad \vdash_a^- \lambda() : \mathbf{1} \to \mathbf{0}$$

$d(a, b) = \text{if } a = b \text{ then } \mathbf{1} \text{ else } \mathbf{0}$

$$\vdash_a \lambda y.\textsc{Sym}\ a : \mathbf{1} \to a \qquad \vdash_a \lambda y.abort\ y : \mathbf{0} \to b$$

$$\vdash_a^- \lambda(\textsc{Sym}\ a).\textsc{Eps} : a \to \mathbf{1} \qquad \vdash_a^- \lambda() : b \to \mathbf{0}$$

Embedding and Projection from Derivation (2)

$d(a, r + s) = d(a, r) + d(a, s)$

$$\frac{\vdash_a t_r : r' \to r \quad \vdash_a t_s : s' \to s}{\begin{array}{l}\vdash_a \lambda y.\text{case } y \text{ of } \textsc{Inl}\ p \to \textsc{Inl}(t_r\ p) \\ \qquad\qquad\qquad\quad \textsc{Inr}\ q \to \textsc{Inr}(t_s\ q) \\ \qquad\qquad\qquad\quad : r' + s' \to r + s\end{array}}$$

$$\frac{\vdash_a^- t_r : r \to r' \quad \vdash_a^- t_s : s \to s'}{\begin{array}{l}\vdash_a^- \lambda y.\text{case } y \text{ of } \textsc{Inl}\ p \to \textsc{Inl}(t_r\ p) \\ \qquad\qquad\qquad\quad \textsc{Inr}\ q \to \textsc{Inr}(t_s\ q) \\ \qquad\qquad\qquad\quad : r + s \to r' + s'\end{array}}$$

Embedding and Projection from Derivation (3)

$$d(a, r.s) = d(a, r).s + \mathcal{N}(r).d(a, s)$$

$$\frac{\vdash_a t_r : r' \to r \quad \vdash_a t_s : s' \to s}{\begin{array}{l}\vdash_a \lambda y.\text{case } y \text{ of } \textsc{Inl}\ (\textsc{Seq}\ p\ q) \to \textsc{Seq}\ (t_r\ p)\ q \\ \qquad\qquad \textsc{Inr}\ (\textsc{Seq}\ \textsc{Eps}\ q) \to \textsc{Seq}\ mkE(r)\ (t_s\ q) \\ \qquad\qquad : r'.s + \mathcal{N}(r).s' \to r.s\end{array}}$$

$$\frac{\vdash_a^- t_r : r \to r' \quad \vdash_a^- t_s : s \to s'}{\begin{array}{l}\vdash_a^- \lambda(\textsc{Seq}\ p\ q).\text{case } fi(p) \text{ of } \mathit{Just}\ a \to \textsc{Inl}\ (\textsc{Seq}\ (t_r\ p)\ q) \\ \qquad\qquad \mathit{Nothing} \to \textsc{Inr}\ (\textsc{Seq}\ \textsc{Eps}\ t_s\ q) \\ \qquad\qquad : r.s \to r'.s + \mathcal{N}(r).s'\end{array}}$$

Embedding and Projection from Derivation (4)

$d(a, r^*) = d(a, r).r^*$

$$\frac{\vdash_a t_r : r' \to r}{\vdash_a \lambda(\textsc{Seq}\ p\ q).\textsc{Cons}\ (t_r\ p)\ q : r'.r^* \to r^*}$$

$$\frac{\vdash_a^- t_r : r \to r'}{\vdash_a^- \lambda y.\text{case } y \text{ of } \textsc{Cons}\ p\ q \to \textsc{Seq}\ (t_r\ p)\ q : r^* \to r'.r^*}$$

Assumes that parse trees for r^* are reduced:

- no unproductive iteration steps, even if $\mathcal{N}(r)$
- $\textsc{Cons}\ p\ q$ implies that $fi(p) = \mathit{Just}\ a$

Example (cont'd)

Embedding from Derivations

- $d(a, a^*.b^*) = (\mathbf{1}.a^*).b^* + \mathbf{1}.\mathbf{0} = (\mathbf{1}.a^*).b^*$
  - $u_a =$
    $\lambda(\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ p)\ q).\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ (\textsc{Cons}\ (\textsc{Sym}\ a)\ p))\ q$
  - $t_a^- =$
    $\lambda(\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ (\textsc{Cons}\ (\textsc{Sym}\ a)\ p))\ q).(\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ p)\ q)$
- $d(b, a^*.b^*) = \mathbf{0}.b^* + \mathbf{1}.b^* = \mathbf{1}.b^*$
  - $u_b = \lambda(\textsc{Seq}\ \textsc{Eps}\ q).\textsc{Seq}\ (\textsc{Cons}\ (\textsc{Sym}\ b)\ \textsc{Empty})\ q$
  - $t_b^- = \lambda(\textsc{Seq}\ (\textsc{Cons}\ (\textsc{Sym}\ b)\ \textsc{Empty})\ q).\textsc{Seq}\ \textsc{Eps}\ q$

