Finiteness of Symbolic Derivatives in Lean

Ekaterina Zhuchko[1], Hendrik Maarand[1],
Margus Veanes[2], Gabriel Ebner[2]

[1]Tallinn University of Technology, Estonia
[2]Microsoft Research, USA

ITP, October 2025

Introduction

- *Brzozowski derivatives* of regular expressions

$$\mathcal{L}(der(c, R)) := \{w \in \Sigma^* \mid c \cdot w \in \mathcal{L}(R)\}$$
$$null(R) := \epsilon \in \mathcal{L}(R)$$

Introduction

- *Brzozowski derivatives* of regular expressions

$$\mathcal{L}(der(c, R)) := \{w \in \Sigma^* \mid c \cdot w \in \mathcal{L}(R)\}$$
$$null(R) := \epsilon \in \mathcal{L}(R)$$

- (Brzozowski, 64): *finiteness* of all iterated derivatives

$$\mathcal{D}er(R) = \{der_w^*(R) \mid w \in \boldsymbol{\sigma}^*\}/_{\cong}$$

quotiented by a relation called *ACI*-similarity:

$$\begin{array}{rcll} (L \Cup R) \Cup S & \cong & L \Cup (R \Cup S) & \text{Associativity} \\ L \Cup (R \Cup L) & \cong & L \Cup R & \text{Commutativity} \\ R \Cup R & \cong & R & \text{Idempotence} \end{array}$$

This work

- We prove finiteness of *symbolic derivatives*:

This work

- We prove finiteness of *symbolic derivatives*:
    1. Derivatives do not take concrete characters

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*
  - The alphabet is symbolic and represented by an *Effective Boolean Algebra* $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*
  - The alphabet is symbolic and represented by an *Effective Boolean Algebra* $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$
- We do not assume commutativity of union

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*
  - The alphabet is symbolic and represented by an *Effective Boolean Algebra* $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$
- We do not assume commutativity of union
  - PCRE (leftmost-greedy) vs POSIX (leftmost-longest)

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*
  - The alphabet is symbolic and represented by an *Effective Boolean Algebra* $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$
- We do not assume commutativity of union
  - PCRE (leftmost-greedy) vs POSIX (leftmost-longest)
  - Let `R = (a ⩀ ab)*` and `s = "abab"`

This work

- We prove finiteness of *symbolic derivatives*:
  1. Derivatives do not take concrete characters
  2. Derivatives return a *transition term* (instead of a regex)
  3. We build an *overapproximation* for the set of all iterated derivatives
- We consider *symbolic regular expressions* with *lookarounds*
  - The alphabet is symbolic and represented by an *Effective Boolean Algebra* $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$
- We do not assume commutativity of union
  - PCRE (leftmost-greedy) vs POSIX (leftmost-longest)
  - Let `R = (a ⩀ ab)*` and `s = "abab"`
  - `"abab"` vs `"abab"`

Regular Expressions with Lookarounds

$$
\begin{aligned}
R, S \;::=\; \psi \in \alpha \;\mid\; \varepsilon \mid\; R \Cup S \mid\; R \Cap S \mid\; R \cdot S \mid\; R^* \mid\; {\sim}R \\
\mid\; \texttt{(?=}R\texttt{)} \mid\; \texttt{(?<=}R\texttt{)} \mid\; \texttt{(?!}R\texttt{)} \mid\; \texttt{(?<!}R\texttt{)}
\end{aligned}
$$

- We work modulo an alphabet theory $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{\mathsf{c}})$
  For example, $\psi_{upper} \in \alpha$ and $[\![\psi_{upper}]\!] = [A - Z]$

Regular Expressions with Lookarounds

$$R, S \;::=\; \psi \in \alpha \;\mid\; \varepsilon \mid\; R \Cup S \mid\; R \Cap S \mid\; R \cdot S \mid\; R^* \mid\; \texttt{\~{}}R$$
$$\mid\; \texttt{(?=}R\texttt{)} \mid\; \texttt{(?<=}R\texttt{)} \mid\; \texttt{(?!}R\texttt{)} \mid\; \texttt{(?<!}R\texttt{)}$$

