KAT and Hoare Logic with Derivatives

Ricardo Almeida Sabine Broda Nelma Moreira
Centro de Matemática da Universidade do Porto
Departamento de Ciência de Computadores,
Faculdade de Ciências, Universidade do Porto

Technical Report Series: DCC-2013-04
Version 1.0 February 2013

Departamento de Ciência de Computadores
&
Centro de Matemtica da Universidade do Porto Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL
Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

‘

KAT and Hoare Logic with Derivatives*

Ricardo Almeida Sabine Broda Nelma Moreira
Centro de Matemática da Universidade do Porto
Departamento de Ciência de Computadores,
Faculdade de Ciências, Universidade do Porto

May 20, 2013

Abstract

Kleene algebra with tests (KAT) is an equational system for program verification, which is the combination of Boolean algebra (BA) and Kleene algebra (KA), the algebra of regular expressions. In particular, KAT subsumes the propositional fragment of Hoare logic (PHL) which is a formal system for the specification and verification of programs, and that is currently the base of most tools for checking program correctness. Both the equational theory of KAT and the encoding of PHL in KAT are known to be decidable. In this paper we present a new decision procedure for the equivalence of two KAT expressions based on the notion of partial derivatives. We also introduce the notion of derivative modulo particular sets of equations. With this we extend the previous procedure for deciding PHL. Some experimental results are also presented.

1 Introduction

Kleene algebra with tests (KAT) is an equational algebraic system for reasoning about programs that combines Kleene algebra (KA) with Boolean algebra [18]. In particular, KAT subsumes PHL [15], the propositional fragment of Hoare logic, which is a formal system for the specification and verification of programs, and that is currently the base of most tools for checking program correctness [11]. Testing if two KAT expressions are equivalent is tantamount to prove that two programs are equivalent or that a Hoare triple is valid. Deciding the equivalence of KAT expressions is as hard as deciding regular expressions (KA expression) equivalence, i.e. PSPACE-complete [8]. In spite of KAT's success in dealing with several software verification tasks, there are very few software applications that implement KAT's equational theory and/or provide adequate decision procedures. Most of them are within (interactive) theorem provers or part of model checking systems, see [1, 12, 6] for some examples.

Based on a rewrite system of Antimirov and Mosses [5], Almeida *et al.* [3] developed an algorithm that decides regular expression equivalence through an iterated process of testing the equivalence of their derivatives, without resorting to the classic method of minimal automaton comparison. Statistically significant experimental tests showed that this method

*This work was partially funded by the ERDF - European Regional Development Fund through the programme COMPETE (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia under the project PEst-C/MAT/UI0144/2011, CANTE-PTDC/EIA-CCO/101904/2008, and FCOMP-01-0124-FEDER-020486

is, on average and using an uniform distribution, more efficient than the classical methods based on automata [2]. Another advantage of this method is that it is easily adapted to other Kleene algebra, such as KAT. In this paper we present an extension of that decision algorithm to test equivalence in KAT. The termination and correctness of the algorithm follow the lines of [3], but are also close to the coalgebraic approach to KAT presented by Kozen [17]. Deciding PHL can be reduced to testing KAT expressions equivalence [15]. Here we present an alternative method by extending the notion of derivative modulo a set of (atomic equational) assumptions. Once again the decision procedure has to be only slightly adapted. The new method reduces the size of the KAT expressions to be compared with the cost of a preprocessing phase. All the procedures were implemented in OCaml and some experimental results are also presented.

2 Preliminaries

We briefly review some basic definitions about regular expressions, Kleene algebras, Kleene algebras with tests (KAT), and KAT expressions. For more details, we refer the reader to [13, 14, 18, 16, 8].

2.1 Kleene Algebra and Regular Expressions

Let $\Sigma = \{p_1, \ldots, p_k\}$, with $k \geq 1$, be an *alphabet*. A *word* w over Σ is any finite sequence of letters. The *empty word* is denoted by 1. Let Σ^* be the set of all words over Σ. A *language* over Σ is a subset of Σ^*. The *left quotient* of a language $L \subseteq \Sigma^*$ by a word $w \in \Sigma^*$ is the language $w^{-1}L = \{x \in \Sigma^* \mid wx \in L\}$. The set of *regular expressions* over Σ, R_Σ, is defined by:

$$r \quad := \quad 0 \mid 1 \mid p \in \Sigma \mid (r_1 + r_2) \mid (r_1 \cdot r_2) \mid r^* \tag{1}$$

where the operator $\cdot$ (concatenation) is often omitted. The language $\mathcal{L}(r)$ associated to r is inductively defined as follows: $\mathcal{L}(0) = \emptyset$, $\mathcal{L}(1) = \{1\}$, $\mathcal{L}(p) = \{p\}$ for $p \in \Sigma$, $\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$, $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2)$, and $\mathcal{L}(r^*) = \mathcal{L}(r)^*$. Two regular expressions r_1 and r_2 are *equivalent* if $\mathcal{L}(r_1) = \mathcal{L}(r_2)$, and we write $r_1 = r_2$. With this interpretation, the algebraic structure $(R_\Sigma, +, \cdot, 0, 1)$ constitutes an idempotent semiring, and with the unary operator *, a Kleene algebra.

A Kleene algebra is an algebraic structure $\mathcal{K} = (K, +, \cdot, ^*, 0, 1)$, satisfying the axioms

below.

$$r_1 + (r_2 + r_3) = (r_1 + r_2) + r_3 \tag{2}$$
$$r_1 + r_2 = r_2 + r_1 \tag{3}$$
$$r + 0 = r + r \ = r \tag{4}$$
$$r_1(r_2 r_3) = (r_1 r_2) r_3 \tag{5}$$
$$1r = r1 \ = r \tag{6}$$
$$r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3 \tag{7}$$
$$(r_1 + r_2) r_3 = r_1 r_3 + r_2 r_3 \tag{8}$$
$$0r = r0 \ = 0 \tag{9}$$
$$1 + r r^* \leq r^* \tag{10}$$
$$1 + r^* r \leq r^* \tag{11}$$
$$r_1 + r_2 r_3 \leq r_3 \rightarrow {r_2}^* r_1 \leq r_3 \tag{12}$$
$$r_1 + r_2 r_3 \leq r_2 \rightarrow r_1 {r_3}^* \leq r_2 \tag{13}$$

In the above, $\leq$ is defined by $r_1 \leq r_2$ if and only if $r_1 + r_2 = r_2$. The axioms say that the structure is an *idempotent semiring* under $+$, $\cdot$, 0 and 1 and that * behaves like the Kleene star operator of formal language theory. This axiom set (with an usual first-order deduction system) constitutes a complete proof system for equivalence between regular expressions [13].

2.2 Kleene Algebra with Tests and KAT Expressions

A Kleene algebra with tests (KAT) is a Kleene algebra with an embedded Boolean subalgebra $\mathcal{K} = (K, B, +, \cdot, ^*, 0, 1, \bar{})$ where $^-$ is an unary operator denoting negation and is defined only on B, such that

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra;
- $(B, +, \cdot, \bar{}, 0, 1)$ is a Boolean algebra;
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, 0, 1)$.