Step 2: Construtive Embeddings

Coinductive Definition of Embedding

$$\frac{\mathcal{N}(s)}{\lambda \textsc{Eps}.mkE(s) : \mathbf{1} \lesssim s}$$

$$\frac{\mathcal{N}(s') \qquad \vdash_a t : s' \to s}{\lambda(\textsc{Sym}\ a).t(mkE(s')) : a \lesssim s}$$

$$\frac{t_1 : r_1 \lesssim s \qquad t_2 : r_2 \lesssim s}{\lambda y.\text{case } y \text{ of } \textsc{Inl}\ p \to \textsc{Inl}\ (t_1\ p) \mid \textsc{Inr}\ q \to \textsc{Inr}\ (t_2\ q) : r_1 + r_2 \lesssim s}$$

Constructing the Embedding (2)

Coinductive Definition of Embedding (2)

$$\frac{\begin{array}{cc} \vdash_a^- t_a^- : r_1 \to r_1' & \vdash_a u_a : s' \to s \\ t_1 : r_1'.r_2 \lesssim s' & t_2 : r_2 \lesssim s \end{array}}{\begin{array}{c} \lambda(\textsc{Seq}\ p_1\ p_2). \\ \text{if } \mathit{fi}(p_1) = \mathit{Nothing} \text{ then } (t_2\ p_2) \\ \text{else } \mathit{fi}(p_1) = \mathit{Just}\ a; u_a(t_1(\textsc{Seq}\ (t_a^-\ p_1)\ p_2)) \\ : r_1.r_2 \lesssim s \end{array}}$$

Constructing the Embedding (3)

Coinductive Definition of Embedding (3)

$$\frac{\vdash_a^- t_a^- : r_1 \to r_1' \qquad \vdash_a u_a : s' \to s \qquad t_1 : r'.r^* \lesssim s' \qquad t_2 : r^* \lesssim s}{\begin{array}{c}\lambda y.\text{case } y \text{ of} \\ \textsc{Empty} \to \mathit{mkE}(s) \\ \textsc{Cons}\ p\ q \to \text{if } \mathit{fi}(p) = \mathit{Nothing} \text{ then } (t_2\ q) \\ \text{else } \mathit{fi}(p) = \mathit{Just}\ a; u_a(t_1(\textsc{Seq}\ (t_a^-\ p)\ q)) \\ : r^* \lesssim s\end{array}}$$

Example

Embedding for $\mathbf{1} + a^*b \lesssim a^*b^*$

- $t = \lambda y.\text{case } y \text{ of } \textsc{Inl}\ p \to \textsc{Inl}\ (t_1\ p) \mid \textsc{Inr}\ q \to \textsc{Inr}\ (t_2\ q)$
- $t_1 = \lambda \textsc{Eps}.mkE(a^*b^*) : \mathbf{1} \lesssim a^*b^*$
- $t_2 = \lambda(\textsc{Seq}\ p_1\ p_2).\text{case } fi(p_1) \text{ of}$
  $\quad Nothing \to (t_{22}\ p_2)$
  $\quad Just\ a \to u_a(t_{21a}(\textsc{Seq}\ (t_a^-\ p_1)\ p_2))$
  $\quad Just\ b \to u_b(t_{21b}(\textsc{Seq}\ (t_b^-\ p_1)\ p_2)) : a^*.b \lesssim a^*.b^*$
- $t_{21a} = \lambda(\textsc{Seq}\ p\ (\textsc{Sym}\ b)).\textsc{Seq}\ p\ \textsc{Cons}\ (\textsc{Sym}\ b)\ \textsc{Empty} : (\mathbf{1}.a^*).b \lesssim (\mathbf{1}.a^*).b^*$
- $t_{21b} = \lambda \textsc{Eps}.mkE(\mathbf{1}.b^*) : \mathbf{1} \lesssim \mathbf{1}.b^*$
- $t_{22} = \lambda(\textsc{Sym}\ b).t_{22b}(mkE(\mathbf{1}.b^*)) : b \lesssim a^*.b^*$
- $\vdash_b t_{22b} = \lambda(\textsc{Seq}\ \textsc{Eps}\ p).\textsc{Seq}\ \textsc{Empty}\ (\textsc{Cons}\ (\textsc{Sym}\ b)\ p) : \mathbf{1}.b^* \to a^*.b^*$

Embedding (continued)