- We work modulo an alphabet theory $\mathcal{A} = (\Sigma, \boldsymbol{\alpha}, \vDash, \bot, \top, \sqcup, \sqcap, ^{c})$
  For example, $\psi_{upper} \in \alpha$ and $[\![\psi_{upper}]\!] = [A - Z]$
- Positive lookahead $\texttt{(?=}R\texttt{)}$ and lookbehind $\texttt{(?<=}R\texttt{)}$
  Negative lookahead $\texttt{(?!}R\texttt{)}$ and lookbehind $\texttt{(?<!}R\texttt{)}$

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`
  - A *location* is of the form `("aAb","c")`

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`
  - A *location* is of the form `("aAb","c")`
  - In our setting, both the derivative and nullability functions take a location rather than a character

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`
  - A *location* is of the form `("aAb","c")`
  - In our setting, both the derivative and nullability functions take a location rather than a character
  - A *span* is of the form `("a","Ab","c")`

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`
  - A *location* is of the form `("aAb","c")`
  - In our setting, both the derivative and nullability functions take a location rather than a character
  - A *span* is of the form `("a","Ab","c")`

Regular Expressions with Lookarounds

- Lookaround conditions do not consume any characters
- They describe a context in which a match should appear
- Given a word `"aAbc"`
  - A *location* is of the form `("aAb","c")`
  - In our setting, both the derivative and nullability functions take a location rather than a character
  - A *span* is of the form `("a","Ab","c")`

Semantics

$$(xs, ys, zs) \models \texttt{(?=}R\texttt{)} \iff ys = \epsilon \wedge (xs, zs, \epsilon) \models R{\cdot}\top*$$

Example: $R = \texttt{(?=}\psi_{upper}\texttt{)}$ and `s = "aAbc"`

Then $(\texttt{"a"}, \epsilon, \texttt{"Abc"}) \models \texttt{(?=}\psi_{upper}\texttt{)}$

since $(\texttt{"a"}, \texttt{"Abc"}, \epsilon) \models \psi_{upper} \cdot \top*$ is a valid *future* match

Transition terms and symbolic derivatives

A symbolic derivative is a transition term i.e. trees of regexes

```
inductive TTerm (α : Type) : Type where
| Leaf : RE α → TTerm α
| Node : RE α → TTerm α → TTerm α → TTerm α
```

Transition terms and symbolic derivatives

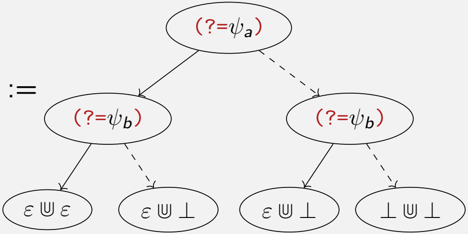
A symbolic derivative is a transition term i.e. trees of regexes

```
inductive TTerm (α : Type) : Type where
| Leaf : RE α → TTerm α
| Node : RE α → TTerm α → TTerm α → TTerm α
```

Example

Let ψ_a and ψ_b be atomic predicates.

$$\delta\ (\psi_a \Cup \psi_b) : \text{TTerm } \alpha :=$$



Transition terms and symbolic derivatives

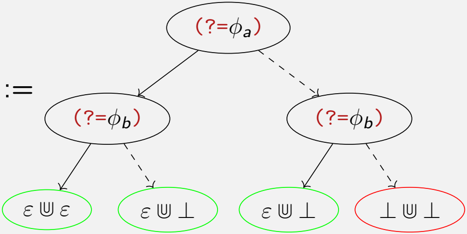
A symbolic derivative is a transition term i.e. trees of regexes

```
inductive TTerm (α : Type) : Type where
| Leaf : RE α → TTerm α
| Node : RE α → TTerm α → TTerm α → TTerm α
```

Example

Let ϕ_a and ϕ_b be atomic predicates.

$\delta\ (\phi_a \Cup \phi_b) : \text{TTerm } \alpha :=$



Semantics of transition terms

- Transition term $\approx$ a function from locations to regexes
- ... but it postpones all the nullability tests
- We define the evaluation function of type
  `Loc` σ `→ TTerm` α `→ RE` α

$$L[x] := L \quad (R, f, g)[x] := \begin{cases} f[x], \text{ if } \textit{null } R\ x; \\ g[x], \text{ otherwise.} \end{cases}$$

