Deciding Kleene Algebra Terms (In-)Equivalence in Coq

Nelma Moreira, David Pereira and Simão Melo de Sousa

Tallinn
September 2016
Outline

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
Table of Contents

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
Kleene Algebra

Idempotent semiring

\((K, +, \cdot, 0, 1)\):

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

Consider \(x \leq y \triangleq x + y = y\).

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

1. \(1 + pp^* \leq p^*\)  \hspace{1cm} (12)
2. \(1 + p^*p \leq p^*\)  \hspace{1cm} (13)
3. \(q + pr \leq r \rightarrow p^*q \leq r\)  \hspace{1cm} (14)
4. \(q + rp \leq r \rightarrow qp^* \leq r\)  \hspace{1cm} (15)

Standard Model of KA: \((\mathbb{RL}_\Sigma, \cup, \cdot, *, \emptyset, \{\epsilon\})\)
Regular expressions and Languages

- Regular expression:

\[ \alpha, \beta ::= 0 \mid 1 \mid a \in \Sigma \mid \alpha + \beta \mid \alpha \beta \mid \alpha^* \]

- Language denoted by a regular expression:

\[
\begin{align*}
\mathcal{L}(0) &= \emptyset \\
\mathcal{L}(1) &= \{\epsilon\} \\
\mathcal{L}(a) &= \{a\} \\
\mathcal{L}(\alpha + \beta) &= \mathcal{L}(\alpha) \cup \mathcal{L}(\beta) \\
\mathcal{L}(\alpha \beta) &= \mathcal{L}(\alpha) \mathcal{L}(\beta) \\
\mathcal{L}(\alpha^*) &= \mathcal{L}(\alpha)^*
\end{align*}
\]

- Regular expression equivalence:

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

- Nullability:

\[ \varepsilon(\alpha) = \begin{cases} 
\text{true} & \text{if } \epsilon \in \mathcal{L}(\alpha) \\
\text{false} & \text{if } \epsilon \notin \mathcal{L}(\alpha)
\end{cases} \]
Definition of Partial Derivative wrt $a \in \Sigma$ [Mirkin, Antimirov]:

\[
\begin{align*}
\partial_a(0) &= \emptyset \\
\partial_a(1) &= \emptyset \\
\partial_a(b) &= \begin{cases} 
\{1\} & \text{if } a \equiv b \\
\emptyset & \text{otherwise}
\end{cases} \\
\partial_a(\alpha + \beta) &= \partial_a(\alpha) \cup \partial_a(\beta) \\
\partial_a(\alpha\beta) &= \begin{cases} 
\partial_a(\alpha)\beta \cup \partial_a(\beta) & \text{if } \varepsilon(\alpha) = \text{true}, \\
\partial_a(\alpha)\beta & \text{otherwise}.
\end{cases} \\
\partial_a(\alpha^*) &= \partial_a(\alpha)\alpha^*
\end{align*}
\]
Partial Derivatives (cont.)

- Partial Derivatives wrt Words:
  \[
  \partial_\varepsilon(\alpha) = \{\alpha\} \\
  \partial_{wa}(\alpha) = \partial_a(\partial_w(\alpha)).
  \]

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

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

- An interesting consequence: \(w \in \mathcal{L}(\alpha) \iff \varepsilon(\partial_w(\alpha)) = \text{true}\)

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

- Finiteness of \(PD\) [Mirkin,Antimirov]: \(PD(\alpha) \leq |\alpha|_{\Sigma} + 1\)
(In-)Equivalence Through Iterated Derivation

\[ \alpha \sim \varepsilon(\alpha) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\alpha)) \]  

(16)

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

\[ \varepsilon(\alpha) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\alpha)) \sim \varepsilon(\beta) \cup \bigcup_{a \in \Sigma} a(\sum \partial_a(\beta)) \]  

(17)

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

\[ (\forall w \in \Sigma^*, \varepsilon(\partial_w(\alpha)) = \varepsilon(\partial_w(\beta))) \leftrightarrow \alpha \sim \beta. \]  

(18)

\[ \varepsilon(\partial_w(\alpha)) \neq \varepsilon(\partial_w(\beta))) \rightarrow \alpha \not\sim \beta, \text{ for some } w \in \Sigma^*. \]  