Embeddings From Derivation

- $\vdash_a u_a =$
  $\lambda(\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ p)\ q).\textsc{Seq}\ (\textsc{Cons}\ (\textsc{Sym}\ a)\ p)\ q$
  $: (\mathbf{1}.a^*).b^* \to a^*.b^*$
- $\vdash_a^- t_a^- =$
  $\lambda(\textsc{Seq}\ (\textsc{Cons}\ (\textsc{Sym}\ a)\ p)\ q).\textsc{Seq}\ (\textsc{Seq}\ \textsc{Eps}\ p)\ q$
  $: a^*.b^* \to (\mathbf{1}.a^*).b^*$
- $\vdash_b u_b = \lambda(\textsc{Seq}\ \textsc{Eps}\ q).\textsc{Seq}\ \textsc{Empty}\ (\textsc{Cons}\ (\textsc{Sym}\ b)\ q)$
  $: \mathbf{1}.b^* \to a^*.b^*$
- $\vdash_b^- t_b^- = \lambda(\textsc{Seq}\ \textsc{Empty}\ (\textsc{Cons}\ (\textsc{Sym}\ b)\ q)).\textsc{Seq}\ \textsc{Eps}\ q$
  $: a^*.b^* \to \mathbf{1}.b^*$

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Context-Free Grammars

UNI FREIBURG

Context-Free Grammar (CFG)

$G = (N, \Sigma, P, S)$

- N a finite set of nonterminal symbols
- Σ an alphabet (terminal symbols)
- $P : N \to \mathbb{P}_{fin}(N \cup \Sigma)^*$ a (finite) set of productions
- $S \in N$ distinguished start symbol

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Derivative of Nonterminal A by a

Algorithm

Let $A \in N$ and $a \in \Sigma$. Let $W = \{A\}$.

- Choose and remove A from W.
- Exhaustively:
  - If $A \to \alpha a \beta \in P$ and $\mathcal{N}(\alpha)$, then add $A_a \to \beta \in P$.
  - If $A \to \alpha B \beta \in P$ and $\mathcal{N}(\alpha)$, then add $A_a \to B_a \beta \in P$; if $B_a \notin N$, then $W = W \cup \{B\}$.
- Repeat until $W = \emptyset$

Derivative of Nonterminal A by a

Algorithm

Let $A \in N$ and $a \in \Sigma$. Let $W = \{A\}$.

- Choose and remove A from W.
- Exhaustively:
  - If $A \to \alpha a \beta \in P$ and $\mathcal{N}(\alpha)$, then add $A_a \to \beta \in P$.
  - If $A \to \alpha B \beta \in P$ and $\mathcal{N}(\alpha)$, then add $A_a \to B_a \beta \in P$; if $B_a \notin N$, then $W = W \cup \{B\}$.
- Repeat until $W = \emptyset$

Remark

- Terminates with $L(A_a) = a^{-1}L(A)$ (prove this!)
- Iterated derivative does not always terminate
- Termination can be improved by mapping A_a to an existing nonterminal with equivalent productions

Pragmatic Parsing

UNI FREIBURG

Algorithm

To parse w compute S_w and test $\mathcal{N}(S_w)$.

Remark

- Effective procedure
- Basis for recent work on parsing with derivatives by Adams, Darais, Hollenbeck, Might, Spievak
- Space leak for many grammars

Pragmatic Parsing Example

CFG for a^*b (right recursive)

- $S \to b$, $S \to aS$
- Derivatives: $S_a \to S$, $S_b \to \varepsilon$
- Iterated derivation terminates: $S_{aa} \approx S_a$, $S_{ab} \approx S_b$

Pragmatic Parsing Example

CFG for a^*b (right recursive)

- $S \to b$, $S \to aS$
- Derivatives: $S_a \to S$, $S_b \to \varepsilon$
- Iterated derivation terminates: $S_{aa} \approx S_a$, $S_{ab} \approx S_b$

CFG for a^*b (left recursive)

- $S \to Ab$, $A \to \varepsilon$, $A \to Aa$
- Derivatives: $S_a \to A_a b$, $A_a \to A_a a$, $A_a \to \varepsilon$, $S_b \to \varepsilon$
- Iterated derivation terminates: $A_a \approx A \Rightarrow S_a \approx S$

Pragmatic Parsing: Context-Free Example

UNI FREIBURG

CFG for $a^n b^n$ (a linear language)

- $S \to \varepsilon$, $S \to aSb$
- Derivatives: $S_a \to Sb$, nothing for S_b
- Iterated derivatives: $S_{aa} \to S_a b$, $S_{ab} \to \varepsilon$
- Iterated derivatives: $S_{aaa} \to S_{aa} b$, $S_{aab} \to S_{ab} b$
- etc

A Context-Free Derivative Construction
[Winter, Bonsangue, Rutten 2011]

UNI FREIBURG

- Every CFL has a grammar in Greibach Normal Form (GNF)
- I.e. every production has the form $A_0 \to aA_1 \dots A_n$