Symbolic derivatives

Let $\ell \in \mathsf{LA}$, $\psi \in \boldsymbol{\alpha}$ then δ : $\texttt{RE}\ \alpha \to \texttt{TTerm}\ \alpha$

$$
\begin{aligned}
\delta\ \varepsilon &:= \bot & \delta\ \ell &:= \bot \\
\delta\ \psi &:= (\texttt{(?=}\psi\texttt{)}, \varepsilon, \bot) & \delta\ (L{\cdot}R) &:= (L, \delta\ L{\cdot}R \Cup \delta\ R, \delta\ L{\cdot}R) \\
\delta\ (L \Cap R) &:= \delta\ L \Cap \delta\ R & \delta\ ({\sim}R) &:= {\sim}(\delta\ R) \\
\delta\ (L \Cup R) &:= \delta\ L \Cup \delta\ R & \delta\ (R*) &:= \delta\ R \cdot R*
\end{aligned}
$$

Symbolic derivatives

Let $\ell \in \mathsf{LA}$, $\psi \in \boldsymbol{\alpha}$ then δ : `RE` α → `TTerm` α

$$
\begin{aligned}
\delta\ \varepsilon &:= \bot & \delta\ \ell &:= \bot \\
\delta\ \psi &:= (\texttt{(?=}\psi\texttt{)}, \varepsilon, \bot) & \delta\ (L{\cdot}R) &:= (L, \delta\ L{\cdot}R \Cup \delta\ R, \delta\ L{\cdot}R) \\
\delta\ (L \Cap R) &:= \delta\ L \Cap \delta\ R & \delta\ ({\sim}R) &:= {\sim}(\delta\ R) \\
\delta\ (L \Cup R) &:= \delta\ L \Cup \delta\ R & \delta\ (R*) &:= \delta\ R \cdot R*
\end{aligned}
$$

We show the equivalence of symbolic and location-based derivatives:

Theorem 1. $\forall x \in \mathsf{Loc}, R \in RE\ \alpha : (\delta\ R)[x] = der\ R\ x$

Symbolic derivatives

Let $\ell \in \mathsf{LA}$, $\psi \in \boldsymbol{\alpha}$ then δ : $\texttt{RE}\ \alpha \rightarrow \texttt{TTerm}\ \alpha$

$$
\begin{aligned}
\delta\ \varepsilon &:= \bot & \delta\ \ell &:= \bot \\
\delta\ \psi &:= (\texttt{(?=}\psi\texttt{)}, \varepsilon, \bot) & \delta\ (L \cdot R) &:= (L, \delta\ L \cdot R \Cup \delta\ R, \delta\ L \cdot R) \\
\delta\ (L \Cap R) &:= \delta\ L \Cap \delta\ R & \delta\ ({\sim}R) &:= {\sim}(\delta\ R) \\
\delta\ (L \Cup R) &:= \delta\ L \Cup \delta\ R & \delta\ (R*) &:= \delta\ R \cdot R*
\end{aligned}
$$

We show the equivalence of symbolic and location-based derivatives:

Theorem 1. $\forall x \in \mathsf{Loc}, R \in RE\ \alpha : (\delta\ R)[x] = der\ R\ x$

$\rightarrow$ from now on, we can just work with the symbolic definition.

Iterated derivatives

- We can now compute the immediate derivatives of R:

  ```
  lvs :  TTerm α → RE α
  step (R : RE α) :  List (RE α) := lvs (δ R)
  ```

Iterated derivatives

- We can now compute the immediate derivatives of R:
  `lvs :  TTerm` α `→ RE` α
  `step (R : RE` α`) :  List (RE` α`) :=` lvs $(\delta\ R)$
- The `step` function is well-behaved wrt operations on `RE` α:

$$
\begin{aligned}
\text{step } (L \Cap R) &= \text{step } L \Cap \text{step } R \\
\text{step } (L \Cup R) &= \text{step } L \Cup \text{step } R \\
\text{step } ({\sim}R) &= {\sim}(\text{step } R) \\
\text{step } (L \cdot R) &= \text{step } L \cdot R \Cup \text{step } R \mathbin{+\!\!+} \text{step } L \cdot R \\
\text{step } (R*) &= \text{step } R \cdot R*
\end{aligned}
$$

Iterated derivatives

- We can now compute the immediate derivatives of R:

  `lvs : TTerm` α `→ RE` α

  `step (R : RE` α`) : List (RE` α`) :=` $\mathsf{lvs}\ (\delta\ R)$

- The `step` function is well-behaved wrt operations on `RE` α:

$$
\begin{aligned}
\mathsf{step}\ (L \Cap R) &= \mathsf{step}\ L \Cap \mathsf{step}\ R \\
\mathsf{step}\ (L \Cup R) &= \mathsf{step}\ L \Cup \mathsf{step}\ R \\
\mathsf{step}\ ({\sim}R) &= {\sim}(\mathsf{step}\ R) \\
\mathsf{step}\ (L \cdot R) &= \mathsf{step}\ L \cdot R \Cup \mathsf{step}\ R \mathbin{+\!\!+} \mathsf{step}\ L \cdot R \\
\mathsf{step}\ (R{*}) &= \mathsf{step}\ R \cdot R{*}
\end{aligned}
$$

- We compute the n-th derivatives (words of length n):

  `steps : RE` α `→ Nat → List (RE` α`)`

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\mathbb{U}$

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)
  - Derivative function returns a set of expressions

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)
  - Derivative function returns a set of expressions
  - Proving finiteness is more straightforward but hard to deal with intersection and complement

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)
  - Derivative function returns a set of expressions
  - Proving finiteness is more straightforward but hard to deal with intersection and complement
- **Our approach**

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)
  - Derivative function returns a set of expressions
  - Proving finiteness is more straightforward but hard to deal with intersection and complement
- **Our approach**
  - Follow Antimirov's strategy for finiteness

Finiteness of the state space

- **Classical approach** (Brzozowski/DFA construction)
  - $\mathcal{D}er(R) = \{der_w(R) \mid w \in \Sigma^*\}/_{\cong}$
  - $\cong$ is the equivalence induced by ACI for union $\Cup$
- **Antimirov (partial) derivatives** (NFA construction)
  - Derivative function returns a set of expressions
  - Proving finiteness is more straightforward but hard to deal with intersection and complement
- **Our approach**
  - Follow Antimirov's strategy for finiteness
  - While dealing with the extended class of expressions

Similarity

We define helpers to reason *up-to* a relation R

- List membership
  `x` $\in$`[ R ] ys` $:=\exists$ `y, R x y` $\wedge$ `y` $\in$ `ys`
- List inclusion
  `xs` $\subseteq$`[ R ] ys` $:=\forall$ `x` $\in$ `xs, x` $\in$`[ R ] ys`
- List equality
  `xs =[ R ] ys` $:=$ `xs` $\subseteq$`[ R ] ys` $\wedge$ `ys` $\subseteq$`[ R ] xs`

Similarity

We define helpers to reason *up-to* a relation R

- List membership
  `x ∈[ R ] ys` $:=$ `∃ y, R x y ∧ y ∈ ys`
- List inclusion
  `xs ⊆[ R ] ys` $:=$ `∀ x ∈ xs, x ∈[ R ] ys`
- List equality
  `xs =[ R ] ys` $:=$ `xs ⊆[ R ] ys ∧ ys ⊆[ R ] xs`

Our relation: the ADI-similarity relation used for quotienting:

$(L \Cup R) \Cup S \cong L \Cup (R \Cup S)$	Associativity
$L \Cup (R \Cup L) \cong L \Cup R$	right Deduplication
$R \Cup R \cong R$	Idempotence

Finiteness of Antimirov derivatives

- Why is proving finiteness easy for Antimirov derivatives?

$$
\begin{array}{rcl}
\mathit{support}(\bot) & := & \emptyset \\
\mathit{support}(\varepsilon) & := & \emptyset \\
\mathit{support}(c) & := & \{\varepsilon\} \text{ with } c \ \in \Sigma \\
\mathit{support}(L \Cup R) & := & \mathit{support}(L) \cup \mathit{support}(R) \\
\mathit{support}(L \cdot R) & := & \mathit{support}(L) \cdot R \cup \mathit{support}(R) \\
\mathit{support}(R*) & := & \mathit{support}(R) \cdot R*
\end{array}
$$

Finiteness of Antimirov derivatives

- Why is proving finiteness easy for Antimirov derivatives?

$$\begin{array}{rcl} support(\bot) & := & \emptyset \\ support(\varepsilon) & := & \emptyset \\ support(c) & := & \{\varepsilon\} \text{ with } c \ \in \Sigma \\ support(L \Cup R) & := & support(L) \cup support(R) \\ support(L{\cdot}R) & := & support(L) \cdot R \cup support(R) \\ support(R*) & := & support(R) \cdot R* \end{array}$$

- All Antimirov derivatives are contained in the set:

$$\{R\} \cup support(R)$$

Finiteness of Antimirov derivatives

- Why is proving finiteness easy for Antimirov derivatives?