Thus, a KAT is an algebraic structure that satisfies the KA axioms (2)–(13) and the axioms for a Boolean algebra B.

Let $\Sigma = \{p_1, \ldots, p_k\}$ be a non-empty set of (primitive) *action* symbols and $T = \{t_1, \ldots, t_l\}$ be a non-empty set of (primitive) *test* symbols. The set of boolean expressions over T is denoted by Bexp and the set of KAT expressions by Exp, with elements b_1, $b_2, \ldots$ and e_1, $e_2, \ldots$, respectively. The abstract syntax of KAT expressions over an alphabet $\Sigma \cup T$ is given by the following grammar,

$$\begin{aligned} b \in \mathsf{Bexp} &:= 0 \mid 1 \mid t \in T \mid \overline{b} \mid b_1 + b_2 \mid b_1 \cdot b_2 \\ e \in \mathsf{Exp} &:= p \in \Sigma \mid b \in \mathsf{Bexp} \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid {e_1}^*. \end{aligned}$$

As usual, we often omit the operator $\cdot$ in concatenations and in conjunctions. The standard language-theoretic models of KAT are regular sets of *guarded strings* over alphabets Σ and T [16]. Let $\overline{T} = \{\overline{t} \mid t \in T\}$ and let At be the set of *atoms*, i.e., of all truth assignments to T,

$$\mathsf{At} = \{b_1 \ldots b_l \mid b_i \text{ is either } t_i \text{ or } \overline{t_i} \text{ for } 1 \leq i \leq l \text{ and } t_i \in T\}.$$

Then the set of *guarded strings* over Σ and T is $\mathsf{GS} = (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At}$. Guarded strings will be denoted by $x, y, \ldots$. For $x = \alpha_1 p_1 \alpha_2 p_2 \cdots p_{n-1} \alpha_n \in \mathsf{GS}$, where $n \geq 1$, $\alpha_i \in \mathsf{At}$ and $p_i \in \Sigma$, we define $\mathsf{first}(x) = \alpha_1$ and $\mathsf{last}(x) = \alpha_n$. If $\mathsf{last}(x) = \mathsf{first}(y)$, then the *fusion product* xy is defined by concatenating x and y, omitting the extra occurrence of the common atom. If $\mathsf{last}(x) \neq \mathsf{first}(y)$, then xy does not exist. For sets $X, Y \subseteq \mathsf{GS}$ of guarded strings, the set $X \diamond Y$ defines the set of all xy such that $x \in X$ and $y \in Y$. We have that $X^0 = \mathsf{At}$ and $X^{n+1} = X \diamond X^n$, for $n \geq 0$.

Every KAT expression $e \in \mathsf{Exp}$ denotes a set of *guarded strings*, $\mathsf{GS}(e) \subseteq \mathsf{GS}$. Given a KAT expression e we define $\mathsf{GS}(e)$ inductively as follows,

$$\begin{array}{lcll} \mathsf{GS}(p) & = & \{\, \alpha_1 p \alpha_2 \mid \alpha_1, \alpha_2 \in \mathsf{At} \,\} & p \in \Sigma \\ \mathsf{GS}(b) & = & \{\, \alpha \in \mathsf{At} \mid \alpha \leq b \,\} & b \in \mathsf{Bexp} \\ \mathsf{GS}(e_1 + e_2) & = & \mathsf{GS}(e_1) \cup \mathsf{GS}(e_2) & \\ \mathsf{GS}(e_1 e_2) & = & \mathsf{GS}(e_1) \diamond \mathsf{GS}(e_2) & \\ \mathsf{GS}(e^*) & = & \cup_{n \geq 0} \mathsf{GS}(e)^n. & \end{array}$$

We say that two KAT expressions e_1 and e_2 are *equivalent*, and write $e_1 = e_2$, if and only if $\mathsf{GS}(e_1) = \mathsf{GS}(e_2)$. Kozen [18] showed that one has $e_1 = e_2$ modulo the KAT axioms, if and only if, $e_1 = e_2$ is true in the free Kleene algebra with tests on generators $\Sigma \cup T$. Two sets of KAT expressions $E, F \subseteq \mathsf{Exp}$ are *equivalent* if and only if $\mathsf{GS}(E) = \mathsf{GS}(F)$, where $\mathsf{GS}(E) = \cup_{e \in E} \mathsf{GS}(e)$.

3 Deciding Equivalence in KAT

In this section we present a decision algorithm to test equivalence in KAT. Kozen [17] presented a coalgebraic theory for KAT extending Rutten's coalgebraic approach for KA [20], and improving the framework of Chen and Pucella [7]. Extending the notion of Brzozowski derivatives to KAT, Kozen proved the existence of a coinductive equivalence procedure. Our approach follows closely that work, but we explicitly define the notion of partial derivatives for KAT, and we effectively provide a (inductive) decision procedure. This decision procedure is an extension of the algorithm for deciding equivalence of regular expressions given in [3, 5], that does not use the axiomatic system. Equivalence of expressions is decided through an iterated process of testing the equivalence of their partial derivatives.

3.1 Derivatives

Given a set of guarded strings R, its derivative with respect to $\alpha p \in \mathsf{At} \cdot \Sigma$, denoted by $\mathsf{D}_{\alpha p}(R)$, is defined as being the *left quotient* of R by αp. As such, one considers the following *derivative* functions,

$$\mathsf{D} : \mathsf{At} \cdot \Sigma \to \mathcal{P}(\mathsf{GS}) \to \mathcal{P}(\mathsf{GS}) \qquad \mathsf{E} : \mathsf{At} \to \mathcal{P}(\mathsf{GS}) \to \{0, 1\}$$

consisting of components,

$$\mathsf{D}_{\alpha p} : \mathcal{P}(\mathsf{GS}) \to \mathcal{P}(\mathsf{GS}) \qquad \mathsf{E}_{\alpha} : \mathcal{P}(\mathsf{GS}) \to \{0, 1\}$$

defined as follows. For $\alpha \in \mathsf{At}$, $p \in \Sigma$ and $R \subseteq \mathsf{GS}$,

$$\mathsf{D}_{\alpha p}(R) = \{\, y \in \mathsf{GS} \mid \alpha p y \in R \,\} \qquad \text{and} \qquad \mathsf{E}_{\alpha}(R) = \begin{cases} 1 & \text{if } \alpha \in R \\ 0 & \text{otherwise.} \end{cases}$$