$\Rightarrow$ Nonterminal A_0 has derivative $A_1 \dots A_n$

- Drawback: Requires grammar in GNF
- Every CFG of size n can be converted to a CFG in GNF of size $O(n^4)$

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

ω-Languages

- Let ω denote the set of natural numbers
- Σ^ω
  - the set of maps from $\omega \to \Sigma$
  - the set of infinite sequences over Σ
  - the set of infinite Σ-words
- $\mathbb{P}(\Sigma^\omega)$ the set of ω-languages over Σ
- concatenation only defined for $u \in \Sigma^*$ and $v \in \Sigma^\omega$

Operations on ω Languages

UNI FREIBURG

Product

Let $U \subseteq \Sigma^*$ and $V \subseteq \Sigma^\omega$

$$U \cdot V = \{u \cdot v \mid u \in U, v \in V\}$$

Iteration

Let $U \subseteq \Sigma^*$ with $\varepsilon \notin U$

$$U^\omega = U \cdot U^\omega$$

ω-Regular Languages

ω-regular expressions

$$R_\Sigma^\omega \ni \alpha, \beta ::= \mathbf{0} \mid \alpha + \beta \mid r.\alpha \mid s^\omega$$

where $r, s \in R_\Sigma$ and $\mathcal{N}(s) = \mathbf{0}$.

ω-Regular Languages

ω-regular expressions

$$R_\Sigma^\omega \ni \alpha, \beta ::= \mathbf{0} \mid \alpha + \beta \mid r.\alpha \mid s^\omega$$

where $r, s \in R_\Sigma$ and $\mathcal{N}(s) = \mathbf{0}$.

Semantics

$$\begin{aligned} [\![\mathbf{0}]\!] &= \emptyset \\ [\![\alpha + \beta]\!] &= [\![\alpha]\!] \cup [\![\beta]\!] \\ [\![r.\alpha]\!] &= [\![r]\!] \cdot [\![\alpha]\!] \\ [\![s^\omega]\!] &= [\![s]\!]^\omega \end{aligned}$$

ω-Regular Languages

ω-regular expressions

$$R_\Sigma^\omega \ni \alpha, \beta ::= \mathbf{0} \mid \alpha + \beta \mid r.\alpha \mid s^\omega$$

where $r, s \in R_\Sigma$ and $\mathcal{N}(s) = \mathbf{0}$.

Semantics

$$\begin{aligned} [\![\mathbf{0}]\!] &= \emptyset \\ [\![\alpha + \beta]\!] &= [\![\alpha]\!] \cup [\![\beta]\!] \\ [\![r.\alpha]\!] &= [\![r]\!] \cdot [\![\alpha]\!] \\ [\![s^\omega]\!] &= [\![s]\!]^\omega \end{aligned}$$

Definition

An ω-language V is regular if there is an ω-regular expression α such that $V = [\![\alpha]\!]$.

ω Automata

A nondeterministic Büchi automaton (NBA)

$\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ consists of

- Q finite set of states
- Σ alphabet
- $I \subseteq Q$ initial states
- $\delta : Q \times \Sigma \to \mathbb{P}Q$ transition function
- $F \subseteq Q$ accepting states

$\mathcal{M}$ is deterministic (DFA) if $|I| = 1$ and, for all q, a, $|\delta(q, a)| = 1$.

Runs and Languages

UNI FREIBURG

Run

A run of $\mathcal{M}$ on $a_1, a_2 \cdots \in \Sigma^\omega$ is a sequence $q_0, q_1, \cdots \in Q^\omega$ s.t.

- $q_0 \in I$,
- for all $i \in \omega$, $q_{i+1} \in \delta(q, a_i)$.

The run is (Büchi-) accepting if $q_i \in F$ for infinitely many $i \in \omega$.

Language of Automaton $\mathcal{M}$

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^\omega \mid \mathcal{M} \text{ has an accepting run on } w\}$$

Theorem

An ω-language is regular iff it is accepted by an NBA.

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Derivatives (Attempt #1)

$$
\begin{aligned}
d_a(\mathbf{0}) &= \mathbf{0} \\
d_a(\alpha + \beta) &= d_a(\alpha) \oplus d_a(\beta) \\
d_a(r.\alpha) &= (d_a(r) \odot \alpha) \oplus (d_a(\alpha) \odot \mathcal{N}(r)) \\
d_a(s^\omega) &= d_a(s) \odot s^\omega
\end{aligned}
$$

where $\oplus, \odot$ are "clever" constructors that normalize modulo $\approx$.

