derivatives of
regular expressions

ernest ng & laura zielinski

Regular Expressions

```
Inductive re :=
| Void : re
| Epsilon : re
| Atom : char → re
| Union : re → re → re
| Concat : re → re → re
| Star : re → re.
```

$$a^* + b \quad \leftrightarrow \quad \texttt{Union (Star (Atom a)) (Atom b)}$$

Matching a String (list char)

```
Inductive matches : re → string → Prop :=
| matches_epsilon : matches Epsilon []
| matches_atom a : matches (Atom a) [a]
| matches_union_l r1 r2 s :
    matches r1 s → matches (Union r1 r2) s
...
```

```
matches (Union (Star (Atom a)) (Atom b)) [‘a’; ‘a’; ‘a’] ✅
matches (Union (Star (Atom a)) (Atom b)) [‘b’; ‘a’] ❌
```

The Brzozowski Derivative

```
Fixpoint b_der (r : re) (c : char) : re
```

$\longleftarrow \delta_c(r)$

...returns a regex which matches “the rest of the string” after matching c, e.g.

$$\delta_c(c \cdot a \cdot t) = a \cdot t$$

$$\delta_b(a^* \cdot b + b) = \varepsilon$$

Matching Using the Brzozowski Derivative

$$\begin{aligned}\delta_{abb}(a \cdot b^*) &= \delta_{bb}(\delta_a(a \cdot b^*)) \\ &= \delta_{bb}(b^*) = \delta_b(\delta_b(b^*)) \\ &= \delta_b(b^*) = b^*\end{aligned}$$

$$b^* \text{ matches } \text{``''} \Rightarrow (a \cdot b^*) \text{ matches } \text{``}abb\text{''}$$

Matching Using the Brzozowski Derivative

```
(** isEmpty returns true iff r matches the empty string *)

Definition b_matches (r : re) (s : string) : bool :=
    isEmpty (fold_left b_der s r).



Lemma b_matches_matches (r : re) (s : string) :
    b_matches r s = true ⟷ matches r s.
```

Blowing Up

$$\delta_{aaa}(a^*(a+b)(a+b)(a+b))$$

$$= \delta_{aa}(a^*(a+b)(a+b)(a+b) + (a+b)(a+b))$$

$$= \delta_{a}(a^*(a+b)^3 + (a+b)^2 + (a+b))$$

$$= (a^*(a+b)^3 + (a+b)^2 + (a+b) + \varepsilon)$$

The Antimirov Derivative

```
Fixpoint a_der (r : re) (c : char) : gset re
```
⟵ $\alpha_c(r)$

...returns a **set** of regexes, one of which matches "the rest of the string" after matching c, e.g.

$$\alpha_c(c \cdot a \cdot t + c \cdot o \cdot w) = \{a \cdot t, o \cdot w\}$$

$$\alpha_a(a^* \cdot (a+b)^3) = \{a^* \cdot (a+b)^3, (a+b)^2)\}$$

↑ "partial derivative"

Matching Using the Antimirov Derivative

```
(** a_der_set applies a_der pointwise to elements in a set *)
(** nullable returns true iff some regex in the set matches the
empty string *)

Definition a_matches (r : re) (s : string) : bool :=
  nullable (fold_left a_der_set s {[ r ]}).
```

Antimirov 🤝 Brzozowski

```
Theorem a_b_matches : forall (r : re) (s : string),
  a_matches r s ⟷ b_matches r s.
```

Finitude

```
author = {Brzozowski, Janusz A.},
title = {Derivatives of Regular Expressions},
year = {1964}
```

THEOREM 5.2. *Every regular expression has only a finite number of dissimilar derivatives.*

PROOF. The proof is given in Appendix II. As a consequence of this result, *a state diagram can be constructed even if only similarity among the derivatives is recognized.*