3.2 Partial Derivatives

The notion of set of *partial derivatives*, cf. [4, 19], corresponds to a finite set representation of the derivatives of an expression. Given $\alpha \in \mathsf{At}$, $p \in \Sigma$ and $e \in \mathsf{Exp}$, the set $\Delta_{\alpha p}(e)$ of partial derivatives of e with respect to αp is inductively defined as follows,

$$\begin{aligned}
\Delta : \mathsf{At} \cdot \Sigma \;&\to\; \mathsf{Exp} \;\to\; \mathcal{P}(\mathsf{Exp}) \\
\Delta_{\alpha p}(p') &= \begin{cases} \{1\} & \text{if } p = p' \\ \emptyset & \text{otherwise} \end{cases} \\
\Delta_{\alpha p}(b) &= \emptyset \\
\Delta_{\alpha p}(e_1 + e_2) &= \Delta_{\alpha p}(e_1) \cup \Delta_{\alpha p}(e_2) \\
\Delta_{\alpha p}(e_1 e_2) &= \begin{cases} \Delta_{\alpha p}(e_1) \cdot e_2 & \text{if } \mathsf{E}_\alpha(e_1) = 0 \\ \Delta_{\alpha p}(e_1) \cdot e_2 \cup \Delta_{\alpha p}(e_2) & \text{if } \mathsf{E}_\alpha(e_1) = 1 \end{cases} \\
\Delta_{\alpha p}(e^*) &= \Delta_{\alpha p}(e) \cdot e^*,
\end{aligned}$$

where for $\Gamma \subseteq \mathsf{Exp}$ and $e \in \mathsf{Exp}$, $\Gamma \cdot e = \{\, e'e \mid e' \in \Gamma \,\}$ if $e \neq 0$ and $e \neq 1$, and $\Gamma \cdot 0 = \emptyset$ and $\Gamma \cdot 1 = \Gamma$, otherwise. We note that $\Delta_{\alpha p}(e)$ corresponds to an equivalence class of $D_{\alpha p}(e)$ (the syntactic Brzozowski derivative, defined in [17]) modulo axioms (2)–(4), (8), and (9). Kozen calls such a structure a *right presemiring*.

The following syntactic definition of $\mathsf{E}_\alpha : \mathsf{At} \to \mathsf{Exp} \to \{0,1\}$ is from [17] and simply evaluates an expression with respect to the truth assignment α.

$$\begin{aligned}
\mathsf{E}_\alpha(p) &= 0 & \qquad \mathsf{E}_\alpha(e_1 + e_2) &= \mathsf{E}_\alpha(e_1) + \mathsf{E}_\alpha(e_2) \\
\mathsf{E}_\alpha(b) &= \begin{cases} 1 & \text{if } \alpha \leq b \\ 0 & \text{otherwise} \end{cases} & \qquad \mathsf{E}_\alpha(e_1 e_2) &= \mathsf{E}_\alpha(e_1)\mathsf{E}_\alpha(e_2) \\
& & \qquad \mathsf{E}_\alpha(e^*) &= 1.
\end{aligned}$$

One can show that,

$$\mathsf{E}_\alpha(e) = \begin{cases} 1 & \text{if } \alpha \leq e \\ 0 & \text{if } \alpha \not\leq e \end{cases} = \begin{cases} 1 & \text{if } \alpha \in \mathsf{GS}(e) \\ 0 & \text{if } \alpha \notin \mathsf{GS}(e). \end{cases}$$

The next proposition shows that for all KAT expressions e the set of guarded strings correspondent to the set of partial derivatives of e w.r.t. $\alpha p \in \mathsf{At} \cdot \Sigma$ is the derivative of $\mathsf{GS}(e)$ by αp.

Proposition 1. *For all KAT expressions e, all atoms α and all symbols p,*

$$\mathsf{D}_{\alpha p}(\mathsf{GS}(e)) = \mathsf{GS}(\Delta_{\alpha p}(e)).$$

Proof. The proof is obtained by induction on the structure of e. We exemplify with the case $e = e_1 e_2$, where

$$\begin{aligned}
\mathsf{D}_{\alpha p}(\mathsf{GS}(e)) &= \mathsf{D}_{\alpha p}(\mathsf{GS}(e_1) \diamond \mathsf{GS}(e_2)) \\
&= \begin{cases} \mathsf{D}_{\alpha p}(\mathsf{GS}(e_1)) \diamond \mathsf{GS}(e_2) & \text{if } \alpha \notin \mathsf{GS}(e_1) \\ \mathsf{D}_{\alpha p}(\mathsf{GS}(e_1)) \diamond \mathsf{GS}(e_2) \cup \mathsf{D}_{\alpha p}(\mathsf{GS}(e_2)) & \text{if } \alpha \in \mathsf{GS}(e_1) \end{cases} \\
&\quad \text{applying the induction hypothesis} \\
&= \begin{cases} (\cup_{e' \in \Delta_{\alpha p}(e_1)} \mathsf{GS}(e')) \diamond \mathsf{GS}(e_2) & \text{if } \mathsf{E}_\alpha(e_1) = 0 \\ (\cup_{e' \in \Delta_{\alpha p}(e_1)} \mathsf{GS}(e')) \diamond \mathsf{GS}(e_2) \cup \mathsf{GS}(\Delta_{\alpha p}(e_2)) & \text{if } \mathsf{E}_\alpha(e_1) = 1 \end{cases} \\
&= \begin{cases} \cup_{e' \in \Delta_{\alpha p}(e_1)} \mathsf{GS}(e'e_2) & \text{if } \mathsf{E}_\alpha(e_1) = 0 \\ (\cup_{e' \in \Delta_{\alpha p}(e_1)} \mathsf{GS}(e'e_2)) \cup \mathsf{GS}(\Delta_{\alpha p}(e_2)) & \text{if } \mathsf{E}_\alpha(e_1) = 1 \end{cases} \\
&= \begin{cases} \mathsf{GS}(\Delta_{\alpha p}(e_1) \cdot e_2) & \text{if } \mathsf{E}_\alpha(e_1) = 0 \\ \mathsf{GS}(\Delta_{\alpha p}(e_1) \cdot e_2) \cup \mathsf{GS}(\Delta_{\alpha p}(e_2)) & \text{if } \mathsf{E}_\alpha(e_1) = 1 \end{cases} \\
&= \mathsf{GS}(\Delta_{\alpha p}(e_1 e_2)) = \mathsf{GS}(\Delta_{\alpha p}(e)).
\end{aligned}$$

□

The notion of partial derivative of an expression w.r.t. $\alpha p \in \mathsf{At} \cdot \Sigma$ can be extended to words $x \in (\mathsf{At} \cdot \Sigma)^*$, as follows,

$$\begin{aligned} \hat{\Delta} : (\mathsf{At} \cdot \Sigma)^* &\rightarrow \mathsf{Exp} \rightarrow \mathcal{P}(\mathsf{Exp}) \\ \hat{\Delta}_1(e) &= \{e\} \\ \hat{\Delta}_{w\alpha p}(e) &= \Delta_{\alpha p}(\hat{\Delta}_w(e)). \end{aligned}$$

Here, the notion of (partial) derivatives has been extended to sets of KAT expressions $E \subseteq \mathsf{Exp}$, by defining, as expected, $\Delta_{\alpha p}(E) = \cup_{e \in E} \Delta_{\alpha p}(e)$, for $\alpha p \in \mathsf{At} \cdot \Sigma$. Analogously, we also consider $\hat{\Delta}_x(E)$ and $\hat{\Delta}_R(E)$, for $x \in (\mathsf{At} \cdot \Sigma)^*$ and $R \subseteq (\mathsf{At} \cdot \Sigma)^*$.