Properties of Brzozowski-style ω-Derivatives

UNI FREIBURG

Quotient

$$\mathcal{L}^\omega(d_a(\alpha)) = a^{-1}\mathcal{L}^\omega(\alpha)$$

Representation

$$\mathcal{L}^\omega(\alpha) = \bigcup_{a\in\Sigma} \{a\} \cdot \mathcal{L}^\omega(d_a(\alpha))$$

Finiteness

For each $r \in R^\omega_\Sigma$, the set $d_{\Sigma^*}(r)/_\approx$ is finite.
($\approx$ is a similarity relation on R^ω_Σ)

Properties of Brzozowski-style ω-Derivatives

UNI FREIBURG

Quotient

$$\mathcal{L}^\omega(d_a(\alpha)) = a^{-1}\mathcal{L}^\omega(\alpha)$$

Representation

$$\mathcal{L}^\omega(\alpha) = \bigcup_{a\in\Sigma} \{a\} \cdot \mathcal{L}^\omega(d_a(\alpha))$$

Finiteness

For each $r \in R^\omega_\Sigma$, the set $d_{\Sigma^*}(r)/_\approx$ is finite.
($\approx$ is a similarity relation on R^ω_Σ)

ω-RE $\rightarrow$???

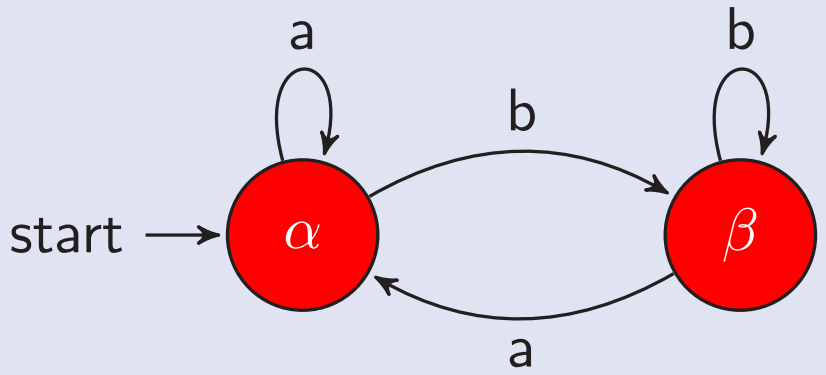
The analogous automaton construction does not work out because it yields deterministic automata.

Counterexample

Example for DBA $\subsetneq$ NBA: finitely many 'a's

$$\alpha = (a + b)^*.b^\omega$$

Wrong automaton constructed by naïve extension



- $\alpha = (a + b)^*.b^\omega$
- $\beta = \alpha + b^\omega$
- deterministic!
- accepting states?

Plan

UNI FREIBURG

1. Derivatives for Regular Languages
   - Brzozowski Derivatives
   - Antimirov's Partial Derivatives
   - Applications: Equivalence and Containment
   - Computational Interpretation
2. Derivatives for Context-Free Languages
   - Pragmatic Parsing
   - Deriving Automata
   - Computational Interpretation
3. Derivation for ω-Regular Languages
   - Attempt #1: Brzozowski-style Derivatives
   - Attempt #2: Extending Partial Derivatives

Insight

UNI FREIBURG

Construction for nondeterministic automaton needed!

Insight

UNI FREIBURG

Construction for nondeterministic automaton needed!

Alternative

Construction based on Antimirov's partial derivatives

Refresher: Antimirov's Algorithm for Partial Derivatives

UNI FREIBURG

Step 1: Compute the set of *Linear Factors* $\mathrm{LF}(r)$

$$
\begin{array}{lcl@{\qquad}lcl}
\mathrm{LF}(\mathbf{0}) & = & \{\} & \mathrm{LF}(r.s) & = & \mathrm{LF}(r) \odot s \cup \mathrm{LF}(s) \odot \mathcal{N}(r) \\
\mathrm{LF}(\mathbf{1}) & = & \{\} & \mathrm{LF}(r + s) & = & \mathrm{LF}(r) \cup \mathrm{LF}(s) \\
\mathrm{LF}(a) & = & \{\langle a, \mathbf{1}\rangle\} & \mathrm{LF}(r^*) & = & \mathrm{LF}(r) \odot r^*
\end{array}
$$

Refresher: Antimirov's Algorithm for Partial Derivatives

UNI FREIBURG

Step 1: Compute the set of *Linear Factors* $\mathrm{LF}(r)$

$$
\begin{array}{lcl@{\qquad}lcl}
\mathrm{LF}(\mathbf{0}) & = & \{\} & \mathrm{LF}(r.s) & = & \mathrm{LF}(r) \odot s \cup \mathrm{LF}(s) \odot \mathcal{N}(r) \\
\mathrm{LF}(\mathbf{1}) & = & \{\} & \mathrm{LF}(r+s) & = & \mathrm{LF}(r) \cup \mathrm{LF}(s) \\
\mathrm{LF}(a) & = & \{\langle a, \mathbf{1}\rangle\} & \mathrm{LF}(r^*) & = & \mathrm{LF}(r) \odot r^*
\end{array}
$$

