# Deciding Kleene Algebra Terms (In-)Equivalence in Coq

Nelma Moreira, David Pereira and Simão Melo de Sousa

Tallinn
September 2016

# Outline

# Table of Contents

# Kleene Algebra

Idempotent semiring $(K, +, \cdot, 0, 1)$:

$$x + x = x \tag{1}$$
$$x + 0 = x \tag{2}$$
$$x + y = y + x \tag{3}$$
$$x + (y + z) = (x + y) + z \tag{4}$$
$$0x = 0 \tag{5}$$
$$x0 = 0 \tag{6}$$
$$1x = x \tag{7}$$
$$x1 = x \tag{8}$$
$$x(yz) = (xy)z \tag{9}$$
$$x(y + z) = xy + xz \tag{10}$$
$$(x + y)z = xz + yz. \tag{11}$$

Consider $x \leq y \triangleq x + y = y$.

Kleene Algebra (KA): $(K, +, \cdot, ^\star, 0, 1)$ such that the sub-algebra $(K, +, \cdot, 0, 1)$ is an idempotent semiring and that the operator $^\star$ is characterized by the following axioms:

$$1 + pp^\star \leq p^\star \tag{12}$$
$$1 + p^\star p \leq p^\star \tag{13}$$
$$q + pr \leq r \to p^\star q \leq r \tag{14}$$
$$q + rp \leq r \to qp^\star \leq r \tag{15}$$

**Standard Model of KA:** $(\mathrm{RL}_\Sigma, \cup, \cdot, ^\star, \emptyset, \{\epsilon\})$

# Regular expressions and Languages

- Regular expression:

$$\alpha, \beta \ ::= \ 0 \mid 1 \mid a \in \Sigma \mid \alpha + \beta \mid \alpha\beta \mid \alpha^\star$$

- Language denoted by a regular expression:

$$\begin{array}{ccc} \mathcal{L}(0) = \emptyset & \mathcal{L}(1) = \{\epsilon\} & \mathcal{L}(a) = \{a\} \\ \mathcal{L}(\alpha + \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta) & \mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta) & \mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^\star \end{array}$$

- Regular expression equivalence:

$$\alpha \sim \beta \text{ iff } \mathcal{L}(\alpha) = \mathcal{L}(\beta)$$

- Nullability:

$$\varepsilon(\alpha) = \begin{cases} \texttt{true} & \text{if} \quad \epsilon \in \mathcal{L}(\alpha) \\ \texttt{false} & \text{if} \quad \epsilon \notin \mathcal{L}(\alpha) \end{cases}$$

# Partial Derivatives

- Definition of Partial Derivative wrt $a \in \Sigma$ [Mirkin,Antimirov]:

$$
\begin{array}{rcl}
\partial_a(0) & = & \emptyset \\
\partial_a(1) & = & \emptyset \\
\partial_a(b) & = & \left\{ \begin{array}{ll} \{1\} & \text{if } a \equiv b \\ \emptyset & \text{otherwise} \end{array} \right. \\
\partial_a(\alpha + \beta) & = & \partial_a(\alpha) \cup \partial_a(\beta) \\
\partial_a(\alpha\beta) & = & \left\{ \begin{array}{ll} \partial_a(\alpha)\beta \cup \partial_a(\beta) & \text{if } \varepsilon(\alpha) = \texttt{true}, \\ \partial_a(\alpha)\beta & \text{otherwise.} \end{array} \right. \\
\partial_a(\alpha^\star) & = & \partial_a(\alpha)\alpha^\star
\end{array}
$$

## Partial Derivatives (cont.)

- Partial Derivatives wrt Words:

$$\begin{array}{rcl} \partial_{\varepsilon}(\alpha) & = & \{\alpha\} \\ \partial_{wa}(\alpha) & = & \partial_a(\partial_w(\alpha)). \end{array}$$

- Language of Partial Derivative: $\mathcal{L}(\partial_a(\alpha)) = a^{-1}(\mathcal{L}(\alpha))$

- **Example:**
$\partial_{abb}(ab^\star) = \partial_b(\partial_b(\partial_a(ab^\star))) = \partial_b(\partial_b(\partial_a(a)b^\star))$
$= \partial_b(\partial_b(\{b^\star\})) = \partial_b(\partial_b(b)b^\star) = \partial_b(\{b^\star\}) = \{b^\star\}$

- An interesting consequence: $w \in \mathcal{L}(\alpha) \leftrightarrow \varepsilon(\partial_w(\alpha)) = \texttt{true}$