Finitude

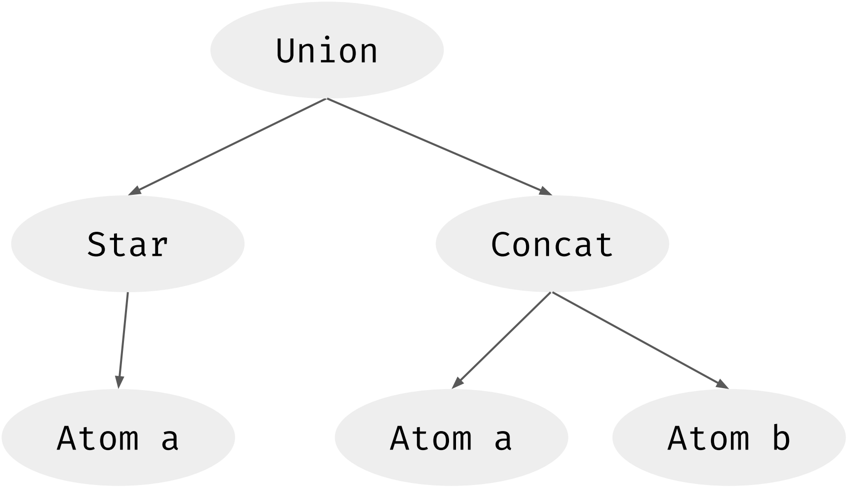
```
(** All possible partial derivatives (maybe more) *)
Fixpoint A_der (r : re) : gset re :=
  match r with
  | Void => {[ Void ]}
...

(** gsets are finite *)
Theorem a_finite : forall (r : re) (s : string),
    a_der_str r s ⊆ A_der r.
```

Viewing Regexes as Trees

```
Union (Star (Atom a))
      (Concat (Atom a) (Atom b))
```

$a^* + (a \cdot b)$



Size & Height of Derivatives

Michael Greenberg proved that:

1. `∀ r c. height` δ_c`(r) ⩽ 2 * height r`

2. No constant bounds the size increase of Brzozowski derivatives

(size & height are defined wrt the regex AST)

<u>Our aim</u>: Prove similar results for Antimirov!

(So far, we've proven (1) holds for the <u>*max*</u> height of all terms contained in the set of Antimirov derivatives)

Zippers: Background

(JFP 1997)

FUNCTIONAL PEARL

The Zipper

GÉRARD HUET

INRIA Rocquencourt, France

Purely functional data structure for navigating trees

(Our zippers will operate over the regex AST)

Zipper = (subtree in focus, context)

path taken to reach the focused subtree `t`
+ siblings of `t`

Zippers, illustrated

```
zipper := tree * context
```

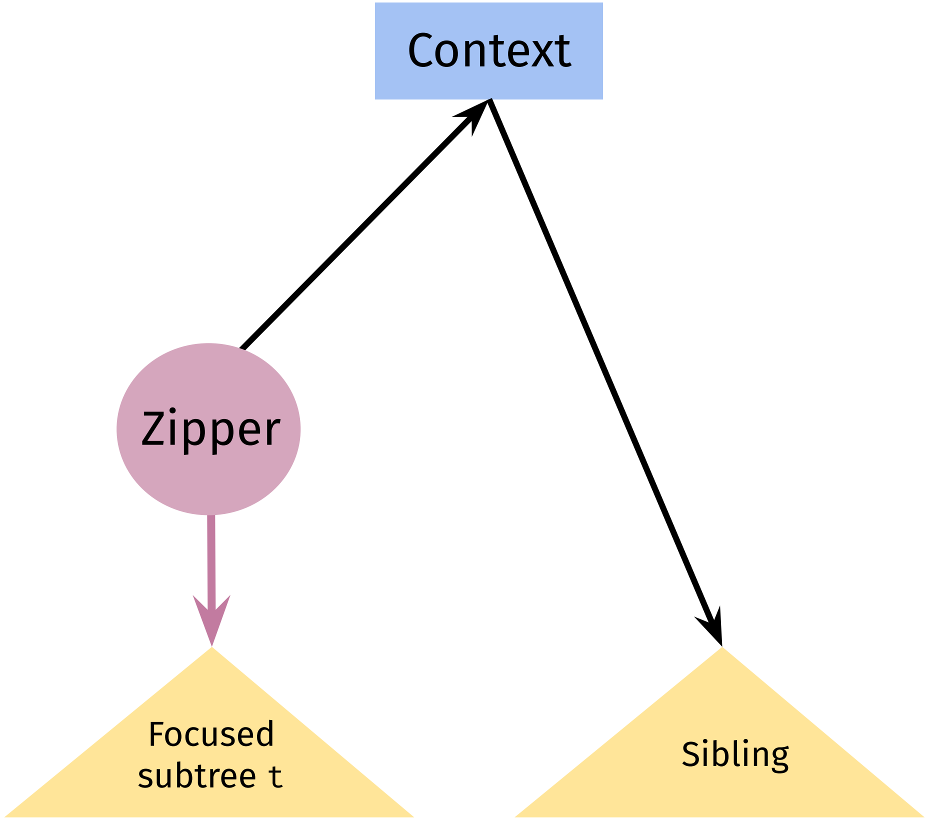
subtree t currently in focus

path taken to reach t + siblings of t

```
context :=
  | Empty
  | Left  of tree * context
  | Right of tree * context
```