$$
\begin{array}{rcl}
support(\bot) & := & \emptyset \\
support(\varepsilon) & := & \emptyset \\
support(c) & := & \{\varepsilon\} \text{ with } c \ \in \Sigma \\
support(L \Cup R) & := & support(L) \cup support(R) \\
support(L \cdot R) & := & support(L) \cdot R \cup support(R) \\
support(R*) & := & support(R) \cdot R*
\end{array}
$$

- All Antimirov derivatives are contained in the set:

$$\{R\} \cup support(R)$$

- ACI is built into the set representation

Finiteness of Antimirov derivatives

- Why is proving finiteness easy for Antimirov derivatives?

$$
\begin{array}{rcl}
\mathit{support}(\bot) & := & \emptyset \\
\mathit{support}(\varepsilon) & := & \emptyset \\
\mathit{support}(c) & := & \{\varepsilon\} \text{ with } c \in \Sigma \\
\mathit{support}(L \Cup R) & := & \mathit{support}(L) \cup \mathit{support}(R) \\
\mathit{support}(L{\cdot}R) & := & \mathit{support}(L) \cdot R \cup \mathit{support}(R) \\
\mathit{support}(R*) & := & \mathit{support}(R) \cdot R*
\end{array}
$$

- All Antimirov derivatives are contained in the set:

$$\{R\} \cup \mathit{support}(R)$$

- ACI is built into the set representation
- Can we use a similar strategy for Brzozowski-style derivatives?

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R
- We have to show finiteness of this set

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R
- We have to show finiteness of this set
- Solution: **finite overapproximation** (modulo ADI)

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R
- We have to show finiteness of this set
- Solution: **finite overapproximation** (modulo ADI)

$$
\begin{tikzcd}
a \Cup b \cdot c \arrow[r, "\mathrm{der}_a"] \arrow[d, "\texttt{pieces}"'] & \varepsilon \Cup \bot \cdot c \arrow[dl, "\ni"] \\
{[\bot, \varepsilon, a] \mathbin{+\!\!+} [\bot, \varepsilon, c, \bot \cdot c, \varepsilon \cdot c, b \cdot c]} &
\end{tikzcd}
$$

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R
- We have to show finiteness of this set
- Solution: **finite overapproximation** (modulo ADI)

$$\begin{tikzcd} a \Cup b \cdot c \arrow[r, "\mathrm{der}_a"] \arrow[d, "\texttt{pieces}"'] & \varepsilon \Cup \bot \cdot c \arrow[dl, "\ni"] \\ {[\bot, \varepsilon, a] \mathbin{+\!\!+} [\bot, \varepsilon, c, \bot \cdot c, \varepsilon \cdot c, b \cdot c]} & \end{tikzcd}$$

- One step: ε and $\bot \cdot c$ are contained in *pieces* $(a \Cup b \cdot c)$

Constructing the overapproximation

- What we have: a way to reason about derivatives and their iterated forms to describe all states reachable from R
- We have to show finiteness of this set
- Solution: **finite overapproximation** (modulo ADI)

$$
\begin{tikzcd}
a \Cup b \cdot c \arrow[r, "\mathrm{der}_a"] \arrow[d, "\texttt{pieces}"'] & \varepsilon \Cup \bot \cdot c \arrow[dl, "\ni"] \\
{[\bot, \varepsilon, a] \mathbin{+\!\!+} [\bot, \varepsilon, c, \bot \cdot c, \varepsilon \cdot c, b \cdot c]} &
\end{tikzcd}
$$

- One step: ε and $\bot \cdot c$ are contained in *pieces* $(a \Cup b \cdot c)$
- **Key idea:** all derivatives can be given as union of `pieces`

Pieces

- We don't have commutativity of union so we have to consider all permutations of a list:
  $\bigoplus$`[a, b] = [a, a ⋓ b, b ⋓ a, b]`

Pieces

- We don't have commutativity of union so we have to consider all permutations of a list:
  `⨁[a, b] = [a, a ⋓ b, b ⋓ a, b]`
- For intersection we use the Cartesian product:
  `productWith (· + ·) [1,2] [3,4,5] = [4,5,6,5,6,7]`

Pieces

- We don't have commutativity of union so we have to consider all permutations of a list:
  `⨁[a, b] = [a, a ⋓ b, b ⋓ a, b]`
- For intersection we use the Cartesian product:
  `productWith (· + ·) [1,2] [3,4,5] = [4,5,6,5,6,7]`