(19)
The Procedure equivP

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

1: procedure EquivP\((S, H)\)
2: while \( S \neq \emptyset \) do
3: \((S_\alpha, S_\beta) \leftarrow \text{POP}(S)\)
4: if \( \varepsilon(S_\alpha) \neq \varepsilon(S_\beta) \) then
5: return false
6: end if
7: \( H \leftarrow H \cup \{ (S_\alpha, S_\beta) \} \)
8: for \( a \in \Sigma \) do
9: \((S'_\alpha, S'_\beta) \leftarrow \partial_a(S_\alpha, S_\beta)\)
10: if \( (S'_\alpha, S'_\beta) \notin H \) then
11: \( S \leftarrow S \cup \{ (S'_\alpha, S'_\beta) \} \)
12: end if
13: end for
14: end while
15: return true
16: 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
Consider $\alpha = (ab)^*a$ and $\beta = a(ba)^*$. Then $s_0 = (\{\alpha, \beta\}) = (\{(ab)^*a\}, \{a(ba)^*\})$. We must show that $\text{equivP}(\{s_0\}, \emptyset) = \text{true}$.

equivP for such $\alpha$ and $\beta$ computes $s_1 = (\{1, b(ab)^*a\}, \{(ba)^*\})$ and $s_2 = (\emptyset, \emptyset)$.

Execution traces:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$S_i$</th>
<th>$H_i$</th>
<th>$\text{drvs.}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${s_0}$</td>
<td>$\emptyset$</td>
<td>$\partial_a(s_0) = s_1, \partial_b(s_0) = s_2$</td>
</tr>
<tr>
<td>1</td>
<td>${s_1, s_2}$</td>
<td>${s_0}$</td>
<td>$\partial_a(s_1) = s_2, \partial_b(s_1) = s_0$</td>
</tr>
<tr>
<td>2</td>
<td>${s_2}$</td>
<td>${s_0, s_1}$</td>
<td>$\partial_a(s_2) = s_2, \partial_b(s_2) = s_2$</td>
</tr>
<tr>
<td>3</td>
<td>$\emptyset$</td>
<td>${s_0, s_1, s_2}$</td>
<td>$\text{true}$</td>
</tr>
</tbody>
</table>
Table of Contents

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
Ingredient 1: Representation of Derivatives

- Derivatives as dependent records:

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

Example (Original regular expression)

```plaintext
Definition Ddrv_1st (α β:re) : Ddrv 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 $\alpha$ $\beta$)(a:A) : Drv $\alpha$ $\beta$.

  `refine(match x with
    | mkDrv $\alpha$ $\beta$ $p$ $w$ $H$ ⇒
      mkDrv $\alpha$ $\beta$ (pdrvp $p$ a) ($w++[a]$) _
      end).

  abstract((* Proof of $\partial_a(\partial_w(\alpha),\partial_w(\beta)) = (\partial_{wa}(\alpha),\partial_{wa}(\beta))$ *))

  Defined.

- Derivation of Drv terms wrt a set of symbols:

  **Definition** Drv_pdrv_set(x:Drv $\alpha$ $\beta$)(Sig:set A) : set (Drv $\alpha$ $\beta$) :=

    fold (fun a:A ⇒ add (Drv_pdrv $\alpha$ $\beta$ x a)) Sig $\emptyset$.

- Ignoring already existing derivatives in $H$:

  **Definition** Drv_pdrv_set_filtered(x:Drv $\alpha$ $\beta$)

    $(H:set(Drv $\alpha$ $\beta$))(sig:set A):set (Drv $\alpha$ $\beta$) :=

    filter (fun y ⇒ negb (y $\in$ $H$)) (Drv_pdrv_set_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 α β) * 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
  
  \[
  \text{step } \alpha \beta H S = ((H', S'), \text{proceed } \alpha \beta)
  \]

  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**:

  ```plaintext
  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:

\[
\text{Definition equivP_aux}(\alpha \beta:\text{re})(H\ S:\text{set(Drv }\alpha \beta)) (\text{sig:} \text{set } A)(D:\text{DP }\alpha \beta \ H \ S) := \\
\text{let } H' := \text{iterate } \alpha \beta \ H \ S \ \text{sig } D \ \text{in} \\
\text{match } H' \text{ with} \\
\mid \text{Ok } \_ \Rightarrow \text{true} \\
\mid \text{NotOk } \_ \Rightarrow \text{false} \\
\text{end.}
\]

- instantiate with the correct arguments:

\[
\text{Definition equivP } (\alpha \beta:\text{re}) := \\
\text{equivP_aux} \alpha \beta \emptyset \{\text{Drv}_1\text{st }\alpha \beta\} (\text{setSy } \alpha \cup \text{setSy } \beta) \\
(\text{mkDP}_\text{ini } \alpha \beta).
\]
Correctness

Lemma equiv_re_false :
\[ \forall \alpha \beta, \text{equivP} \alpha \beta = \text{false} \rightarrow \alpha \not\sim \beta \]