- Set of all Partial Derivatives: $PD(\alpha) = \bigcup_{w \in \Sigma^\star}(\partial_w(\alpha))$

- Finiteness of *PD* [Mirkin,Antimirov] : $PD(\alpha) \leq |\alpha|_\Sigma + 1$

# (In-)Equivalence Through Iterated Derivation

$$\alpha \sim \varepsilon(\alpha) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\alpha)) \tag{16}$$

If $\alpha \sim \beta$, then by (16) :

$$\varepsilon(\alpha) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\alpha)) \sim \varepsilon(\beta) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\beta)) \tag{17}$$

By (17) and knowing that $w \in \mathcal{L}(\alpha) \leftrightarrow \varepsilon(\partial_w(\alpha)) = \texttt{true}$, we obtain:

$$(\forall w \in \Sigma^\star, \varepsilon(\partial_w(\alpha)) = \varepsilon(\partial_w(\beta))) \leftrightarrow \alpha \sim \beta. \tag{18}$$

$$\varepsilon(\partial_w(\alpha)) \neq \varepsilon(\partial_w(\beta))) \rightarrow \alpha \not\sim \beta, \quad \textit{for some } w \in \Sigma^\star. \tag{19}$$

# The Procedure equivP

**Require:** $S = \{(\{\alpha\}, \{\beta\})\}$, $H = \emptyset$
**Ensure:** true or false

**procedure** EquivP($S$, $H$)

  **while** $S \neq \emptyset$ **do**

    $(S_\alpha, S_\beta) \leftarrow POP(S)$

    **if** $\varepsilon(S_\alpha) \neq \varepsilon(S_\beta)$ **then**

      **return** `false`

    **end if**

    $H \leftarrow H \cup \{(S_\alpha, S_\beta)\}$

    **for** $a \in \Sigma$ **do**

      $(S'_\alpha, S'_\beta) \leftarrow \partial_a(S_\alpha, S_\beta)$

      **if** $(S'_\alpha, S'_\beta) \notin H$ **then**

        $S \leftarrow S \cup \{(S'_\alpha, S'_\beta)\}$

      **end if**

    **end for**

  **end while**

**return** `true`

**end procedure**

- Construct a bisimulation that leads to (18) or finds a counter-example that prove that such a bisimulation does not exist (19).
- $S$: Derivatives yet to be processed
- $H$: Processed derivatives ($H$ is finite)
- if *false*, then counter-example

# The Procedure equivP, an example

- Consider $\alpha = (ab)^\star a$ and $\beta = a(ba)^\star$.
- Then $s_0 = (\{\alpha, \beta\}) = (\{(ab)^\star a\}, \{a(ba)^\star\})$
- We must show that $\mathrm{equivP}(\{s_0\}, \emptyset) = \mathtt{true}$.
- equivP for such $\alpha$ and $\beta$ computes $s_1 = (\{1, b(ab)^\star a\}, \{(ba)^\star\})$ and $s_2 = (\emptyset, \emptyset)$.
- Execution traces:

| $i$ | $S_i$ | $H_i$ | *drvs.* |
|---|---|---|---|
| 0 | $\{s_0\}$ | $\emptyset$ | $\partial_a(s_0) = s_1, \partial_b(s_0) = s_2$ |
| 1 | $\{s_1, s_2\}$ | $\{s_0\}$ | $\partial_a(s_1) = s_2, \partial_b(s_1) = s_0$ |
| 2 | $\{s_2\}$ | $\{s_0, s_1\}$ | $\partial_a(s_2) = s_2, \partial_b(s_2) = s_2$ |
| 3 | $\emptyset$ | $\{s_0, s_1, s_2\}$ | $\mathtt{true}$ |

# Table of Contents

# Ingredient 1 : Representation of Derivatives

- Derivatives as *dependent records*:

```
Record Drv (α β:re) := mkDrv {
 dp :> set re * set re ;
 w : word ;
 cw : dp = (∂w(α),∂w(β))
}.
```

## Example (Original regular expression)

```
Definition Drv_1st (α β:re) : Drv α β.
refine(mkDrv ({α},{β}) ϵ _).
abstract(reflexivity).
Defined.
```

# Ingredient 2 : Derivation of Drv terms

- Derivation of Drv terms wrt $a \in \Sigma$:

```
Definition Drv_pdrv(x:Drv α β)(a:A) : Drv α β.
refine(match x with
       | mkDrv α β p w H ⇒
            mkDrv α β (pdrvp p a) (w++[a]) _
       end).
abstract((* Proof of ∂_a(∂_w(α),∂_w(β)) = (∂_wa(α),∂_wa(β)) *)).
Defined.
```