Pieces

- We don't have commutativity of union so we have to consider all permutations of a list:
  `⊕[a, b] = [a, a ⋓ b, b ⋓ a, b]`
- For intersection we use the Cartesian product:
  `productWith (· + ·) [1,2] [3,4,5] = [4,5,6,5,6,7]`

```
def pieces : RE α → List (RE α)
  | ε       => [ε, Pred ⊥]
  | Pred φ  => [Pred φ, ε, Pred ⊥]
  | ?= r    => [?= r, ε, Pred ⊥] | ...
  | l ⋓ r   => pieces l ++ pieces r
  | l ⋒ r   => productWith (· ⋒ ·) ⊕(pieces l) ⊕(pieces r)
  | ~r      => map (~ ·) ⊕(pieces r)
  | l · r   => map (· · r) ⊕(pieces l) ++ pieces r
  | r*      => r* :: map (· · r*) ⊕(pieces r)
```

Main theorem

1. **Reflexivity**:
```
∀ r,
∃ xs, toSum xs ≅ r ∧ xs ∈ neSublists (pieces r)
```
2. **Transitivity**:
```
   e ∈ pieces f
 → f ∈ pieces g
 → e ∈[ (· ≅ ·) ] pieces g
```
3. **One-step reconstruction**:
```
∀ r d, d ∈ step r
∃ xs, toSum xs ≅ d ∧ xs ∈ neSubsets (pieces r)
```

Main theorem

1. **Reflexivity**:

```
∀ r,
∃ xs, toSum xs ≅ r ∧ xs ∈ neSublists (pieces r)
```

2. **Transitivity**:

```
   e ∈ pieces f
 → f ∈ pieces g
 → e ∈[ (· ≅ ·) ] pieces g
```

3. **One-step reconstruction**:

```
∀ r d, d ∈ step r
∃ xs, toSum xs ≅ d ∧ xs ∈ neSubsets (pieces r)
```

- **Main result**: every iterated derivative of R can be reconstructed as a sum of regexes from `pieces R`

Main theorem

1. **Reflexivity**:
```
∀ r,
∃ xs, toSum xs ≅ r ∧ xs ∈ neSublists (pieces r)
```
2. **Transitivity**:
```
   e ∈ pieces f
 → f ∈ pieces g
 → e ∈[ (· ≅ ·) ] pieces g
```
3. **One-step reconstruction**:
```
∀ r d, d ∈ step r
∃ xs, toSum xs ≅ d ∧ xs ∈ neSubsets (pieces r)
```

- **Main result**: every iterated derivative of R can be reconstructed as a sum of regexes from `pieces R`

```
theorem finiteness [DecidableEq α] {r : RE α} :
  ∃ (xs : List (RE α)),
    ∀ {n : ℕ}, steps r n ⊆[ (· ≅ ·) ] xs
```

Main theorem

1. **Reflexivity**:
```
∀ r,
∃ xs, toSum xs ≅ r ∧ xs ∈ neSublists (pieces r)
```
2. **Transitivity**:
```
   e ∈ pieces f
 → f ∈ pieces g
 → e ∈[ (· ≅ ·) ] pieces g
```
3. **One-step reconstruction**:
```
∀ r d, d ∈ step r
∃ xs, toSum xs ≅ d ∧ xs ∈ neSubsets (pieces r)
```

- **Main result**: every iterated derivative of R can be reconstructed as a sum of regexes from `pieces R`

```
theorem finiteness [DecidableEq α] {r : RE α} :
  ∃ (xs : List (RE α)),
    ∀ {n : ℕ}, steps r n ⊆[ (· ≅ ·) ] xs
```

- The witness is `xs := ⊕(pieces R)`

Previous work

- Coquand & Siles (2011), Nipkow & Traytel (2014) use canonical/normal forms

Previous work

- Coquand & Siles (2011), Nipkow & Traytel (2014) use canonical/normal forms
  - Two versions of the normalisation function: one that only implements ACI and one which implements more aggressive simplifications

Previous work

- Coquand & Siles (2011), Nipkow & Traytel (2014) use canonical/normal forms
  - Two versions of the normalisation function: one that only implements ACI and one which implements more aggressive simplifications
- We instead compute a finite overapproximation of all reachable derivatives