The fact, that for any $e \in \mathsf{Exp}$ the set $\hat{\Delta}_{(\mathsf{At} \cdot \Sigma)^*}(e)$ is finite, ensures the termination of the decision procedure presented in the next section.

3.3 A Decision Procedure for KAT Expressions Equivalence

In this section we describe an algorithm for testing the equivalence of a pair of KAT expressions using partial derivatives. Following Antimirov [4], and for the sake of efficiency, we define the function f that given an expression e computes the set of pairs $(\alpha p, e')$, such that for each $\alpha p \in \mathsf{At} \cdot \Sigma$, the corresponding e' is a partial derivative of e with respect to αp.

$$\begin{aligned} \mathsf{f} : \mathsf{Exp} &\rightarrow \mathcal{P}(\mathsf{At} \cdot \Sigma \times \mathsf{Exp}) \\ \mathsf{f}(p) &= \{ (\alpha p, 1) \mid \alpha \in \mathsf{At} \} \\ \mathsf{f}(b) &= \emptyset \\ \mathsf{f}(e_1 + e_2) &= \mathsf{f}(e_1) \cup \mathsf{f}(e_2) \\ \mathsf{f}(e_1 e_2) &= \mathsf{f}(e_1) \cdot e_2 \cup \{ (\alpha p, e) \in \mathsf{f}(e_2) \mid \mathsf{E}_\alpha(e_1) = 1 \} \\ \mathsf{f}(e^*) &= \mathsf{f}(e) \cdot e^* \end{aligned}$$

where, as before, $\Gamma \cdot e = \{ (\alpha p, e'e) \mid (\alpha p, e') \in \Gamma \}$ if $e \neq 0$ and $e \neq 1$, and $\Gamma \cdot 0 = \emptyset$ and $\Gamma \cdot 1 = \Gamma$, otherwise. Also, we denote by $\mathsf{hd}(\mathsf{f}(e)) = \{\alpha p \mid (\alpha p, e') \in \mathsf{f}(e)\}$ the set of *heads* (i.e. first components of each element) of $\mathsf{f}(e)$. The function $\mathsf{der}_{\alpha p}$, defined in (14), collects all the partial derivatives of an expression e w.r.t. αp, that were computed by function f.

$$\mathsf{der}_{\alpha p}(e) = \{e' \mid (\alpha p, e') \in \mathsf{f}(e)\} \tag{14}$$

The proof of the following Proposition is almost trivial and follows from the symmetry of the definitions of $\mathsf{der}_{\alpha p}$, f, and $\Delta_{\alpha p}$.

Proposition 2. *For all $e, e' \in \mathsf{Exp}$, $\alpha \in \mathsf{At}$ and $p \in \Sigma$ one has, $\mathsf{der}_{\alpha p}(e) = \Delta_{\alpha p}(e)$.*

To define the decision procedure we need to consider the above functions and the ones defined in Section 3.2 applied to sets of KAT expressions. Then, we define the function $\mathsf{derivatives}$ that given two sets of KAT expressions E_1 and E_2 computes all pairs of sets of partial derivatives of E_1 and E_2 w.r.t. $\alpha p \in \mathsf{At} \cdot \Sigma$, respectively.

$$\begin{aligned} &\mathsf{derivatives} : \mathcal{P}(\mathsf{Exp})^2 \rightarrow \mathcal{P}(\mathcal{P}(\mathsf{Exp})^2) \\ &\mathsf{derivatives}(E_1, E_2) = \{(\mathsf{der}_{\alpha p}(E_1), \mathsf{der}_{\alpha p}(E_2)) \mid \alpha p \in \mathsf{hd}(E_1 \cup E_2)\} \end{aligned}$$

Finally, we present the function equiv that tests if two (sets of) KAT expressions are equivalent. For two sets of KAT expressions E_1 and E_2 the function returns True, if for every atom α, $\mathsf{E}_\alpha(E_1) = \mathsf{E}_\alpha(E_2)$ and if, for every αp, the partial derivative of E_1 w.r.t. αp is equivalent to the partial derivative of E_2 w.r.t. αp.

$$\begin{aligned}\mathsf{equiv} :\ & \mathcal{P}(\mathcal{P}(\mathsf{Exp})^2) \times \mathcal{P}(\mathcal{P}(\mathsf{Exp})^2) \to \{\mathsf{True}, \mathsf{False}\} \\ & \mathsf{equiv}(\emptyset, H) = \mathsf{True} \\ & \mathsf{equiv}(\{(E_1, E_2)\} \cup S, H) = \begin{cases} \mathsf{False} & \text{if } \exists \alpha \in \mathsf{At} : \mathsf{E}_\alpha(E_1) \neq \mathsf{E}_\alpha(E_2) \\ \mathsf{equiv}(S \cup S', H') & \text{otherwise,} \end{cases}\end{aligned}$$

where

$$S' = \{d \mid d \in \mathsf{derivatives}(E_1, E_2) \text{ and } d \notin H'\} \quad \text{and} \quad H' = \{(E_1, E_2)\} \cup H.$$

The function equiv accepts two sets S and H as arguments. At each step, S contains the pairs of expressions that still need to be checked for equivalence, whereas H contains the pairs of expressions that have already been tested. The use of the set H is important to ensure that the derivatives of the same pair of expressions are not computed more than once, and thus prevent a possible infinite loop.

To compare two expressions e_1 and e_2, the initial call must be $\mathsf{equiv}(\{(\{e_1\}, \{e_2\})\}, \emptyset)$. At each step the function takes a pair (E_1, E_2) and verifies if there exists an atom α such that $\mathsf{E}_\alpha(E_1) \neq \mathsf{E}_\alpha(E_2)$. If such an atom exists, then $E_1 \neq E_2$ and the function halts, returning False. If no such atom exists, then the function adds (E_1, E_2) to H and then replaces in S the pair (E_1, E_2) by the pairs of its corresponding derivatives provided that these are not in H already. The return value of equiv will be the result of recursively calling equiv with the new sets as arguments. If the function ever receives $\emptyset$ as S, then the initial call ensures that $e_1 = e_2$, since all derivatives have been successfully tested, and the function returns True.