- Derivation of Drv terms wrt a set of symbols:

```
Definition Drv_pdrv_set(x:Drv α β)(Sig:set A) :
    set (Drv α β) :=
  fold (fun a:A ⇒ add (Drv_pdrv α β x a)) Sig ∅.
```

- Ignoring already existing derivatives in $H$:

```
Definition Drv_pdrv_set_filtered(x:Drv α β)
  (H:set(Drv α β))(sig:set A):set (Drv α β) :=
  filter (fun y ⇒ negb (y ∈ H)) (Drv_pdrv_set x sig).
```

# Ingredient 3 : One Step of Computation

```
Inductive step_case (α β:re) : Type :=
|proceed : step_case α β
|termtrue : set (Drv α β) → step_case
    α β
|termfalse : Drv α β → step_case α β.
```

- proceed: continue the iterative process;
- termtrue: the procedure must terminate and use the parameter as a witness of equivalence;
- termfalse: the procedure must terminate and use the parameter as a counter-example of equivalence.

```
(*step = lines 8-13, for loop of EquivP*)
Definition step (H S:set (Drv α β))(sig:set A) :
 ((set (Drv αβ) * set (Drv α β)) * step_case α β) :=
 match choose s with
 |None ⇒ ((H,S),termtrue α β H)
 |Some (Sα,Sβ) ⇒
   if c_of_Drv _ _ (Sα,Sβ) then
    let H' := add (Sα,Sβ) H in
     let S' := remove (Sα,Sβ) S in
      let ns := Drv_pdrv_set_filtered α β (Sα,Sβ) H' sig in
       ((H',ns ∪ S'),proceed α β)
   else
    ((H,S),termfalse α β (Sα,Sβ))
 end.
```

# Ingredient 4 : Termination

- Considering

  `step` $\alpha$ $\beta$ $H$ $S$ `= ((`$H'$`,`$S'$`),proceed` $\alpha$ $\beta$`)`

  and

  $$S \cap H = \emptyset$$

- the termination is ensured by:

  $$(2^{(|\alpha|_\Sigma+1)} \times 2^{(|\beta|_\Sigma+1)}+1) - |H'| < (2^{(|\alpha|_\Sigma+1)} \times 2^{(|\beta|_\Sigma+1)}+1) - |H|$$

# Ingredient 4 : Main function

- iterator :

```
Function iterate(α β:re)(H S:set (Drv α β))
 (sig:set A)(D:DP α β h s){wf (LLim α β) H}:
     term_cases α β :=
  let ((H',S',next) := step H S in
   match next with
    |termfalse x ⇒ NotOk α β x
    |termtrue h ⇒ Ok α β h
    |progress ⇒ iterate α β H' S' sig (DP_upd α β H S
         sig D)
  end.
```

- where DP is defined as

```
Inductive DP (h s:set (Drv α β)) : Prop :=
| is_dpt : h ∩ s = ∅ → ε(h) = true → DP h s.
```

# The function equivP

- wrap iterate into a Boolean function:

```
Definition equivP_aux(α β:re)(H S:set(Drv α β))
 (sig:set A)(D:DP α β H S):=
 let H' := iterate α β H S sig D in
  match H' with
  | Ok _ ⇒ true
  | NotOk _ ⇒ false
 end.
```

- instantiate with the correct arguments:

```
Definition equivP (α β:re) :=
  equivP_aux α β ∅ {Drv_1st α β} (setSy α ∪ setSy β)
      (mkDP_ini α β).
```

# Correctness

```
Lemma equiv_re_false :
  ∀ α β, equivP α β = false → α ≁ β
```

1. this only happens when :

$$\texttt{iterate}\ H\ S = \texttt{NotOk}\ \alpha\ \beta\ (S_\alpha, S_\beta)$$

2. which means that:

$$\texttt{step}\ H'\ S' = \texttt{termfalse}\ \alpha\ \beta\ (S_\alpha, S_\beta)$$

3. be definition of `step` we know that:

$$\varepsilon(S_\alpha) \neq \varepsilon(S_\beta)$$

4. thus:

$$\alpha \not\sim \beta$$

# Correctness

```
Lemma equiv_re_true :
  ∀ α β, equivP α β = true → α ∼ β
```

1. define the following invariant:

$$INV(H,S) =_{def} \forall x,\ x \in H \rightarrow \forall a \in \Sigma,\ \partial_a(x) \in S \cup H$$