Step 2: Collect linear factors for each input symbol

$$\partial_a(r) = \{r' \mid \langle a, r'\rangle \in \mathrm{LF}(r)\}$$

Extending to ω-Regular Expressions

Step 1: ω-Linear Factors

$\mathrm{LF}^\omega : R^\omega_\Sigma \to \wp(\Sigma \times R^\omega_\Sigma \times \{0,1\})$

$$
\begin{array}{lcl}
\mathrm{LF}^\omega(\mathbf{0}) & = & \emptyset \\
\mathrm{LF}^\omega(\alpha + \beta) & = & \mathrm{LF}^\omega(\alpha) \cup \mathrm{LF}^\omega(\beta) \\
\mathrm{LF}^\omega(r.\alpha) & = & \mathrm{LF}(r) \odot \alpha \times \{0\} \cup \mathrm{LF}^\omega(\alpha) \odot \mathcal{N}(r) \\
\mathrm{LF}^\omega(s^\omega) & = & \mathrm{LF}(s) \odot s^\omega \times \{1\}
\end{array}
$$

Extending to ω-Regular Expressions

Step 1: ω-Linear Factors

$\mathrm{LF}^\omega : R^\omega_\Sigma \to \wp(\Sigma \times R^\omega_\Sigma \times \{0,1\})$

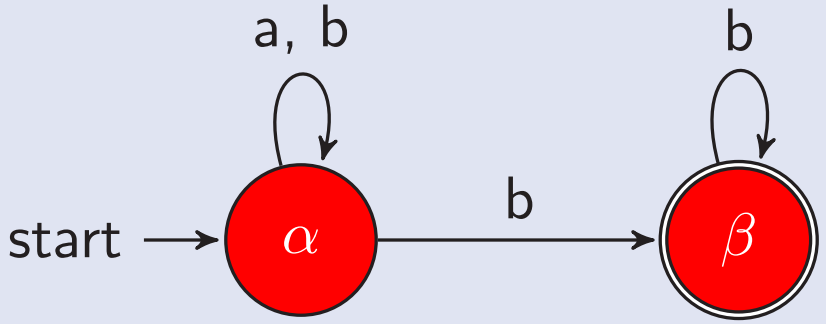
$$
\begin{array}{lcl}
\mathrm{LF}^\omega(\mathbf{0}) & = & \emptyset \\
\mathrm{LF}^\omega(\alpha + \beta) & = & \mathrm{LF}^\omega(\alpha) \cup \mathrm{LF}^\omega(\beta) \\
\mathrm{LF}^\omega(r.\alpha) & = & \mathrm{LF}(r) \odot \alpha \times \{0\} \cup \mathrm{LF}^\omega(\alpha) \odot \mathcal{N}(r) \\
\mathrm{LF}^\omega(s^\omega) & = & \mathrm{LF}(s) \odot s^\omega \times \{1\}
\end{array}
$$

Step 2: Collect ω-linear factors (as before)

$$\partial^\omega_a(r) = \{r' \mid \langle a, r', u\rangle \in \mathrm{LF}^\omega(r)\}$$

Success?

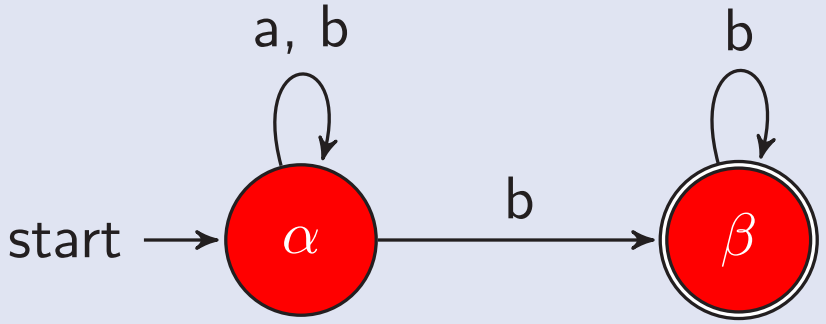
NFA constructed by naïve extension of Antimirov's algorithm



- $\alpha = (a + b)^*.b^\omega$
- $\beta = b^\omega$
- nondeterministic!
- Choose $F = \{\beta\}$

Success?