3.4 Termination and Correctness

First, we show that the function equiv terminates. For every KAT expression e, we define the set $\mathsf{PD}(e)$ and show that, for every KAT expression e, the set of partial derivatives of e is a subset of $\mathsf{PD}(e)$, which on the other hand is clearly finite. The set $\mathsf{PD}(e)$ coincides with the *closure* of a KAT expression e, defined by Kozen, and is also similar to Mirkin's prebases [19].

$$\begin{array}{ll} \mathsf{PD}(b) & = \{b\} \\ \mathsf{PD}(p) & = \{p, 1\} \end{array} \qquad \begin{array}{ll} \mathsf{PD}(e_1 + e_2) & = \{e_1 + e_2\} \cup \mathsf{PD}(e_1) \cup \mathsf{PD}(e_2) \\ \mathsf{PD}(e_1 e_2) & = \{e_1 e_2\} \cup \mathsf{PD}(e_1) \cdot e_2 \cup \mathsf{PD}(e_2) \\ \mathsf{PD}(e^*) & = \{e^*\} \cup \mathsf{PD}(e) \cdot e^*. \end{array}$$

Lemma 1. *Consider $e, e' \in \mathsf{Exp}$, $\alpha \in \mathsf{At}$ and $p \in \Sigma$. If $e' \in \mathsf{PD}(e)$, then $\Delta_{\alpha p}(e') \subseteq \mathsf{PD}(e)$.*

Proof. The proof is obtained by induction on the structure of e. We exemplify with the case $e = e_1 e_2$. Let $e' \in \mathsf{PD}(e_1 e_2) = \{e_1 e_2\} \cup \mathsf{PD}(e_1) \cdot e_2 \cup \mathsf{PD}(e_2)$.

- If $e' \in \{e_1 e_2\}$, then $\Delta_{\alpha p}(e') \subseteq \Delta_{\alpha p}(e_1) \cdot e_2 \cup \Delta_{\alpha p}(e_2)$. But $e_1 \in \mathsf{PD}(e_1)$ and $e_2 \in \mathsf{PD}(e_2)$, so applying the induction hypothesis twice, we obtain $\Delta_{\alpha p}(e') \subseteq \mathsf{PD}(e_1) \cdot e_2 \cup \mathsf{PD}(e_2) \subseteq \mathsf{PD}(e)$.
- If $e' \in \mathsf{PD}(e_1) \cdot e_2$, then $e' = e'_1 e_2$ such that $e'_1 \in \mathsf{PD}(e_1)$. So $\Delta_{\alpha p}(e') \subseteq \Delta_{\alpha p}(e'_1) \cdot e_2 \cup \Delta_{\alpha p}(e_2) \subseteq \mathsf{PD}(e_1) \cdot e_2 \cup \mathsf{PD}(e_2) \subseteq \mathsf{PD}(e)$.

- Finally, if $e' \in \mathsf{PD}(e_2)$, again by the induction hypothesis we have $\Delta_{\alpha p}(e') \subseteq \mathsf{PD}(e_2) \subseteq \mathsf{PD}(e)$.

□

Proposition 3. *For all $x \in (\mathsf{At} \cdot \Sigma)^*$, one has $\hat{\Delta}_x(e) \subseteq \mathsf{PD}(e)$.*

Proof. We prove this lemma by induction on the length of x. If $|x| = 0$, i.e. $x = 1$, then $\hat{\Delta}_1(e) = \{e\} \subseteq \mathsf{PD}(e)$. If $x = w\alpha p$, then $\hat{\Delta}_{w\alpha p} = \cup_{e' \in \hat{\Delta}_w(e)} \Delta_{\alpha p}(e')$. By induction hypothesis, we know that $\hat{\Delta}_w(e) \subseteq \mathsf{PD}(e)$. By Lemma 1, if $e' \in \mathsf{PD}(e)$, then $\Delta_{\alpha p}(e') \subseteq \mathsf{PD}(e)$. Consequently, $\cup_{e' \in \hat{\Delta}_w(e)} \Delta_{\alpha p}(e') \subseteq \mathsf{PD}(e)$. □

Corollary 1. *For all KAT expressions e, the set $\hat{\Delta}_{(\mathsf{At} \cdot \Sigma)^*}(e)$ is finite.*

It is obvious that the previous results also apply to sets of KAT expressions.

Proposition 4. *The function* equiv *is terminating.*

Proof. When the set S is empty it follows directly from the definition of the function that it terminates. We argue that when S is not empty the function also terminates based on these two aspects:

- In order to ensure that the set of partial derivatives of a pair of (sets of) expressions are not computed more than once, the set H is used to store the ones which have already been calculated.
- Each function call removes one pair (E_1, E_2) from the set S and appends the set of partial derivatives of (E_1, E_2), which have not been calculated yet, to S. By Corollary 1, the set of partial derivatives of an expression by any word is finite, and so eventually S becomes $\emptyset$.

Thus, since at each call the function analyzes one pair from S, after a finite number of calls the function terminates. □

The next proposition states the correctness of our algorithm. Coalgebraically it states that two KAT expressions are equivalent if and only if there exists a bisimulation between them [17, Thm. 5.2].

Proposition 5. *For all KAT expressions e_1 and e_2,*

$$\mathsf{GS}(e_1) = \mathsf{GS}(e_2) \quad \Leftrightarrow \quad \begin{cases} \mathsf{E}_\alpha(e_1) = \mathsf{E}_\alpha(e_2) & \text{and} \\ \mathsf{GS}(\Delta_{\alpha p}(e_1)) = \mathsf{GS}(\Delta_{\alpha p}(e_2)), & \forall \alpha \in \mathsf{At}. \end{cases}$$

Proof. Let us first prove the $\Leftarrow$ implication. If $\mathsf{GS}(e_1) \neq \mathsf{GS}(e_2)$, then there is $x \in \mathsf{GS}$, such that $x \in \mathsf{GS}(e_1)$ and $x \notin \mathsf{GS}(e_2)$ (or vice-versa). If $x = \alpha$, then we have $\mathsf{E}_\alpha(e_1) = 1 \neq 0 = \mathsf{E}_\alpha(e_2)$ and the test fails. If $x = \alpha p w$, such that $w \in (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At}$, then since $\alpha p w \in \mathsf{GS}(e_1)$ and $\alpha p w \notin \mathsf{GS}(e_2)$, we have that $w \in \mathsf{GS}(\Delta_{\alpha p}(e_1))$ and $w \notin \mathsf{GS}(\Delta_{\alpha p}(e_2))$. Thus, $\mathsf{GS}(\Delta_{\alpha p}(e_1)) \neq \mathsf{GS}(\Delta_{\alpha p}(e_2))$.