2. prove that it holds for step:

$$INV(H,S) \rightarrow \texttt{step}\ H\ S = ((H',S'),\texttt{proceed}) \rightarrow INV(H',S')$$

3. prove that all derivatives are computed :

$$INV(H,S) \rightarrow \texttt{iterate}\ H\ S\ = \texttt{Ok}\ \_\ \_\ H' \rightarrow INV(H',\emptyset)$$

4. prove that all derivatives $(S_\alpha, S_\beta)$ verify $\varepsilon(S_\alpha) = \varepsilon(S_\beta)$
5. thus we obtain $\forall w \in \Sigma^\star, \varepsilon(\partial_w(\alpha)) = \varepsilon(\partial_w(\beta)))$
6. from which follows $\alpha \sim \beta$

# Completeness

Obtained by trivial case analysis:

- $\alpha \sim \beta$:
  1. if $\texttt{equivP}\ \alpha\ \beta = \texttt{true}$ : trivial from correctness proof;
  2. if $\texttt{equivP}\ \alpha\ \beta = \texttt{false}$ : contradiction
- $\alpha \not\sim \beta$: by similar reasoning

# The Reflexive Tactic

From the soundness results we were able to construct the following tactic:

```
Ltac re_equiv :=
 apply equiv_re_true;vm_compute;
  first [ reflexivity | fail 2 "Regular expressions are not
      equivalent" ].

Ltac re_inequiv :=
 apply equiv_re_false;vm_compute;
  first [ reflexivity | fail 2 "Regular expressions not
      inequivalent" ].

Ltac dec_re :=
 match goal with
 | |- ℒ(?R1) ∼ ℒ(?R2) ⇒ re_equiv
 | |- ℒ(?R1) ≁ ℒ(?R2) ⇒ re_inequiv
 | |- ℒ(?R1) ≤ ℒ(?R2) ⇒
    unfold lleq;change (ℒ(R1) ∪ ℒ(R2)) with (ℒ(R1 + R2));
     re_equiv
 | |- _ ⇒ fail 2 "Not a regular expression (in−)equivalence
     equation."
 end.
```

# Table of Contents

# Performance

Some indicators (10000 pairs of uniform, randomly generated regular expressions):

- $|\alpha| = 25$ and 10 symbols : 0.142 (eq) and 0.025 (ineq)
- $|\alpha| = 50$ and 20 symbols : 0.446 (eq) and 0.060 (ineq)
- $|\alpha| = 100$ and 30 symbols : 1.142s (eq) and 0.112s (ineq)
- $|\alpha| = 250$ and 40 symbols : 5.142s (eq) and 0.147s (ineq)
- $|\alpha| = 1000$ and 50 symbols : 46.037s (eq) and 0.280 (ineq)

| alg./$(k, n)$ | $(20, 200)$ | | $(50, 500)$ | | $(50, 1000)$ | |
|---|---|---|---|---|---|---|
| | *eq* | *ineq* | *eq* | *ineq* | *eq* | *ineq* |
| equivP | 2.211 | 0.048 | 9.957 | 0.121 | 17.768 | 0.149 |
| ATBR | 3.001 | 1.654 | 5.876 | 2.724 | 16.682 | 12.448 |

Table: Comparison of the performances (ATBR - Braibant & Pous).

Regular expression generated using the FAdo toolbox:
`http://http://fado.dcc.fc.up.pt/`

# Table of Contents

# Relations vs. Regular expressions

Claim: Equations over relation can be decided using regular expressions

First ingredient:

```
Fixpoint reRel(v:nat→ relation B)(α:re) : relation B :=
 match r with
 | 0 ⇒ EmpRel
 | 1 ⇒ IdRel
 | ‘a ⇒ v a
 | x + y ⇒ UnionRel (reRel v x) (reRel v y)
 | x · y ⇒ CompRel (reRel v x) (reRel v y)
 | x⋆ ⇒ TransRefl (reRel v x)
 end.
```

# Example

Consider:

- $\Sigma = \{a, b\}$,
- $R_a$ and $R_b$ : binary relations over `B`,
- a regular expression $\alpha = a(b+1)$
- $v$: a function that maps $a$ to the relation $R_a$, and $b$ to the relation $R_b$.
- The computation of `reRel` $\alpha$ `v` gives the relation $R_a \circ (R_b \cup \mathcal{I})$, and can be described as follows:

$$
\begin{aligned}
\mathit{reRel}\ \alpha\ v &= \mathit{reRel}(a(b+1))\ v \\
&= \mathit{CompRel}(\mathit{reRel}\ a\ v)(\mathit{reRel}\ (b+1)\ v) \\
&= \mathit{CompRel}\ R_a\ (\mathit{reRel}\ (b+1)\ v) \\
&= \mathit{CompRel}\ R_a\ (\mathit{UnionRel}\ (\mathit{reRel}\ b\ v)(\mathit{reRel}\ 1\ v)) \\
&= \mathit{CompRel}\ R_a\ (\mathit{UnionRel}\ R_b(\mathit{reRel}\ 1\ v)) \\
&= \mathit{CompRel}\ R_a\ (\mathit{UnionRel}\ R_b \mathit{IdRel}). \\
&(\ = R_a \circ (R_b \cup \mathcal{I})\ \ )
\end{aligned}
$$

# From Regular Expressions to Relations and back

$$\alpha \sim \beta \quad \rightarrow \quad \textbf{reRel}\ v\ \alpha \sim_{\mathcal{R}} \textbf{reRel}\ v\ \beta$$

This theorem allows for

- the design of a Coq tactic that transforms a goal of the form `reRel` $v\ \alpha \sim_{\mathcal{R}}$ `reRel` $v\ \beta$ into a goal stating that $\alpha \sim \beta$
- and then applies the tactic for regular expressions (in-)equivalence to close the proof.

# Table of Contents

# Kleene Algebra with tests

Kleene Algebra with tests (KAT): KA extended with a boolean algebra $(K, T, +, \cdot, ^\star, ^-, 0, 1)$ such that

- $(K, +, \cdot, ^\star, 0, 1)$ is a KA,
- $(T, +, \cdot, ^-, 0, 1)$ is a Boolean algebra
- $T \subseteq K$
- KAT satisfies the axioms of KA and the axioms of Boolean algebra, that is, the set of axioms (1–15) and the following ones, for $b, c, d \in T$:

$$bc = cb \tag{20}$$
$$b + (cd) = (b + c)(b + d) \tag{21}$$
$$\overline{b + c} = \overline{b}\,\overline{c} \tag{22}$$
$$b + \overline{b} = 1 \tag{23}$$
$$bb = b \tag{24}$$
$$b + 1 = 1 \tag{25}$$
$$b + 0 = b \tag{26}$$
$$\overline{bc} = \overline{b} + \overline{c} \tag{27}$$
$$b\overline{b} = 0 \tag{28}$$
$$\overline{\overline{b}} = b \tag{29}$$

# Why formalizing Kleene Algebra with tests?

- Tests embedded in expressions $\longrightarrow$ encoding of imperative program constructions
- KAT :
  - KAT subsumes (can encode) PHL;
  - Capture and verify properties of simple imperative programs. An equational way to deal with partial correctness and program equivalence.
- Consequently, proving that a given program $C$ is partially correct using the deductive system of PHL **can be reduced to** checking if $C$ is partially correct by equational reasoning in KAT.
- Moreover, some formulas of KAT **can be reduced to** standard equalities and the equalities can be decided automatically.

# KAT terms

- $\mathcal{B} = \{b_1, \ldots, b_n\}$: set of *primitive tests*
- $\overline{\mathcal{B}} = \{\overline{b} \mid b \in \mathcal{B}\}$.
- $l \in \mathcal{B} \cup \overline{\mathcal{B}}$ is called a *literal*.
- An *atom* $\alpha$ is a finite sequence of literals $l_1 l_2 \ldots l_n$, such that each $l_i$ is either $b_i$ or $\overline{b_i}$, for $1 \leq i \leq n$, where $n = |\mathcal{B}|$.
- At: set of atoms
- $\alpha \leq b \ \triangleq\ \alpha \to b$ is a propositional tautology (with $\alpha \in \text{At}$ and $b \in \mathcal{B}$,).
- tests are booleans expressions inductively defined by:
  - 0 and 1 are tests
  - if $b \in \mathcal{B}$ then $b$ is a test
  - if $t_1$ and $t_2$ are tests then $t_1 + t_2$, $t_1 \cdot t_2$, and $\overline{t_1}$ are tests
- KAT terms = KA terms (i.e. regular expressions) + tests, inductively defined by:
  - a test $t$ is a KAT term
  - if $p \in \Sigma$ then $p$ is a KAT term
  - if $e_1$ and $e_2$ are KAT terms, then so are $e_1 + e_2$, $e_1 e_2$, and $e_1^\star$.