left/right sibling

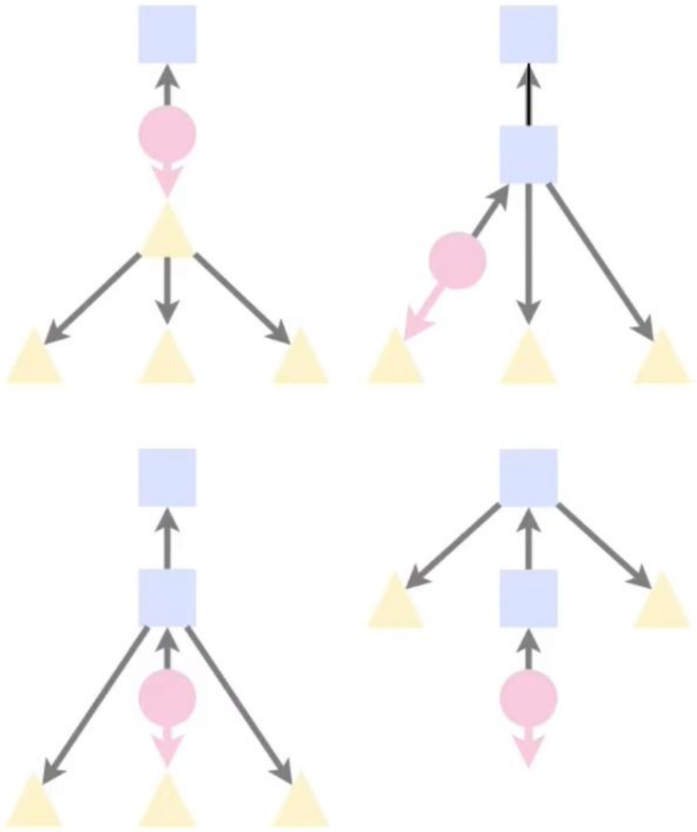
parent context



Zippers, illustrated

Key idea:

We can focus on different subtrees,
which creates a new zipper
with an updated context



Computing Brzozowski Derivatives using Zippers

doctoral thesis

Efficient Parsing with Derivatives and Zippers

Edelmann, Romain

2020

EPFL

<u>Idea:</u> Use (a variant of) zippers to compute Brzozowski derivatives!
⇒ Efficient lexing + parsing (no DFAs needed)

Computing Brzozowski Derivatives using Zippers

Step 1: Represent regex as a zipper
Step 2: Move the focus & update the context
every time δc is recursively called
(multiple recursive calls == multiple focuses)

$$
\begin{array}{lcl}
\delta_c(c) & = & \epsilon \\
\delta_c(c') & = & \bot \\
\delta_c(\epsilon) & = & \bot \\
\delta_c(r_1 \cdot r_2) & = & \begin{cases} \delta_c(r_1) \cdot r_2 \vee \delta_c(r_2) & \text{if } r_1 \text{ nullable} \\ \delta_c(r_1) \cdot r_2 & \text{otherwise} \end{cases} \\
\delta_c(\bot) & = & \bot \\
\delta_c(r_1 \vee r_2) & = & \delta_c(r_1) \vee \delta_c(r_2) \\
\delta_c(r*) & = & \delta_c(r) \cdot r*
\end{array}
$$

Encoding Regexes as Zippers: Handling +

Edelmann's insight:

Only 2 *new* kinds of AST constructors are introduced by the Brzozowski derivative:

{+, •}

When we encounter +, we need to *split* the focus between two sub-terms:

$$\delta_c(r_1 + r_2) = \delta_c(r_1) + \delta_c(r_2)$$

⇒ use **sets** to keep track of different choices of focus

Encoding Regexes as Zippers: Handling •

When we encounter •, we have to keep the rest of the expression in the context before recursively calling δ_c on r_1

$$\delta_c(r_1 \cdot r_2) = \begin{cases} \delta_c(r_1) \cdot r_2 + \ldots \\ \delta_c(r_1) \cdot r_2 \end{cases}$$

rest of the expression

Order matters ⇒ use **lists** to represent •

Encoding Regexes as (a variant of) Zippers

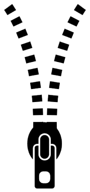
```
zipper := set context
  where
    context := list re
```

(elements in set = different choices of focus)

(elements in list = subterms to be concatenated)

$$(r_1 \cdot r_2) + r_3 \quad \cong \quad \{\ [r_1, r_2], [r_3]\ \}$$

re



Zippers ≅ Antimirov derivatives?