Let us now prove the $\Rightarrow$ implication. For $\alpha \in \mathsf{At}$, there is either $\alpha \in \mathsf{GS}(e_1)$ and $\alpha \in \mathsf{GS}(e_2)$, thus $\mathsf{E}_\alpha(e_1) = \mathsf{E}_\alpha(e_2) = 1$; or $\alpha \notin \mathsf{GS}(e_1)$ and $\alpha \notin \mathsf{GS}(e_2)$, thus $\mathsf{E}_\alpha(e_1) = \mathsf{E}_\alpha(e_2) = 0$. For $\alpha p \in \mathsf{At} \cdot \Sigma$, by Proposition 1, one has $\mathsf{GS}(\Delta_{\alpha p}(e_1)) = \mathsf{GS}(\Delta_{\alpha p}(e_2))$ if and only if $\mathsf{D}_{\alpha p}(\mathsf{GS}(e_1)) = \mathsf{D}_{\alpha p}(\mathsf{GS}(e_2))$. This follows trivially from $\mathsf{GS}(e_1) = \mathsf{GS}(e_2)$. □

4 Implementation

The algorithm presented in the previous section was implemented in *OCaml* [21]. Alternations, conjunctions, and disjunctions are represented by sets, and thus, commutativity and idempotence properties are naturally enforced. Concatenations are represented by lists of expressions. Primitive tests occurring in a KAT expression are represented by integers, and atoms by lists of boolean values (where primitive tests correspond to indexes). For each KAT expression e, we consider At as the set of atoms that correspond to the primitive tests that occur in e. The implementation of the functions defined in Section 3.2 and Section 3.3, do not differ much from their formal definitions. A common choice was the use of comprehension lists to define the inclusion criteria of elements in a set. Because of our basic representation of KAT expressions, we treat in a uniform way both expressions and sets of expressions. The function E_α, used in equiv, is implemented using a function called eAll, that takes as arguments two (sets of) expressions E_1 and E_2 and verifies if for every atom the truth assignments for E_1 and E_2 coincide.

4.1 Experimental Results

In order to test the performance of our decision procedure we ran some experiments. We used the FAdo system [9] to uniformly random generate samples of KAT expressions. Each sample has 10000 KAT expressions of a given length $|e|$ (number of symbols in the syntactic tree of $e \in \mathsf{Exp}$). The size of each sample is more than enough to ensure results statistically significant with 95% confidence level within a 5% error margin. The tests were executed in the same computer, an Intel® Xeon® 5140 at 2.33 GHz with 4 GB of RAM, running a minimal 64 bit Linux system. For each sample we performed two experiments: (1) we tested the equivalence of each KAT expression against itself; (2) we tested the equivalence of two consecutive KAT expressions. For each pair of KAT expressions we measured: the size of the set H produced by equiv (that measures the number of iterations) and the number of primitive tests in each expression ($|e|_T$). Table 1 summarizes some of the results obtained. Each row corresponds to a sample, where the three first columns characterize the sample, respectively, the number of primitive actions (k), the number of primitive tests (l), and the length of each KAT expression generated. Column four has the number of primitive tests in each expression ($|e|_T$). Columns five and six give the average size of H in the experiment (1) and (2), respectively. Column seven is the ratio of the equivalent pairs in experiment (2). Finally, columns eight and nine contain the average times, in seconds, of each comparison in the experiments (1) and (2). More than comparing with existent systems, which is difficult by the reasons pointed out in the introduction, these experiments aimed to test the feasibility of the procedure. As expected, the main *bottleneck* is the number of different primitive tests in the KAT expressions.

1	2	3	4	5	6	7	8	9
k	l	$\|e\|$	$\|e\|_T$	$H(1)$	$H(2)$	$=(2)$	Time(1)	Time(2)
5	5	50	9.98	7.35	0.53	0.42	0.0097	0.00087
5	5	100	19.71	15.74	0.76	0.48	0.0875	0.00223
10	10	50	11.12	8.30	0.50	0.07	0.5050	0.30963
10	10	100	21.93	16.78	0.67	0.18	20.45	1.31263
15	15	50	11.57	8.47	0.47	0.10	6.4578	55.22

Table 1: Experimental results for uniformly random generated KAT expressions.

5 Hoare Logic and KAT

Hoare logic was first introduced in 1969, cf. [11], and is a formal system widely used for the specification and verification of programs. Hoare logic uses *partial correctness assertions* (PCA's) to reason about program correctness. A PCA is a triple, $\{b\}P\{c\}$ with P being a program, and b and c logic formulas. We read such an assertion as *if b holds before the execution of P, then c will necessarily hold at the end of the execution, provided that P halts.* A deductive system of Hoare logic provides inference rules for deriving valid PCA's, where rules depend on the program constructs. We consider a simple **while** language, where a program P can be defined, as usual, by an assignment $x := v$; a **skip** command; a sequence $P; Q$, conditional **if** b **then** P **else** Q, and a loop **while** b **do** P.

There are several variations of Hoare logic and here we choose an inference system, considered in [10], that enjoys the *sub-formula* property, where the premises of a rule can be obtained from the assertions that occur in the rule's conclusion. With this property, given a PCA $\{b\}P\{c\}$, where P has also some annotated assertions, it is possible to automatically generate verification conditions that will ensure its validity. The inference rules for this system are the following:

$$\frac{b \to c}{\{b\}\ \mathbf{skip}\ \{c\}} \qquad \frac{b \to c[x/e]}{\{b\}\ x := e\ \{c\}} \qquad \frac{\{b\}\ P\ \{c\} \qquad \{c\}\ Q\ \{d\}}{\{b\}\ P\ ;\{c\}\ Q\ \{d\}}$$

$$\frac{\{b \wedge c\}\ P\ \{d\} \qquad \{\neg b \wedge c\}\ Q\ \{d\}}{\{c\}\ \mathbf{if}\ b\ \mathbf{then}\ P\ \mathbf{else}\ Q\ \{d\}} \qquad \frac{\{b \wedge i\}\ P\ \{i\} \qquad c \to i \qquad (i \wedge \neg b) \to d}{\{c\}\ \mathbf{while}\ b\ \mathbf{do}\ \{i\}P\ \{d\}}$$

5.1 Encoding Propositional Hoare Logic in KAT

The propositional fragment of Hoare logic (PHL), i.e., the fragment without the rule for assignment, can be encoded in KAT [15]. The encoding of an annotated **while** program P and of our inference system follow the same lines. In PHL, all assignment instructions are represented by primitive symbols p. The **skip** command is encoded by a distinguished primitive symbol $p_{\mathbf{skip}}$. If e_1, e_2 are respectively the encodings of programs P_1 and P_2, then the encoding of more complex constructs of an annotated **while** program involving P_1 and P_2 is as follows.

$$\begin{aligned} P_1\ ;\{c\}\ P_2 &\ \Rightarrow\ e_1 c e_2 \\ \mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 &\ \Rightarrow\ be_1 + \bar{b}e_2 \\ \mathbf{while}\ b\ \mathbf{do}\ \{i\}\ P_1 &\ \Rightarrow\ (bie_1)^*\bar{b} \end{aligned}$$

A PCA of the form $\{b\}P\{c\}$ is encoded in KAT as an equational identity of the form

$$be = bec \qquad \text{or equivalently by} \qquad be\overline{c} = 0,$$

where e is the encoding of the program P.

Now, suppose we want to prove the PCA $\{b\}P\{c\}$. Since the inference system for Hoare logic, that we are considering in this paper, enjoys the sub-formula property, one can generate mechanically in a backward fashion the verification conditions that ensure the PCA's validity.