# Guarded Strings

- A *guarded string* is a sequence $x = \alpha_0 p_0 \alpha_1 p_1 \ldots p_{(n-1)} \alpha_n$, with $\alpha_i \in \mathrm{At}$ and $p_i \in \Sigma$.

| Regular Languages | Language Theoretic Model of KAT |
|---|---|
| word | guarded string |
| regular expression | KAT Term |
| concatenation | fusion of compatible guarded string |
| Languages | set of guarded strings |

- $\epsilon_\alpha$ defined by induction: $\epsilon_\alpha(p) = \mathit{false}$, $\epsilon_\alpha(e^*) = \mathit{true}$, $\epsilon_\alpha(t) = \mathit{true}$ if $\alpha \leq t, \epsilon_\alpha(t) = \mathit{false}$ otherwise, $\epsilon_\alpha(e_1 + e_2) = \epsilon_\alpha(e_1) \vee \epsilon_\alpha(e_2)$, $\epsilon_\alpha(e_1 e_2) = \epsilon_\alpha(e_1) \wedge \epsilon_\alpha(e_2)$
- $E(e)$ is defined by $\{\alpha \in At \mid \epsilon_\alpha(e) = \mathit{true}\}$

# Kleene Algebra with tests

Let $\alpha p \in (\mathsf{At} \cdot \Sigma)$ and let $e$ be a KAT term. The *set* $\partial_{\alpha p}(e)$ *of partial derivatives* of $e$ with respect to $\alpha p$ is inductively defined by

$$\begin{array}{rcl}
\partial_{\alpha p}(t) & = & \emptyset \\
\partial_{\alpha p}(q) & = & \begin{cases} \{1\} & \textit{if } p \equiv q, \\ \emptyset & \textit{otherwise}. \end{cases} \\
\partial_{\alpha p}(e_1 + e_2) & = & \partial_{\alpha p}(e_1) \cup \partial_{\alpha p}(e_2) \\
\partial_{\alpha p}(e_1 e_2) & = & \begin{cases} \partial_{\alpha p}(e_1)e_2 \cup \partial_{\alpha p}(e_2) & \textit{if } \varepsilon_\alpha(e_1) = \mathtt{true}, \\ \partial_{\alpha p}(e_1)e_2, & \textit{otherwise}. \end{cases} \\
\partial_{\alpha p}(e^\star) & = & \partial_{\alpha p}(e)e^\star
\end{array}$$

KAT Partial derivatives for words $w \in (\mathsf{At} \cdot \Sigma)^\star$, inductively by $\partial_\epsilon(e) = \{e\}$, and $\partial_{w\alpha p}(e) = \partial_{\alpha p}(\partial_w(e))$.
The (proven finite) set of all partial derivatives of a KAT term is the set

$$\partial_{(\mathsf{At}\cdot\Sigma)^\star}(e) = \bigcup_{w \in (\mathsf{At}\cdot\Sigma)^\star} \{e' \mid e' \in \partial_w(e)\}$$

# An Example

**Example**

Let $\mathcal{B} = \{b_1, b_2\}$, $\Sigma = \{p, q\}$, and $e = b_1 p(b_1 + b_2)q$. The partial derivative of $e$ with respect to the sequence $b_1 b_2 p \overline{b_1} b_2 q$ is the following:

$$\begin{aligned}
\partial_{b_1 b_2 p \overline{b_1} b_2 q}(e) &= \partial_{b_1 b_2 p \overline{b_1} b_2 q}(b_1 p(b_1 + b_2)q) \\
&= \partial_{\overline{b_1} b_2 q}(\partial_{b_1 b_2 p}(b_1 p(b_1 + b_2)q)) \\
&= \partial_{\overline{b_1} b_2 q}(\partial_{b_1 b_2 p}(b_1)(p(b_1 + b_2)q) \cup \partial_{b_1 b_2 p}(p(b_1 + b_2)q)) \\
&= \partial_{\overline{b_1} b_2 q}(\partial_{b_1 b_2 p}(b_1)(p(b_1 + b_2)q)) \cup \partial_{\overline{b_1} b_2 q}(\partial_{b_1 b_2 p}(p(b_1 + b_2)q)) \\
&= \partial_{\overline{b_1} b_2 q}(\partial_{b_1 b_2 p}(p)(b_1 + b_2)q) \\
&= \partial_{\overline{b_1} b_2 q}((b_1 + b_2)q) \\
&= \partial_{\overline{b_1} b_2 q}(b_1 + b_2)q \cup \partial_{\overline{b_1} b_2 q}(q) \\
&= \partial_{\overline{b_1} b_2 q}(q) \\
&= \{1\}.
\end{aligned}$$