1. this only happens when :

\[ \text{iterate } H S = \text{NotOk } \alpha \beta (S_\alpha, S_\beta) \]

2. which means that:

\[ \text{step } H' S' = \text{termfalse } \alpha \beta (S_\alpha, S_\beta) \]

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 :
\[ \forall \alpha \, \beta, \text{equivP } \alpha \, \beta = \text{true} \rightarrow \alpha \sim \beta \]

1. define the following invariant:

\[ INV(H, S) = \text{def } \forall x, \ x \in H \rightarrow \forall a \in \Sigma, \ \partial_a(x) \in S \cup H \]

2. prove that it holds for step:

\[ INV(H, S) \rightarrow \text{step } H S = ((H', S'), \text{proceed}) \rightarrow INV(H', S') \]

3. prove that all derivatives are computed :

\[ INV(H, S) \rightarrow \text{iterate } H S = \text{Ok } \_ \_ \_ H' \rightarrow INV(H', \emptyset) \]

4. prove that all derivatives \((S_\alpha, S_\beta)\) verify \(\varepsilon(S_\alpha) = \varepsilon(S_\beta)\)

5. thus we obtain \(\forall w \in \Sigma^*, \varepsilon(\partial_w(\alpha)) = \varepsilon(\partial_w(\beta))\)

6. from which follows \(\alpha \sim \beta\)
Completeness

Obtained by trivial case analysis:

▶ $\alpha \sim \beta$:
  1. if $\text{equivP } \alpha \beta = \text{true}$: trivial from correctness proof;
  2. if $\text{equivP } \alpha \beta = \text{false}$: 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
   | |- \( L(?R1) \sim L(?R2) \) \implies \text{re-equiv}
   | |- \( L(?R1) \not\sim L(?R2) \) \implies \text{re-inequiv}
   | |- \( L(?R1) \leq L(?R2) \) \implies
     unfold lleq;change (\( L(R1) \cup L(R2) \)) with (\( L(R1 + R2) \));
     re_equiv
   | |- _ \implies fail 2 "Not a regular expression (in-)equivalence equation."
   end.
Table of Contents

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
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)

<table>
<thead>
<tr>
<th>alg.</th>
<th>(20, 200)</th>
<th>(50, 500)</th>
<th>(50, 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eq</td>
<td>ineq</td>
<td>eq</td>
</tr>
<tr>
<td>equivP</td>
<td>2.211</td>
<td>0.048</td>
<td>9.957</td>
</tr>
<tr>
<td>ATBR</td>
<td>3.001</td>
<td>1.654</td>
<td>5.876</td>
</tr>
</tbody>
</table>

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

Regular expression generated using the FAdo toolbox:
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression (In-)Equivalence</td>
</tr>
<tr>
<td>Implementation in Coq</td>
</tr>
<tr>
<td>Experimental Results</td>
</tr>
<tr>
<td>Deciding Relation Algebra Equations</td>
</tr>
<tr>
<td>(In-)Equivalence of KAT terms</td>
</tr>
<tr>
<td>Applications</td>
</tr>
<tr>
<td>Conclusions and Future Work</td>
</tr>
</tbody>
</table>
Claim: Equations over relation can be decided using regular expressions