NFA constructed by naïve extension of Antimirov's algorithm



- $\alpha = (a + b)^*.b^\omega$
- $\beta = b^\omega$
- nondeterministic!
- Choose $F = \{\beta\}$

Assessment

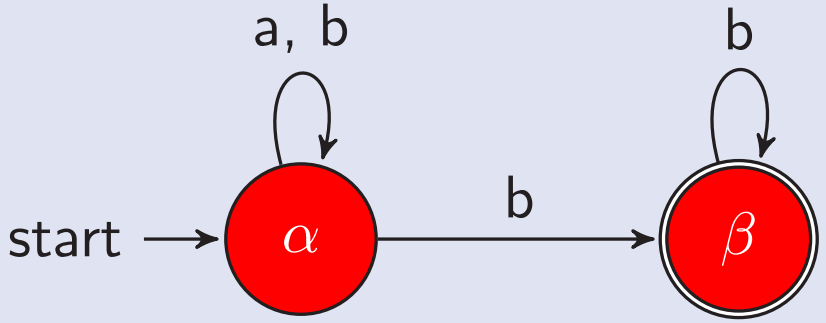
- Conjecture: states of the form s^ω are accepting
- Works for this particular expression, but not in general!

Counterexample

Another expression for the same language ...

$$\begin{aligned} \alpha &= (a+b)^*.(b.b^*)^\omega \\ \beta &= b^*.(b.b^*)^\omega \end{aligned}$$

NFA constructed by naïve extension of Antimirov's algorithm

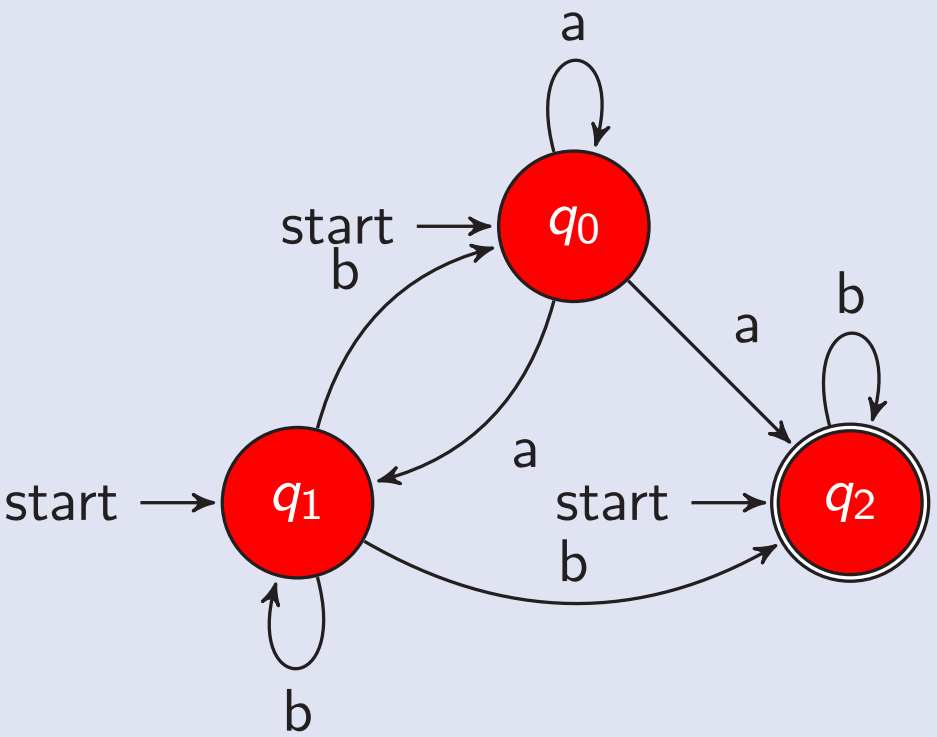


- nondeterministic!
- Choose $F = \{\beta\}$
- but why?

A Working Approach

$$\alpha = (a + b)^*.b^\omega$$

Automaton Constructed from ω-Linear Factors



$$\mathrm{LF}^\omega(\alpha) =$$

$$\begin{array}{ll} \langle a, (a + b)^*.b^\omega, 0\rangle & q_0 \\ \langle b, (a + b)^*.b^\omega, 0\rangle & q_1 \\ \langle b, b^\omega, 1\rangle\} & q_2 \end{array}$$

