Some Results on (Synchronous) Kleene Algebra with Tests

Sabine Broda
António Machiavelo
Nelma Moreira
Rogério Reis
Ricardo Almeida
Sílvia Cavadas

Faculty of Sciences, University of Porto, Portugal

1 Computer Science Department & CMUP
2 Mathematics Department & CMUP
3 Project-Grants (CANTE & AVIACC)
**Kleene Algebra with Tests (KAT)**

- extends Kleene algebra, the algebra of regular expressions, by combining it with Boolean algebra;
- addition of tests allows to express imperative program constructions;
- equational system suitable for propositional program verification (program equivalence, partial correctness, subsumes propositional Hoare logic).
KAT expressions

- $P = \{p_1, \ldots, p_k\}$ set of program symbols
- $T = \{t_1, \ldots, t_l\}$ set of test symbols

$B\text{Exp} : \quad b \rightarrow 0 \mid 1 \mid t \mid \neg b \mid b + b \mid b \cdot b$

$E\text{Exp} : \quad e \rightarrow p \mid b \mid e + e \mid e \cdot e \mid e^*$
Encoding Programs in KAT
(a simple while language)

\[
x := v
\]

\[
\text{skip}
\]

\[
P_1; P_2
\]

\[
\text{if } b \text{ then } P_1 \text{ else } P_2
\]

\[
\text{while } b \text{ do } P_1
\]

primitive symbol \( p \)

distinguished primitive symbol \( p_{\text{skip}} \)

\( e_1 e_2 \)

\( b e_1 + \overline{b} e_2 \).

\( (b e_1)^* \overline{b} \).
Encoding Programs in KAT

\[ P_1 : \]
while t1 do (p1; while t2 do p2)

\[ P_2 : \]
if t1 then (p1; while (t1+t2) do (if t2 then p2 else p1))

\[ e_1 = (t_1 p_1 (t_2 p_2) \land \neg t_2) \land \neg t_1 \]

\[ e_2 = t_1 p_1 ((t_1 + t_2)(t_2 p_2 + \neg t_2 p_1)) \land \neg (t_1 + t_2) + \neg t_1 \]

Equivalent programs/expressions?
Hoare Logic and KAT

- Hoare logic uses partial correctness assertions (PCA’s) to reason about program correctness;

- A PCA is a triple \( \{b\}P\{c\} \) meaning, “if \( b \) holds before the execution of \( P \), and if \( P \) halts, then \( c \) will necessarily hold at the end of the execution of \( P \)”;

- The propositional fragment of Hoare logic (PHL) can be encoded in KAT;

- A PCA \( \{b\}P\{c\} \) is encoded as \( be = bec \) or equivalently by \( be\bar{c} = 0 \), where \( e \) encodes \( P \).
Inference Rules for Hoare Logic

\[
\frac{b \rightarrow c}{\{b\} \textbf{skip} \{c\}} \quad \frac{b \rightarrow c[x/e]}{\{b\} x := e \{c\}}
\]

\[
\frac{\{b\} P \{c\} \quad \{c\} Q \{d\}}{\{b\} P; \{c\} Q \{d\}} \quad \frac{\{b \land c\} P \{d\} \quad \{\neg b \land c\} Q \{d\}}{\{c\} \textbf{if} \ b \textbf{then} \ P \textbf{else} \ Q \{d\}}
\]

\[
\frac{\{b \land i\} P \{i\} \quad c \rightarrow i \quad (i \land \neg b) \rightarrow d}{\{c\} \textbf{while} \ b \textbf{do} \ \{i\} P \{d\}}
\]
Generating a Set of Assumptions from a PCA \( \{b\}P\{c\} \) (in [1])

\[
\begin{align*}
\text{Gen}(b \, p_{\text{skip}} \, \overline{c}) & = \{b \leq c\} \\
\text{Gen}(b \, p \, \overline{c}) & = \{b \, p \, \overline{c}\} \quad \text{if } p_{\text{skip}} \neq p \in \Sigma \\
\text{Gen}(b \, e_1 \, c \, e_2 \, \overline{d}) & = \text{Gen}(b \, e_1 \, \overline{c}) \cup \text{Gen}(c \, e_2 \, \overline{d}) \\
\text{Gen}(b \, (c e_1 + \overline{c} e_2) \, \overline{d}) & = \text{Gen}(b c \, e_1 \, \overline{d}) \cup \text{Gen}(b \overline{c} \, e_2 \, \overline{d}) \\
\text{Gen}(b \, ((c i e) \ast \overline{c}) \, \overline{d}) & = \text{Gen}(i c \, e \, \overline{i}) \cup \{b \leq i, i \overline{c} \leq d\}
\end{align*}
\]

\[\Gamma = \{b_1 p_1 b'_1 = 0, \ldots, b_m p_m b'_m = 0\} \cup \{c_1 \leq c'_1, \ldots, c_n \leq c'_n\},\]

where \(p_1, \ldots, p_m \in \Sigma\) and \(b_i, c_i \in \text{Bexp} \).
# A Small Example