First ingredient:

```plaintext
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 \( \alpha = a(b + 1) \)
- \( \nu \): a function that maps \( a \) to the relation \( R_a \), and \( b \) to the relation \( R_b \).
- The computation of \( \text{reRel} \alpha \nu \) gives the relation
  \( R_a \circ (R_b \cup I) \), and can be described as follows:

\[
\text{reRel} \alpha \nu = \text{reRel}(a(b + 1)) \nu \\
= \text{CompRel}(\text{reRel} a \nu)(\text{reRel} (b + 1) \nu) \\
= \text{CompRel} R_a (\text{reRel} (b + 1) \nu) \\
= \text{CompRel} R_a (\text{UnionRel} (\text{reRel} b \nu)(\text{reRel} 1 \nu)) \\
= \text{CompRel} R_a (\text{UnionRel} R_b(\text{reRel} 1 \nu)) \\
= \text{CompRel} R_a (\text{UnionRel} R_b \text{IdRel}).
\]

( \( = R_a \circ (R_b \cup I) \) )
From Regular Expressions to Relations and back

\[ \alpha \sim \beta \quad \rightarrow \quad \text{reRel} \lor \alpha \sim_{\mathcal{R}} \text{reRel} \lor \beta \]

This theorem allows for

- the design of a Coq tactic that transforms a goal of the form \( \text{reRel} \lor \alpha \sim_{\mathcal{R}} \text{reRel} \lor \beta \) 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

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
Kleene Algebra with tests

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

- \((K, +, \cdot, *, 0, 1)\) 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 \(b, c, d \in T\):

\[
\begin{align*}
bc &= cb \\
\quad & (20) \\
b + (cd) &= (b + c)(b + d) \\
& (21) \\
b + c &= \overline{bc} \\
& (22) \\
b + \overline{b} &= 1 \\
& (23) \\
b &= \overline{bb} = b \\
& (24) \\
b + 1 &= 1 \\
& (25) \\
b + 0 &= b \\
& (26) \\
\overline{bc} &= \overline{b} + \overline{c} \\
& (27) \\
\overline{bb} &= 0 \\
& (28) \\
\overline{b} &= b \\
& (29)
\end{align*}
\]
Why formalizing Kleene Algebra with tests?

- Tests embedded in expressions $\rightarrow$ 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 $n = |\mathcal{B}|$.
- $\text{At}$: set of atoms
- $\alpha \leq b \triangleq \alpha \rightarrow b$ is a propositional tautology (with $\alpha \in \text{At}$ and $b \in \mathcal{B}$).

- tests are boolean 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 $= \text{KA terms}$ (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 $e_1^*$. 
Guarded Strings

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

<table>
<thead>
<tr>
<th>Regular Languages</th>
<th>Language Theoretic Model of KAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>guarded string</td>
</tr>
<tr>
<td>regular expression</td>
<td>KAT Term</td>
</tr>
<tr>
<td>concatenation</td>
<td>fusion of compatible guarded string</td>
</tr>
<tr>
<td>Languages</td>
<td>set of guarded strings</td>
</tr>
</tbody>
</table>

- $\epsilon_\alpha$ defined by induction: $\epsilon_\alpha(p) = false$, $\epsilon_\alpha(e^*) = true$, $\epsilon_\alpha(t) = true$ if $\alpha \leq t$, $\epsilon_\alpha(t) = false$ otherwise, $\epsilon_\alpha(e_1 + e_2) = \epsilon_\alpha(e_1) \lor \epsilon_\alpha(e_2)$, $\epsilon_\alpha(e_1 e_2) = \epsilon_\alpha(e_1) \land \epsilon_\alpha(e_2)$

- $E(e)$ is defined by $\{\alpha \in \text{At} \mid \epsilon_\alpha(e) = true\}$
Kleene Algebra with tests

Let \( \alpha p \in (\text{At} \cdot \Sigma) \) 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

\[
\begin{align*}
\partial_{\alpha p}(t) &= \emptyset \\
\partial_{\alpha p}(q) &= \begin{cases} 
\{1\} & \text{if } p \equiv q, \\
\emptyset & \text{otherwise}. 
\end{cases} \\
\partial_{\alpha p}(e_1 + e_2) &= \partial_{\alpha p}(e_1) \cup \partial_{\alpha p}(e_2) \\
\partial_{\alpha p}(e_1 e_2) &= \begin{cases} 
\partial_{\alpha p}(e_1)e_2 \cup \partial_{\alpha p}(e_2) & \text{if } \varepsilon_{\alpha}(e_1) = \text{true}, \\
\partial_{\alpha p}(e_1)e_2, & \text{otherwise}. 
\end{cases} \\
\partial_{\alpha p}(e^*) &= \partial_{\alpha p}(e)e^*
\end{align*}
\]

KAT Partial derivatives for words \( w \in (\text{At} \cdot \Sigma)^* \), inductively by \( \partial_{\varepsilon}(e) = \{e\} \), 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

\[
\partial_{(\text{At} \cdot \Sigma)^*}(e) = \bigcup_{w \in (\text{At} \cdot \Sigma)^*} \{e' \mid e' \in \partial_w(e)\}
\]
An Example

Example
Let $B = \{b_1, b_2\}$, $\Sigma = \{p, q\}$, and $e = b_1p(b_1 + b_2)q$. The partial derivative of $e$ with respect to the sequence $b_1b_2p\overline{b_1}b_2q$ is the following:

$$
\partial_{b_1b_2p\overline{b_1}b_2q}(e) = \partial_{b_1b_2p\overline{b_1}b_2q}(b_1p(b_1 + b_2)q)
= \partial_{\overline{b_1}b_2q}(\partial_{b_1b_2p}(b_1p(b_1 + b_2)q))
= \partial_{\overline{b_1}b_2q}(\partial_{b_1b_2p}(b_1)(p(b_1 + b_2)q) \cup \partial_{b_1b_2p}(p(b_1 + b_2)q))
= \partial_{\overline{b_1}b_2q}(\partial_{b_1b_2p}(b_1)(p(b_1 + b_2)q)) \cup \partial_{\overline{b_1}b_2q}(\partial_{b_1b_2p}(p(b_1 + b_2)q))
= \partial_{\overline{b_1}b_2q}(\partial_{b_1b_2p}(p)(b_1 + b_2)q)
= \partial_{\overline{b_1}b_2q}((b_1 + b_2)q)
= \partial_{\overline{b_1}b_2q}(b_1 + b_2)q \cup \partial_{\overline{b_1}b_2q}(q)
= \partial_{\overline{b_1}b_2q}(q)
= \{1\}.

$$
A Procedure for KAT Terms Equivalence

Let \( e \) be a KAT term,

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

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

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

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

$$e_1 \sim e_2 \iff \forall \alpha \in \text{At}, \forall w \in (\text{At} \cdot \Sigma)^*, \varepsilon_\alpha(\partial_w(e_1)) = \varepsilon_\alpha(\partial_w(e_2))$$ (30)

and

$$e_1 \not\sim e_2 \iff (\exists w \exists \alpha, \varepsilon_\alpha(\partial_w(e_1)) \neq \varepsilon_\alpha(\partial_w(e_2))).$$ (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:    \((\Gamma, \Delta) \leftarrow \text{POP}(S)\)
4:    for \( \alpha \in \text{At} \) 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 \( \alpha p \in (\text{At} \cdot \Sigma) \) 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 $B = \{b\}$ and let $\Sigma = \{p\}$, are $e_1 = (pb)^* p$ and $e_2 = p(bp)^*$ equivalent? Consider $s_0 = ((pb)^* p, \{p(bp)^*\})$, it is enough to show that \(\text{equivKAT}([s_0], \emptyset) = \text{true}\).

The first step of the computation generates the two new following pairs of derivatives:

\[
\partial_{bp}(e_1, e_2) = (\{1, b(pb)^*\}, (bp)^*)
\]

\[
\partial_{bp}(e_1, e_2) = (\{1, b(pb)^*\}, (bp)^*)
\]

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

\[
\partial_{bp}(\{1, b(pb)^*\}, (bp)^*) = (\{b(pb)^*\}, (bp)^*)
\]

\[
\partial_{bp}(\{1, b(pb)^*\}, (bp)^*) = (\emptyset, \emptyset)
\]

The next iteration of the procedure will have $S = \{s_2, s_3\}$ and $H = \{s_0, s_1\}$, with $s_2 = (\{b(pb)^*\}, (bp)^*)$ 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

Regular Expression (In-)Equivalence

Implementation in Coq

Experimental Results

Deciding Relation Algebra Equations

(In-)Equivalence of KAT terms

Applications

Conclusions and Future Work
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.

\[
C_1; C_2 \triangleq e_1 e_2,
\]

if $B$ then $C_1$ else $C_2$ fi $\triangleq (te_1 + \bar{t}e_2)$,

while $B$ do $C_1$ end $\triangleq (te_1)^*\bar{t}$. 
Example
Let $B = \{b, c\}$ and $\Sigma = \{p, q\}$ be the set of primitive tests and set of primitive programs, respectively, and let $P_1$ and $P_2$ be the following two programs:

$$P_1 \triangleq \text{while } B \text{ do } C_1; \text{while } B' \text{ do } C_2 \text{ end end}$$

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

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

$$e_1 = (bp((cq)^*c))^*\bar{b} \quad \text{and} \quad e_2 = bp((b + c)(cq + \bar{c}p))^*(b + c) + \bar{b},$$

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 \textit{classical} program verification based on contracts (pre-post condition, invariants).
# Table of Contents

- Regular Expression (In-)Equivalence
- Implementation in Coq
- Experimental Results
- Deciding Relation Algebra Equations
- (In-)Equivalence of KAT terms
- Applications

**Conclusions and Future Work**
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]:
  \[
  \begin{align*}
  \pi(\emptyset) &= \emptyset \\
  \pi(\varepsilon) &= \emptyset \\
  \pi(a) &= \{1\} \\
  \pi(\alpha + \beta) &= \pi(\alpha) \cup \pi(\beta) \\
  \pi(\alpha\beta) &= \pi(\alpha)\beta \cup \pi(\beta) \\
  \pi(\alpha^*) &= \pi(\alpha)\alpha^*
  \end{align*}
  \]

- Another way of looking at PD:
  \[PD(\alpha) = \{\alpha\} \cup \pi(\alpha)\]

- Again, the upper bound of PD:
  \[
  |\pi(\alpha)| \leq |\alpha|_\Sigma \\
  |PD(\alpha)| \leq |\alpha|_\Sigma + 1
  \]