Since in the KAT encoding, $be\overline{c} = 0$, we do not have the rule for assignment, besides verification conditions (proof obligations) of the form $b' \to c'$ we will also have assumptions of the form $b'p\overline{c'} = 0$.

One can generate a set of assumptions, $\Gamma = \mathsf{Gen}(be\overline{c})$, backwards from $be\overline{c} = 0$, where Gen is inductively defined by:

$$\begin{array}{lcll}
\mathsf{Gen}(b\ p_{\mathbf{skip}}\ \overline{c}) & = & \{b \leq c\} & \\
\mathsf{Gen}(b\ p\ \overline{c}) & = & \{b\ p\ \overline{c}\} & p_{\mathbf{skip}} \neq p \in \Sigma \\
\mathsf{Gen}(b\ e_1\ c\ e_2\ \overline{d}) & = & \mathsf{Gen}(b\ e_1\ \overline{c}) \cup \mathsf{Gen}(c\ e_2\ \overline{d}) & \\
\mathsf{Gen}(b\ (ce_1 + \overline{c}e_2)\ \overline{d}) & = & \mathsf{Gen}(bc\ e_1\ \overline{d}) \cup \mathsf{Gen}(b\overline{c}\ e_2\ \overline{d}) & \\
\mathsf{Gen}(b\ ((cie)^*\overline{c})\ \overline{d}) & = & \mathsf{Gen}(ic\ e\ \overline{i}) \cup \{b \leq i, i\overline{c} \leq d\} &
\end{array}$$

Note that Γ is necessarily of the form

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

where $p_1, \ldots, p_m \in \Sigma$ and such that all b's and c's are Bexp expressions. In Section 6, we show how one can prove the validity of $be_P\overline{c} = 0$ in the presence of such a set of assumptions Γ, but first we illustrate the encoding and generation of the assumption set with an example.

5.2 A Small Example

Consider the program P in Table 2, that calculates the factorial of a non-negative integer. We wish to prove that, at the end of the execution, the variable y contains the factorial of x, i.e. to verify the assertion $\{\mathsf{True}\}$ P $\{y = x!\}$.

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
	}	

Table 2: A program for the factorial

In order to apply the inference rules we need to annotate program P, obtaining program P'. Applying the inference rules for deriving PCA's in a backward fashion to $\{\mathsf{True}\}$ P$'$ $\{y = x!\}$, one easily generates the corresponding set of assumptions provided by the annotated version of the program. However, because we do not have the assignment rule in the KAT encoding, here we simulate that by considering not only verification conditions but also atomic PCA's $\{b'\}x := e\{c'\}$. Thus the assumption set is

$$\Gamma_P = \left\{ \begin{array}{l} \{\mathsf{True}\}y := 1\{y = 0!\}, \{\mathsf{y = 0!}\}z := 0\{y = z!\}, \\ \{\mathsf{y = z!} \wedge \neg \mathsf{z = x}\}z := z + 1\{y \times z = z!\}, \{y \times z = z!\}y := y \times z\{y = z!\}, \\ y = z! \rightarrow y = z!, (y = z! \wedge \neg\neg z = x) \rightarrow y = x! \end{array} \right\}.$$

On the other hand, using the correspondence of KAT primitive symbols and atomic parts of the annotated program P', as in Table 2, and additionally encoding True as t_0 and $y = x!$

as t_5, respectively, the encoding of $\{\mathsf{True}\}\ \mathrm{P}'\ \{y = x!\}$ in KAT is

$$t_0 p_1 t_1 p_2 t_2 (t_3 t_2 p_3 t_4 p_4)^* \overline{t_3} t_5 = 0. \tag{15}$$

The corresponding set of assumptions Γ in KAT is

$$\Gamma = \{t_0 p_1 \overline{t_1} = 0, t_1 p_2 \overline{t_2} = 0, t_2 t_3 p_3 \overline{t_4} = 0, t_4 p_4 \overline{t_2} = 0, t_2 \leq t_2, t_2 \overline{t_3} \leq t_5\}. \tag{16}$$

In the next section we will see how to prove in KAT an equation such as (15) in the presence of a set of assumptions such as (16).

6 Deciding Hoare Logic

Rephrasing the observation in the end of last section, we are interested in proving in KAT the validity of implications of the form

$$b_1 p_1 \overline{b'_1} = 0 \ \wedge \cdots \wedge \ b_m p_m \overline{b'_m} = 0 \ \wedge \ c_1 \leq c'_1 \ \wedge \cdots \wedge c_n \leq c'_n \quad \rightarrow \quad bp\overline{b'} = 0. \tag{17}$$

This can be reduced to proving the equivalence of KAT expressions, since it has been shown, cf. [15], 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

$$e_1 + uru = e_2 + uru \tag{18}$$

where $u = (p_1 + \cdots + p_k)^*$ and $r = r_1 + \ldots + r_n$. Testing this last equality can of course be done by applying our algorithm to $e_1 + uru$ and $e_2 + uru$. However, in the next subsection, we present an alternative method of proving the validity of implications of the form 17. This method has the advantage of prescinding from the expressions u and r, above.

6.1 Equivalence of KAT Expressions Modulo a Set of Assumptions

In the presence of a finite set of assumptions of the form

$$\Gamma = \{b_1 p_1 \overline{b'_1} = 0, \ldots, b_m p_m \overline{b'_m} = 0\} \cup \{c_1 \leq c'_1, \ldots, c_n \leq c'_n\} \tag{19}$$

we have to restrict ourselves to atoms that satisfy the restrictions in Γ. Thus, let

$$\mathsf{At}^\Gamma = \{\ \alpha \in \mathsf{At} \mid \alpha \leq c \rightarrow \alpha \leq c',\ \text{for all } c \leq c' \in \Gamma\ \}. \tag{20}$$

Given a KAT expression e, the *set of guarded strings modulo* Γ, $\mathsf{GS}^\Gamma(e)$, is inductively defined as follows.

$$\begin{array}{lcl}
\mathsf{GS}^\Gamma(p) & = & \{\ \alpha p \beta \mid \alpha, \beta \in \mathsf{At}^\Gamma \wedge \forall_{bp\overline{b'}=0 \,\in\, \Gamma}\ (\alpha \leq b \rightarrow \beta \leq b')\ \} \\
\mathsf{GS}^\Gamma(b) & = & \{\ \alpha \in \mathsf{At}^\Gamma \mid \alpha \leq b\ \} \\
\mathsf{GS}^\Gamma(e_1 + e_2) & = & \mathsf{GS}^\Gamma(e_1) \ \cup \ \mathsf{GS}^\Gamma(e_2) \\
\mathsf{GS}^\Gamma(e_1 e_2) & = & \mathsf{GS}^\Gamma(e_1) \diamond \mathsf{GS}^\Gamma(e_2) \\
\mathsf{GS}^\Gamma(e^*) & = & \cup_{n \geq 0} \mathsf{GS}^\Gamma(e)^n.
\end{array}$$

The following proposition characterizes the *equivalence* modulo a set of assumptions Γ, and ensures the correctness of the new Hoare logic decision procedure.