Edelmann: the zipper-based technique is reminiscent of Antimirov's partial derivatives

Our goal: prove that Antimirov derivatives & zipper representation of Brzozowski derivatives result in equivalent sets of regexes

What we've done:

Auxiliary lemmas, e.g. `zipper (r₁ + r₂) = zipper r₁ ∪ zipper r₂`

Extracted Edelmann's Coq zipper implementation → OCaml code

Future work:

Prove that matchers based on zippers & {Antimirov, Brzozowski} accept the same strings

Using QuickCheck to guide our Coq development

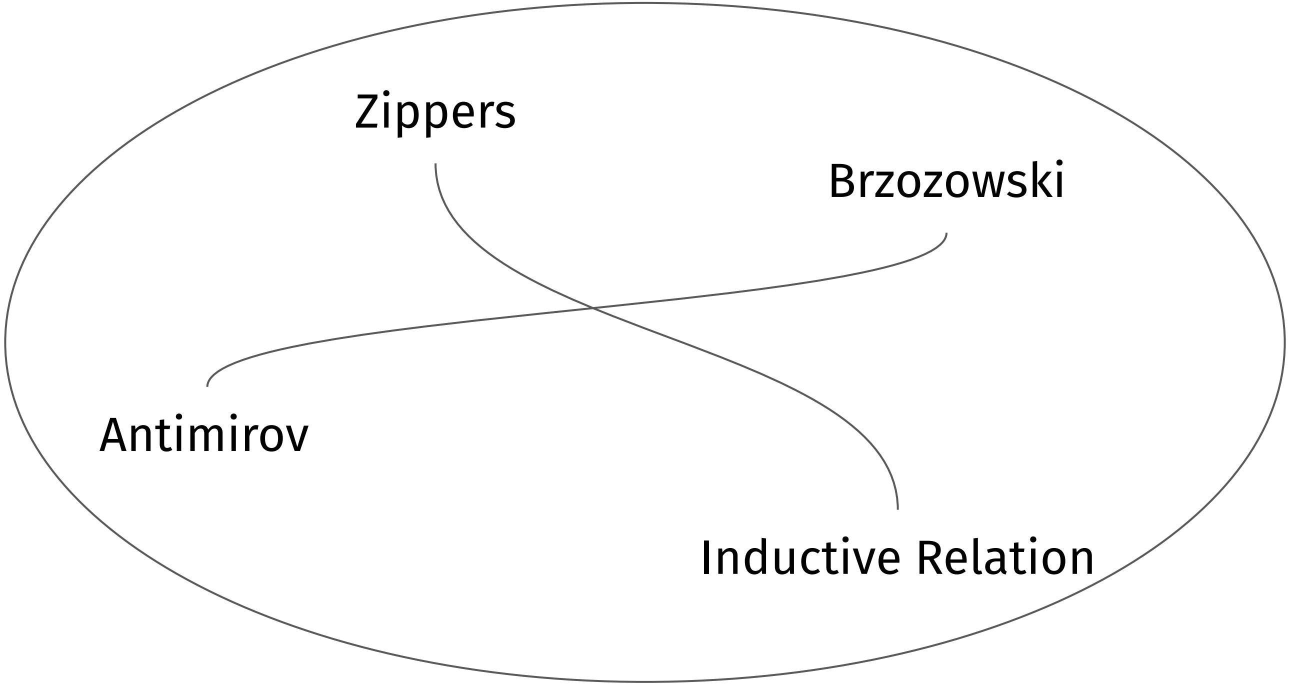
BASE_QUICKCHECK

Idea: *Test* lemma statements in OCaml before *proving* them in Coq

- Tested lemma statements on 1-10 million random regexes
- QuickCheck found counterexamples to some of our conjectured lemma statements!

The Big Picture

Bounds on height and size



Finitely many derivatives for a given regex

Big Thank You to Jules!