<table>
<thead>
<tr>
<th>Program $P$</th>
<th>Annotated Program $P'$</th>
<th>Symbols used in the encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y := 1;$</td>
<td>$y := 1;$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>$z := 0;$</td>
<td>$z := 0;$</td>
<td>$t_1$</td>
</tr>
<tr>
<td>while $\neg z = x$ do</td>
<td>while $\neg z = x$ do</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>{</td>
<td>$p_2$</td>
</tr>
<tr>
<td>$z := z+1;$</td>
<td>$z := z+1;$</td>
<td>$t_2$</td>
</tr>
<tr>
<td>$y := y \times z;$</td>
<td>$y := y \times z;$</td>
<td>$t_3$</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>$p_3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$t_4$</td>
</tr>
<tr>
<td>${y = z!}$</td>
<td>${y = z!}$</td>
<td>$p_4$</td>
</tr>
<tr>
<td>$y := y \times z;$</td>
<td>$y := y \times z;$</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>${\text{True}}$ $P'$ ${y = x!}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using the correspondence of KAT primitive symbols and atomic parts of the annotated program $P'$, as in the table and additionally encoding True as $t_0$ and $y = x!$ as $t_5$, respectively, the encoding of $\{\text{True}\} P' \{y = x!\}$ in KAT is

$$t_0p_1t_1p_2t_2(t_3t_2p_3t_4p_4)^*t_3 \bar{t}_5 = 0$$

The corresponding set of assumptions $\Gamma$ is

$$\Gamma = \{t_0p_1t_1 = 0, t_1p_2t_2 = 0, t_2t_3p_3t_4 = 0, t_4p_4t_2 = 0, t_2 \leq t_2, t_2 \bar{t}_3 \leq t_5\}$$
Deciding Equivalence Modulo a Set of Assumptions

It has been shown (Kozen’00), that for all KAT expressions $r_1, \ldots, r_n, e_1, e_2$ over $\Sigma = \{p_1, \ldots, p_k\}$ and $T = \{t_1, \ldots, t_l\}$, an implication of the form

$$r_1 = 0 \land \cdots \land r_n = 0 \rightarrow e_1 = e_2$$

is a theorem of KAT if and only if

$$e_1 + uru = e_2 + uru$$

where $u = (p_1 + \cdots + p_k)^*$ and $r = r_1 + \ldots + r_n$.

For the factorial program this is equivalent to proving

$$t_0p_1t_1p_2t_2(t_3t_2p_3t_4p_4)^*t_3t_5 + uru = 0 + uru,$$

where $u = (p_1 + p_2 + p_3 + p_4)^*$ and $r = t_0p_1t_1 + t_1p_2t_2 + t_3t_2p_3t_4 + t_4p_2t_2 + t_2t_3t_5$. 
we were particularly interested in ...

- transferring and extending classical results and techniques for regular expressions to KAT;
- compact representations of KAT expressions by (non-)deterministic automata;
- feasible algorithms for checking equivalence of KAT expressions.
The standard language theoretic model of KAT: Guarded Strings over \( P \) and \( T \)

\[
\text{At} = \{x_1 \cdots x_l \mid x_i \in \{t_i, \bar{t}_i\}, \ t_i \in T\}
\]

set of all truth assignments to \( T \)

\[
\text{GS} = (\text{At} \cdot P)^* \cdot \text{At}
\]

set of guarded strings over \( P \) and \( T \)

\[
\alpha_1 p_1 \alpha_2 p_2 \cdots p_{n-1} \alpha_n \in \text{GS}.
\]

\[
X \odot Y = \{x\alpha y \mid x\alpha \in X, \alpha y \in Y\}
\]

\[
X^0 = \text{At}
\]

\[
X^{n+1} = X \odot X^n
\]
The language theoretic model of KAT (cont.)

every \( e \in \text{Exp} \) denotes a set \( \text{GS}(e) \subseteq \text{GS} \)

\[
\begin{align*}
\text{GS}(p) & = \{ \alpha p \beta \mid \alpha, \beta \in \text{At} \} \\
\text{GS}(b) & = \{ \alpha \mid \alpha \in \text{At} \land \alpha \leq b \} \\
\text{GS}(e_1 + e_2) & = \text{GS}(e_1) \cup \text{GS}(e_2) \\
\text{GS}(e_1 \cdot e_2) & = \text{GS}(e_1) \diamond \text{GS}(e_2) \\
\text{GS}(e_1^*) & = \bigcup_{n \geq 0} \text{GS}(e_1)^n,
\end{align*}
\]

where \( \alpha \leq b \) if \( \alpha \rightarrow b \) is a propositional tautology.

