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

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

$$\begin{array}{lccl} \mathsf{BExp:} & b & \to & 0 \mid 1 \mid t \mid \neg b \mid b + b \mid b \cdot b \\ \mathsf{Exp:} & e & \to & p \mid b \mid e + e \mid e \cdot e \mid e^\star \end{array}$$

Encoding Programs in KAT
(a simple while language)

$x := v$	primitive symbol p
$\texttt{skip}$	distinguished primitive symbol $p_{\texttt{skip}}$
$P_1; P_2$	$e_1 e_2$
if b **then** P_1 **else** P_2	$be_1 + \bar{b}e_2$
while b **do** P_1	$(be_1)^*\bar{b}$

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))
```

$$e_1 = (t_1 p_1 (t_2 p_2)^\star \neg t_2)^\star \neg t_1$$

$$e_2 = t_1 p_1 ((t_1 + t_2)(t_2 p_2 + \neg t_2 p_1))^\star \neg (t_1 + t_2) + \neg t_1$$

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

$$\frac{b \to c}{\{b\}\ \texttt{skip}\ \{\texttt{c}\}}$$

$$\frac{b \to c[x/e]}{\{b\}\ x := e\ \{c\}}$$

$$\frac{\{b\}\ P\ \{c\} \qquad \{c\}\ Q\ \{d\}}{\{b\}\ P; \{c\}\ Q\ \{d\}}$$

$$\frac{\{b \wedge c\}\ P\ \{d\} \qquad \{\neg b \wedge c\}\ Q\ \{d\}}{\{c\}\ \texttt{if}\ b\ \texttt{then}\ P\ \texttt{else}\ Q\ \{d\}}$$

$$\frac{\{b \wedge i\}\ P\ \{i\} \quad c \to i \quad (i \wedge \neg b) \to d}{\{c\}\ \texttt{while}\ b\ \texttt{do}\ \{i\} P\ \{d\}}$$

Generating a Set of Assumptions from a PCA {b}P{c} (in [1])

$$
\begin{array}{lcl}
\texttt{Gen}(\texttt{b}\ \texttt{p}_{\textbf{skip}}\ \overline{\texttt{c}}) & = & \{b \leq c\} \\
\texttt{Gen}(\texttt{b}\ \texttt{p}\ \overline{\texttt{c}}) & = & \{b\ p\ \overline{c}\} \qquad\qquad p_{\textbf{skip}} \neq p \in \Sigma \\
\texttt{Gen}(\texttt{b}\ \texttt{e}_1\ \texttt{c}\ \texttt{e}_2\ \overline{\texttt{d}}) & = & \texttt{Gen}(\texttt{b}\ \texttt{e}_1\ \overline{\texttt{c}}) \cup \texttt{Gen}(\texttt{c}\ \texttt{e}_2\ \overline{\texttt{d}}) \\
\texttt{Gen}(\texttt{b}\ (\texttt{ce}_1 + \overline{\texttt{c}}\texttt{e}_2)\ \overline{\texttt{d}}) & = & \texttt{Gen}(\texttt{bc}\ \texttt{e}_1\ \overline{\texttt{d}}) \cup \texttt{Gen}(\texttt{b}\overline{\texttt{c}}\ \texttt{e}_2\ \overline{\texttt{d}}) \\
\texttt{Gen}(\texttt{b}\ ((\texttt{cie})^*\overline{\texttt{c}})\ \overline{\texttt{d}}) & = & \texttt{Gen}(\texttt{ic}\ \texttt{e}\ \overline{\texttt{i}}) \cup \{\texttt{b} \leq \texttt{i}, \texttt{i}\overline{\texttt{c}} \leq \texttt{d}\}
\end{array}
$$

$$\Gamma = \{b_1 p_1 \overline{b_1'} = 0, \ldots,\ b_m p_m \overline{b_m'} = 0\} \cup \{c_1 \leq c_1', \ldots, c_n \leq c_n'\},$$

where $p_1, \ldots, p_m \in \Sigma$ and $b_i, c_i \in \texttt{Bexp}$.

A Small Example

Program P	Annotated Program P'	Symbols used in the encoding
	$y := 1;$	p_1
	$\{y = 0!\}$	t_1
$y := 1;$	$z := 0;$	p_2
$z := 0;$	$\{y = z!\}$	t_2
while $\neg z = x$ do	while $\neg z = x$ do	t_3
{	{	
z := z+1;	{y=z!}	t_2
y := y×z;	z := z+1;	p_3
}	{y×z = z!}	t_4
	y := y×z;	p_4
	}	

$\{\texttt{True}\}\ \mathrm{P}'\ \{y = x!\}$

A Small Example (cont.)

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 $\{$`True`$\}$ P' $\{y = x!\}$ in KAT is

$$t_0 p_1 t_1 p_2 t_2 (t_3 t_2 p_3 t_4 p_4)^* \overline{t_3}\ \overline{t_5} = 0$$

The corresponding set of assumptions Γ is