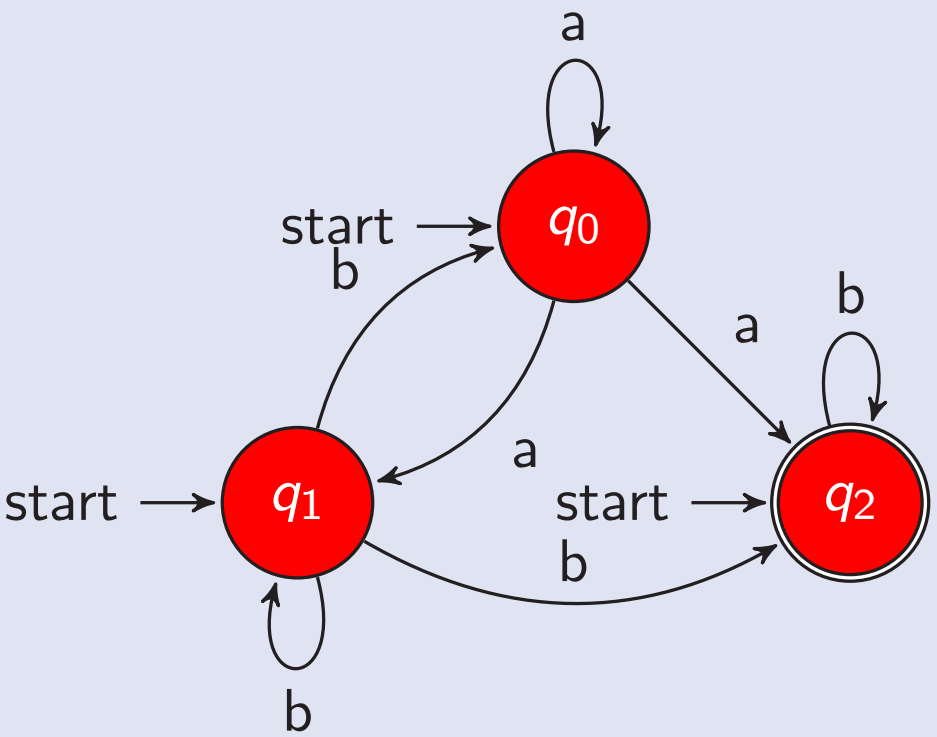
$$Q = I = \{q_0, q_1, q_2\}$$
$$F = \{\langle b, b^\omega, 1\rangle\}$$

Correct,
though not minimal

A Working Approach

$$\alpha = (a + b)^*.b^\omega$$

Automaton Constructed from ω-Linear Factors



$$\mathrm{LF}^\omega(\alpha) =$$

$$\begin{array}{ll} \langle a, (a + b)^*.b^\omega, 0\rangle & q_0 \\ \langle b, (a + b)^*.b^\omega, 0\rangle & q_1 \\ \langle b, b^\omega, 1\rangle\} & q_2 \end{array}$$

$$Q = I = \{q_0, q_1, q_2\}$$
$$F = \{\langle b, b^\omega, 1\rangle\}$$

Correct,
though not minimal

Same automaton for $\alpha = (a + b)^*.(b.b^*)^\omega$

Main Result

UNI FREIBURG

Definition

For $\alpha \in R_\Sigma^\omega$ define the NBA $\mathcal{B}(\alpha) = (Q, \Sigma, \delta, I, F)$ by

- $Q = \mathcal{Q}(\alpha)$ (closure of LF^ω over symbols);
- $I = \mathrm{LF}^\omega(\alpha)$;
- $F = \{\langle a, \beta, g\rangle \in Q \mid g = 1\}$; and
- $\delta(f, a) = \partial_a^\omega(f)$.

Theorem

For $\alpha \in R_\Sigma^\omega$: $\mathcal{L}^\omega(\alpha) = \mathcal{L}^\omega(\mathcal{B}(\alpha))$.

Conclusions

UNI FREIBURG

- Derivatives for ω-regular expressions lose too much structure
- Linear factors retain enough information to construct a correct automaton
- Future work
  - identify equivalent states during construction
  - inclusion test?


Michael D. Adams, Celeste Hollenbeck, and Matthew Might.
On the complexity and performance of parsing with derivatives.
In *PLDI*, pages 224–236. ACM, 2016.


Valentin M. Antimirov.
Rewriting regular inequalities.
In *Proc. of FCT'95*, volume 965 of *LNCS*, pages 116–125. Springer, 1995.


Valentin M. Antimirov.
Partial derivatives of regular expressions and finite automaton constructions.
Theoretical Computer Science, 155(2):291–319, 1996.


Janusz A. Brzozowski.
Derivatives of regular expressions.
J. ACM, 11(4):481–494, 1964.


Matthew Might, David Darais, and Daniel Spiewak.
Parsing with derivatives: a functional pearl.
In *Proc. of ICFP'11*, pages 189–195. ACM, 2011.


Martin Sulzmann and Peter Thiemann.
A computational interpretation of context-free expressions.
In *APLAS*, volume 10695 of *Lecture Notes in Computer Science*, pages 387–405. Springer, 2017.


Peter Thiemann.
Partial derivatives for context-free languages - from μ-regular expressions to pushdown automata.
In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 248–264, 2017.


Peter Thiemann and Martin Sulzmann.
From ω -regular expressions to büchi automata via partial derivatives.
In *Proc. of LATA'15*, volume 8977 of *LNCS*, pages 287–298. Springer, 2015.


Joost Winter, Marcello M. Bonsangue, and Jan J. M. M. Rutten.
Context-free languages, coalgebraically.
In *CALCO*, volume 6859 of *LNCS*, pages 359–376. Springer, 2011.