# A Procedure for KAT Terms Equivalence

Let $e$ be a KAT term,

$$e \sim \mathsf{E}(e) \cup \left( \bigcup_{\alpha p \in (\mathsf{At} \cdot \Sigma)} \alpha p \partial_{\alpha p}(e) \right).$$

Therefore, if $e_1$ and $e_2$ are KAT terms, we can reformulate the equivalence $e_1 \sim e_2$ as

$$\mathsf{E}(e_1) \cup \left( \bigcup_{\alpha p \in (\mathsf{At} \cdot \Sigma)} \alpha p \partial_{\alpha p}(e_1) \right) \sim \mathsf{E}(e_2) \cup \left( \bigcup_{\alpha p \in (\mathsf{At} \cdot \Sigma)} \alpha p \partial_{\alpha p}(e_2) \right),$$

which is tantamount at checking that $\forall \alpha \in \mathsf{At}$, $\varepsilon_\alpha(e_1) = \varepsilon_\alpha(e_2)$ and $\forall \alpha p \in (\mathsf{At} \cdot \Sigma)$, $\partial_{\alpha p}(e_1) \sim \partial_{\alpha p}(e_2)$ hold.

# A Procedure for KAT Terms Equivalence

We can finitely iterate over the previous equations and reduce the (in)equivalence of $e_1$ and $e_2$ to one of the next equivalences:

$$e_1 \sim e_2 \;\leftrightarrow\; \forall \alpha \in \mathsf{At}, \forall w \in (\mathsf{At} \cdot \Sigma)^\star, \varepsilon_\alpha(\partial_w(e_1)) = \varepsilon_\alpha(\partial_w(e_2)) \tag{30}$$

and

$$e_1 \not\sim e_2 \;\leftrightarrow\; (\exists\, w \,\exists\, \alpha,\ \varepsilon_\alpha(\partial_w(e_1)) \neq \varepsilon_\alpha(\partial_w(e_2))). \tag{31}$$

# The procedure equivKAT

**Require:** $S = \{(\{e_1\}, \{e_2\})\}$, $H = \emptyset$
**Ensure:** `true` or `false`

1: **procedure** EquivKAT($S$, $H$)
2:     **while** $S \neq \emptyset$ **do**
3:         $(\Gamma, \Delta) \leftarrow POP(S)$
4:         **for** $\alpha \in \mathsf{At}$ **do**
5:             **if** $\varepsilon_\alpha(\Gamma) \neq \varepsilon_\alpha(\Delta)$ **then**
6:                 **return** `false`
7:             **end if**
8:         **end for**
9:         $H \leftarrow H \cup \{(\Gamma, \Delta)\}$
10:         **for** $\alpha p \in (\mathsf{At} \cdot \Sigma)$ **do**
11:             $(\Lambda, \Theta) \leftarrow \partial_{\alpha p}(\Gamma, \Delta)$
12:             **if** $(\Lambda, \Theta) \notin H$ **then**
13:                 $S \leftarrow S \cup \{(\Lambda, \Theta)\}$
14:             **end if**
15:         **end for**
16:     **end while**
17: **return** `true`
18: **end procedure**

- lines 4-8 and 10-15 : exponential behavior
- Formally proved terminating and correct
- COQ tactic based on equivKAT

## Example

Let $\mathcal{B} = \{b\}$ and let $\Sigma = \{p\}$, are $e_1 = (pb)^\star p$ and $e_2 = p(bp)^\star$ equivalent?
Consider $s_0 = (\{(pb)^\star p\}, \{p(bp)^\star\})$, it is enough to show that equivKAT$(\{s_0\}, \emptyset) = \mathtt{true}$.
The first step of the computation generates the two new following pairs of derivatives:

$$\partial_{bp}(e_1, e_2) = (\{1, b(pb)^\star\}, \{(bp)^\star\}),$$
$$\partial_{\overline{b}p}(e_1, e_2) = (\{1, b(pb)^\star\}, \{(bp)^\star\}).$$

Then, $(e_1, e_2)$ is added to the historic set $H$ and the next iteration of equivKAT considers $S = \{s_1\}$, with $s_1 = (\{1, b(pb)^\star\}, \{(bp)^\star\})$, and $H = \{s_0\}$.