\[
e_1 = e_2 \quad \text{iff} \quad \text{GS}(e_1) = \text{GS}(e_2)
\]
Example:

Consider \( e = t_1p(pq^*t_2 + t_3q)^* \)

where \( P = \{p, q\} \) and \( T = \{t_1, t_2, t_3\} \),

and

\[ At = \{t_1t_2t_3, t_1t_2t_3, t_1t_2t_3, t_1t_2t_3, t_1t_2t_3, t_1t_2t_3, t_1t_2t_3, t_1t_2t_3 \} \]

We have for instance,

\[ t_1\overline{t_2t_3} p t_1t_2t_3 q t_1\overline{t_2t_3} \in GS(e) \]
Automata for guarded strings

\[ A = \langle S, s_0, o, \delta \rangle \]

\[ o(e_0) = 0, \ o(e_1) = 1, \ o(e_2) = t_2 \]

\[ \delta = \{(e_0, (t_1, p), e_1), (e_1, (1, p), e_2), \ldots\} \]

\[ t_1 \bar{t}_2 t_3 \ p \ t_1 t_2 t_3 \ q \ t_1 \bar{t}_2 \bar{t}_3 \in \text{GS}(A) \]
Automata for Guarded Strings and KAT expression equivalence

- In [1] an derivative based algorithm to decide the equivalence of KAT expressions, as well as an algorithm for deciding equivalence, modulo a set of assumptions, were presented;

- In [2] Mirkin’s construction for regular expressions was adapted to obtain an Equation automaton for KAT expressions (avoiding the exponential blow-up on the number of states/ transitions due to the presence of truth-assignments);

- The state complexity of the Equation automaton was shown to be, on average and asymptotically, a quarter of the size of the original KAT expression (and half the size of another construction - the Glushkov automaton).
Automata for Guarded Strings and KAT Expression Equivalence

- in [3] the classical subset construction for determinizing nondeterministic finite automata was adapted to KAT;

- generalisation of the Hopcroft & Karp algorithm for testing deterministic finite automata equivalence to KAT [3].

- decision procedure for testing KAT equivalence without explicitly constructing the automata, by introducing a new notion of partial derivative [3].
Synchronous Kleene Algebra (with Tests)  
SKA & SKAT

- SKA is a decidable framework that combines Kleene Algebra with a synchrony model of concurrency (Prisacariu’10);

- elements of SKA can be seen as processes taking place within a fixed discrete time frame;

- at each time frame they may execute one or more basic actions or then come to a halt.

- the extension Synchronous Kleene Algebra with Tests (SKAT) combines SKA with a boolean algebra.
Let $A_B$ be a set of basic actions, then the set of SKA expressions contains 0 plus all terms generated by the following grammar

$$\alpha \rightarrow 1 | a | \alpha + \alpha | \alpha \cdot \alpha | \alpha \times \alpha | \alpha^* \quad (a \in \text{SKA})$$

Each SKA expression defines a set of words (regular language) over the alphabet

$$\Sigma = \mathcal{P}(A_B) \setminus \{\emptyset\}$$

where the synchronous product of two words $x = \sigma_1 \ldots \sigma_m$ and $y = \tau_1 \ldots \tau_n$, with $n \geq m$, is defined by

$$x \times y = y \times x = (\sigma_1 \cup \tau_1 \ldots \sigma_m \cup \tau_m)\tau_{m+1} \ldots \tau_n.$$

**Example:** Let $A_B = \{a, b\}$, hence $\Sigma = \{\{a\}, \{b\}, \{a, b\}\}$. For $x = \{a\}\{a, b\}\{b\}$ and $y = \{b\}\{a\}\{a, b\}\{a\}\{a, b\}$, we have

$$x \times y = \{a, b\}\{a, b\}\{a, b\}\{a\}\{a, b\}.$$
• SKAT is the natural extension of KAT to the synchronous setting (Prisacariu’10);

• its standard models are sets over guarded synchronous strings (GSS);

• Prisacariu defined automata for GSS, built in two layers: one to process a synchronous string and another to represent the valuations of the booleans.
Contributions to SKA(T)

• in [4]: definition of a partial derivative automaton for SKA;

• new decision procedure for SKA terms equivalence;

• definition of a simple notion of automaton for SKAT;

• extension of the derivative based methods developed for SKA to SKAT.
References

• Almeida, Broda, Moreira; Deciding KAT and Hoare Logic with Derivatives, GANDALF 2012.

• Broda, Machiavelo, Moreira, Reis; On the average size of Glushkov and Equation Automata for KAT expressions, FCT 2013.

• Broda, Machiavelo, Moreira, Reis; On the Equivalence of Automata for KAT expressions, CiE 2014.

• Broda, Cavadas, Moreira, Deciding Synchronous Kleene Algebra with Derivatives, FoSSaCS 2015 (submitted).
Thank You!