$$\Gamma = \{t_0 p_1 \overline{t_1} = 0, t_1 p_2 \overline{t_2} = 0, t_2 t_3 p_3 \overline{t_4} = 0, t_4 p_4 \overline{t_2} = 0, t_2 \leq t_2, t_2 \overline{t_3} \leq t_5\}$$

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

$$r_1 = 0 \;\wedge \cdots \wedge\; r_n = 0 \quad \rightarrow \quad e_1 = e_2$$

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

$$t_0 p_1 t_1 p_2 t_2 (t_3 t_2 p_3 t_4 p_4)^\star \overline{t_3 t_5} + uru = 0 + uru,$$

where $u = (p_1 + p_2 + p_3 + p_4)^\star$ and $r = t_0 p_1 \overline{t_1} + t_1 p_2 \overline{t_2} + t_3 t_2 p_3 \overline{t_4} + t_4 p_2 \overline{t_2} + t_2 \overline{t_3 t_5}$.

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

$$\mathsf{At} = \{x_1 \cdots x_l \mid x_i \in \{t_i, \bar{t}_i\},\ t_i \in \mathsf{T}\}$$

set of all truth assignments to T

$$\mathsf{GS} = (\mathsf{At} \cdot \mathsf{P})^\star \cdot \mathsf{At}$$

set of guarded strings over P and T

$$\alpha_1 p_1 \alpha_2 p_2 \cdots p_{n-1} \alpha_n \in \mathtt{GS}\ .$$

$$X \diamond Y = \{\ x\alpha y \mid x\alpha \in X, \alpha y \in Y\ \}$$

$$X^0 = \mathtt{At}$$

$$X^{n+1} = X \diamond X^n$$

The language theoretic model of KAT (cont.)

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

$$
\begin{array}{lcl}
\mathsf{GS}(p) & = & \{\, \alpha p \beta \mid \alpha, \beta \in \mathsf{At} \,\} \\
\mathsf{GS}(b) & = & \{\, \alpha \mid \alpha \in \mathsf{At} \wedge \alpha \leq b\} \\
\mathsf{GS}(e_1 + e_2) & = & \mathsf{GS}(e_1) \cup \mathsf{GS}(e_2) \\
\mathsf{GS}(e_1 \cdot e_2) & = & \mathsf{GS}(e_1) \diamond \mathsf{GS}(e_2) \\
\mathsf{GS}(e_1^\star) & = & \cup_{n \geq 0} \mathsf{GS}(e_1)^n,
\end{array}
$$

where $\alpha \leq b$ if $\alpha \to b$ is a propositional tautology.

$e_1 = e_2$ iff $\mathsf{GS}(e_1) = \mathsf{GS}(e_2)$

Example:

$$\text{Consider } e = t_1 p(pq^\star t_2 + t_3 q)^\star$$

where $\mathsf{P} = \{p, q\}$ and $\mathsf{T} = \{t_1, t_2, t_3\}$,

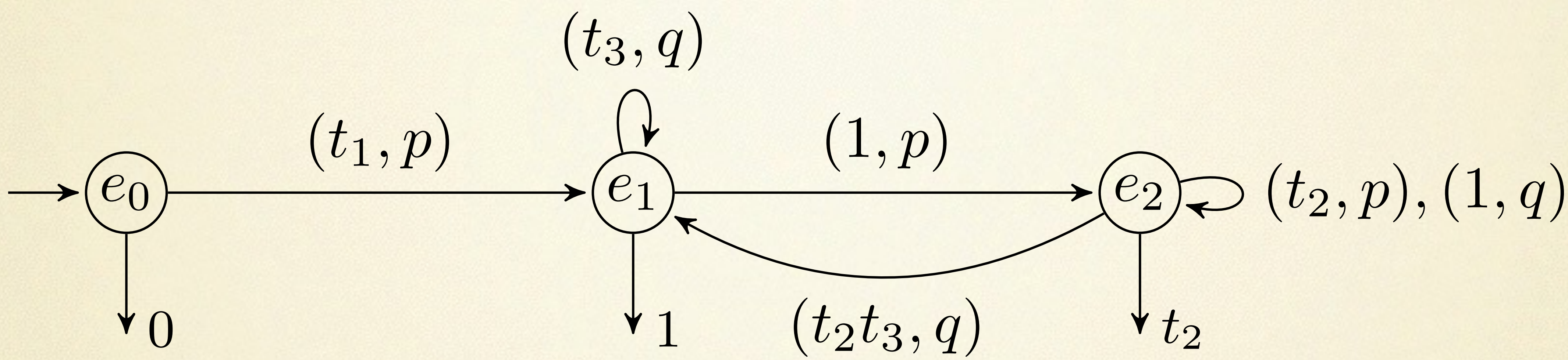
and

$$\mathsf{At} = \{\overline{t_1 t_2 t_3}, \overline{t_1 t_2} t_3, \overline{t_1} t_2 \overline{t_3}, \overline{t_1} t_2 t_3, t_1 \overline{t_2 t_3}, t_1 \overline{t_2} t_3, t_1 t_2 \overline{t_3}, t_1 t_2 t_3\}$$

We have for instance,

$$t_1 \overline{t_2} t_3 \; p \; t_1 t_2 t_3 \; q \; t_1 \overline{t_2 t_3} \in \mathsf{GS}(e)$$

Automata for guarded strings



$$\mathcal{A} = \langle S, s_0, o, \delta \rangle$$

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

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

$$t_1 \overline{t_2} t_3\ p\ t_1 t_2 t_3\ q\ t_1 \overline{t_2 t_3} \in \mathsf{GS}(\mathcal{A})$$

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 $\mathsf{A_B}$ be a set of basic actions, then the set of SKA expressions contains 0 plus all terms generated by the following grammar

$$\alpha \to 1 \mid a \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha \times \alpha \mid \alpha^* \quad (a \in \mathsf{SKA})$$

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

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

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

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

Example: Let $\mathsf{A_B} = \{a, b\}$, 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!