$$\partial_{bp}(\{1, b(pb)^\star\}, \{(bp)^\star\}) = (\{b(pb)^\star\}, \{(bp)^\star\}),$$
$$\partial_{\overline{b}p}(\{1, b(pb)^\star\}, \{(bp)^\star\}) = (\emptyset, \emptyset).$$

The next iteration of the procedure will have $S = \{s_2, s_3\}$ and $H = \{s_0, s_1\}$, with $s_2 = (\{b(pb)^\star\}, \{(bp)^\star\})$ and $s_3 = (\emptyset, \emptyset)$.
Since the derivative of $s_2$ is either $s_2$ or $s_3$ and since the same holds for the derivatives of $s_3$, the procedure will terminate in two iterations with $S = \emptyset$ and $H = \{s_0, s_1, s_2, s_3\}$. Hence, we conclude that $e_1 \sim e_2$.

# Table of Contents

# Program Equivalence

if $e_1$ and $e_2$ are terms encoding the IMP programs $C_1$ and $C_2$, and if the Boolean test $B$ is encoded by the KAT test $t$, then we can encode sequence, conditional instructions and while loops in KAT as follows.

$$
\begin{aligned}
C_1; C_2 &\triangleq e_1 e_2, \\
\text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi} &\triangleq (te_1 + \overline{t}e_2), \\
\text{while } B \text{ do } C_1 \text{ end} &\triangleq (te_1)^\star \overline{t}.
\end{aligned}
$$

## Example

Let $\mathcal{B} = \{b, c\}$ and $\Sigma = \{p, q\}$ be the set of primitive tests and set of primitive programs, respectively, and let $P1$ and $P2$ be the following two programs:

$$P_1 \triangleq \texttt{while}\, B\, \texttt{do}\ C_1;\texttt{while}\, B'\, \texttt{do}\ C_2\ \texttt{end}\ \texttt{end}$$
$$P_2 \triangleq \texttt{if}\, B\, \texttt{then}\ C_1;\texttt{while}\, B + B'\, \texttt{do}\ \texttt{if}\, B'\, \texttt{then}\, C_2\, \texttt{else}\, C_1\, \texttt{fi}\ \texttt{end}\, \texttt{else}\, \texttt{skip}\, \texttt{fi}$$

Let $C_1 = p$, $C_2 = q$, $B = b$ and $B' = c$. The programs $P_1$ and $P_2$ are encoded in KAT as

$$e_1 = (bp((cq)^\star\overline{c}))^\star\overline{b} \;\text{ and }\; e_2 = bp((b + c)(cq + \overline{c}p))^\star\overline{(b + c)} + \overline{b},$$

respectively. The procedure decides the equivalence $e_1 \sim e_2$ in 0.053 seconds.

# Program Correctness

This methodology can be extended in order to encode a non trivial subset of Hoare Logic and allows *classical* program verification based on contracts (pre-post condition, invariants).

# Table of Contents

# Main conclusions and results

- efficient procedure to decide regular expression equivalence ;
- able to solve equations involving relations ;
- a simple extension to decide KAT terms equivalence.
- Application to program verification, but mainly program equivalence
- Extraction to Caml

- Improve the performance of *equivKAT* in order to handle bigger (in)-equivalences (on-going work)
- Extension to Schematic Kleene Algebra with test (widening the actual HL coverage)
- Modal (and concurrent) Kleene Algebra (Equivalence for parallel or concurrent Programs, timing behavior)
- Embedding into program verification frameworks (why3, etc...)
- Application Runtime Verification (e.g. of Ada/Spark programs) (ongoing work)

Thank you!

supplementary slides

# Finiteness of Partial Derivatives

- Recursive definition of PD via *support* [Champarnaud and Ziadi]:

$$\begin{array}{rcl}
\pi(\emptyset) & = & \emptyset \\
\pi(\varepsilon) & = & \emptyset \\
\pi(a) & = & \{1\} \\
\pi(\alpha+\beta) & = & \pi(\alpha)\cup\pi(\beta) \\
\pi(\alpha\beta) & = & \pi(\alpha)\beta\cup\pi(\beta) \\
\pi(\alpha^{\star}) & = & \pi(\alpha)\alpha^{\star}
\end{array}$$

- Another way of looking at *PD*:

$$PD(\alpha) = \{\alpha\}\cup\pi(\alpha)$$

- Again, the upper bound of *PD*:

$$|\pi(\alpha)| \leq |\alpha|_{\Sigma}$$

$$|PD(\alpha)| \leq |\alpha|_{\Sigma} + 1$$