Previous work

- Coquand & Siles (2011), Nipkow & Traytel (2014) use canonical/normal forms
  - Two versions of the normalisation function: one that only implements ACI and one which implements more aggressive simplifications
- We instead compute a finite overapproximation of all reachable derivatives
- (Moreira et al., 2012) avoid the need for normalisation modulo ACI by using Antimirov derivatives

Previous work

- Coquand & Siles (2011), Nipkow & Traytel (2014) use canonical/normal forms
  - Two versions of the normalisation function: one that only implements ACI and one which implements more aggressive simplifications
- We instead compute a finite overapproximation of all reachable derivatives
- (Moreira et al., 2012) avoid the need for normalisation modulo ACI by using Antimirov derivatives
- We take inspiration from the Antimirov finiteness proof, but adapt it to handle intersection and negation

Simplifications

Which simplifications preserve the finiteness result?

Simplifications

Which simplifications preserve the finiteness result?

```
def NonIncreasing (f : RE α → RE α) : Prop :=
  ∀ r, pieces (f r) ⊆ pieces r
```

Simplifications

Which simplifications preserve the finiteness result?

```
def NonIncreasing (f : RE α → RE α) : Prop :=
  ∀ r, pieces (f r) ⊆ pieces r
```

```
∀ (f : RE α → RE α) r,
    NonIncreasing f
  → map f (step r) ⊆[ (· ≅ ·) ] ⊕(pieces r)
```

Simplifications

Which simplifications preserve the finiteness result?

```
def NonIncreasing (f : RE α → RE α) : Prop :=
  ∀ r, pieces (f r) ⊆ pieces r
```

```
∀ (f : RE α → RE α) r,
    NonIncreasing f
  → map f (step r) ⊆[ (· ≅ ·) ] ⊕(pieces r)
```

Allowed simplifications

```
⊥ ⋓ s ⇝ s     ~⊥ ⋓ s ⇝ ~⊥     ε · s ⇝ s
r ⋓ ⊥ ⇝ r     r ⋓ ~⊥ ⇝ ~⊥     r · ⊥ ⇝ ⊥

(r ⋓ s) · t ⇝ r · t ⋓ s · t
```

- `r · s ⇝ s` and `r ⋓ s ⇝ r` are allowed

Simplifications

Which simplifications preserve the finiteness result?

```
def NonIncreasing (f : RE α → RE α) : Prop :=
  ∀ r, pieces (f r) ⊆ pieces r
```

```
∀ (f : RE α → RE α) r,
    NonIncreasing f
  → map f (step r) ⊆[ (· ≅ ·) ] ⊕(pieces r)
```

Allowed simplifications

$$\bot \Cup s \leadsto s \qquad {\sim}\bot \Cup s \leadsto {\sim}\bot \qquad \varepsilon \cdot s \leadsto s$$
$$r \Cup \bot \leadsto r \qquad r \Cup {\sim}\bot \leadsto {\sim}\bot \qquad r \cdot \bot \leadsto \bot$$
$$(r \Cup s) \cdot t \leadsto r \cdot t \Cup s \cdot t$$

- $r \cdot s \leadsto s$ and $r \Cup s \leadsto r$ are allowed
- $r \cdot s \leadsto r$ is **not** allowed

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work
- We show which simplifications can be applied to derivatives, preserving finiteness

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work
- We show which simplifications can be applied to derivatives, preserving finiteness
- No assumption that the alphabet is finite; the alphabet algebra can even be undecidable or semidecidable

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work
- We show which simplifications can be applied to derivatives, preserving finiteness
- No assumption that the alphabet is finite; the alphabet algebra can even be undecidable or semidecidable
- How to make this into a reusable framework for finiteness? (e.g. for other regex classes/logics)

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work
- We show which simplifications can be applied to derivatives, preserving finiteness
- No assumption that the alphabet is finite; the alphabet algebra can even be undecidable or semidecidable
- How to make this into a reusable framework for finiteness? (e.g. for other regex classes/logics)

Conclusion

We formally prove in Lean that the set of symbolic derivatives of regexes with lookarounds is finite modulo ADI

- Almost 2000 loc of Lean, modularly reusing previous work
- We show which simplifications can be applied to derivatives, preserving finiteness
- No assumption that the alphabet is finite; the alphabet algebra can even be undecidable or semidecidable
- How to make this into a reusable framework for finiteness? (e.g. for other regex classes/logics)

Thank you!

`github.com/ezhuchko/finiteness-derivatives`