Proposition 6. *Let e_1 and e_2 be KAT expressions and Γ a set of assumptions as in (19). Then,*

$$\mathsf{KAT}, \Gamma \vdash e_1 = e_2 \qquad \textit{iff} \qquad \mathsf{GS}^\Gamma(e_1) = \mathsf{GS}^\Gamma(e_2).$$

Proof. By (18) one has $\mathsf{KAT}, \Gamma \vdash e_1 = e_2$ if and only if $e_1 + uru = e_2 + uru$ is provable in KAT, where $u = (p_1 + \cdots + p_k)^*$ and $r = b_1 p_1 \overline{b'_1} + \cdots + b_m p_m \overline{b'_m} + c_1 \overline{c'_1} + \cdots + c_n \overline{c'_n}$. The second equality is equivalent to $\mathsf{GS}(e_1 + uru) = \mathsf{GS}(e_2 + uru)$, i.e. $\mathsf{GS}(e_1) \cup \mathsf{GS}(uru) = \mathsf{GS}(e_2) \cup \mathsf{GS}(uru)$. In order to show the equivalence of this last equality and $\mathsf{GS}^\Gamma(e_1) = \mathsf{GS}^\Gamma(e_2)$, it is sufficient to show that for every KAT expression e one has $\mathsf{GS}^\Gamma(e) = \mathsf{GS}(e) \setminus \mathsf{GS}(uru)$ (note that $A \cup C = B \cup C \;\Leftrightarrow\; A \setminus C = B \setminus C$).

First we analyze under which conditions a guarded string x is an element of $\mathsf{GS}(uru)$. Given the values of u and r, it is easy to see that $x \in \mathsf{GS}(uru)$ if and only if in x occurs an atom α such that $\alpha \leq c$ and $\alpha \not\leq c'$ for some $c \leq c' \in \Gamma$, or x has a substring $\alpha p \beta$, such that $\alpha \leq b$ and $\alpha \not\leq b'$ for some $bp\overline{b'} \in \Gamma$. This means that $x \notin \mathsf{GS}(uru)$ if and only if every atom in x is an element of At^Γ and every substring $\alpha p \beta$ of x satisfies $(\alpha \leq b \to \beta \leq b')$, for all $bp\overline{b'} = 0 \in \Gamma$. From this remark and by the definitions of At^Γ and GS^Γ, we conclude that $\mathsf{GS}^\Gamma(e) \cap \mathsf{GS}(uru) = \emptyset$. Note also that, since $\mathsf{GS}^\Gamma(e)$ is a restriction of $\mathsf{GS}(e)$, one has $\mathsf{GS}^\Gamma(e) \subseteq \mathsf{GS}(e)$. Now it suffices to show that for every $x \in \mathsf{GS}(e) \setminus \mathsf{GS}(uru)$, one has $x \in \mathsf{GS}^\Gamma(e)$. This can be easily proved by induction on the structure of e. □

We now define the set of partial derivatives of a KAT expression modulo a set of assumptions Γ. Let $e \in \mathsf{Exp}$. If $\alpha \notin \mathsf{At}^\Gamma$, then $\Delta^\Gamma_{\alpha p}(e) = \emptyset$. For $\alpha \in \mathsf{At}^\Gamma$, let

$$\begin{aligned}
\Delta^\Gamma_{\alpha p}(p') &= \begin{cases} \{\Pi b' \mid bp\overline{b'} = 0 \in \Gamma \wedge \alpha \leq b\} & \text{if } p = p' \\ \emptyset & \text{if } p \neq p' \end{cases} \\
\Delta^\Gamma_{\alpha p}(b) &= \emptyset \\
\Delta^\Gamma_{\alpha p}(e_1 + e_2) &= \Delta^\Gamma_{\alpha p}(e_1) \cup \Delta^\Gamma_{\alpha p}(e_2) \\
\Delta^\Gamma_{\alpha p}(e_1 e_2) &= \begin{cases} \Delta^\Gamma_{\alpha p}(e_1) \cdot e_2 & \text{if } \mathsf{E}_\alpha(e_1) = 0 \\ \Delta^\Gamma_{\alpha p}(e_1) \cdot e_2 \cup \Delta^\Gamma_{\alpha p}(e_2) & \text{if } \mathsf{E}_\alpha(e_1) = 1 \end{cases} \\
\Delta^\Gamma_{\alpha p}(e^*) &= \Delta^\Gamma_{\alpha p}(e) \cdot e^*.
\end{aligned}$$

Note, that by definition, $\Pi\, b' = 1$ if there is no $bp = bpb' \in \Gamma$ such that $\alpha \leq b$ and $\alpha \in \mathsf{At}^\Gamma$. The next proposition states the correctness of the definition of $\Delta^\Gamma_{\alpha p}$.

Proposition 7. *Let Γ be a set of assumptions as above, $e \in \mathsf{Exp}$, $\alpha \in \mathsf{At}$, and $p \in \Sigma$. Then,*

$$\mathsf{D}_{\alpha p}(\mathsf{GS}^\Gamma(e)) = \mathsf{GS}^\Gamma(\Delta^\Gamma_{\alpha p}(e)).$$

Proof. The proof is obtained by induction on the structure of e. We only show the case $e = p$, since the other cases are similar to those in the proof of Proposition 1. If $\alpha \notin \mathsf{At}^\Gamma$, then $\mathsf{GS}^\Gamma(p) = \emptyset = \mathsf{D}_{\alpha p}(\mathsf{GS}^\Gamma(p))$. Also, $\Delta^\Gamma_{\alpha p}(p) = \emptyset = \mathsf{GS}^\Gamma(\Delta^\Gamma_{\alpha p}(p))$. Otherwise, if $\alpha \in \mathsf{At}^\Gamma$, then $\mathsf{GS}^\Gamma(p) = \{\alpha p \beta \mid \alpha, \beta \in \mathsf{At}^\Gamma \wedge \forall_{bp\overline{b'}=0 \in \Gamma}\ (\alpha \leq b \to \beta \leq b')\}$, thus $\mathsf{D}_{\alpha p}(\mathsf{GS}^\Gamma(p)) = \{\beta \in \mathsf{At}^\Gamma \mid \beta \leq b' \text{ for all } bp\overline{b'} = 0 \in \Gamma \text{ such that } \alpha \leq b\}$. On the other hand, $\Delta^\Gamma_{\alpha p}(p) = \{\Pi b' \mid bp\overline{b'} = 0 \in \Gamma \wedge \alpha \leq b\}$. Thus, $\mathsf{GS}^\Gamma(\Delta^\Gamma_{\alpha p}(p)) = \mathsf{GS}^\Gamma(c)$, where $c = \prod_{bp\overline{b'}=0 \in \Gamma, \alpha \leq b} b'$. We conclude that $\mathsf{GS}^\Gamma(c) = \{\beta \in \mathsf{At}^\Gamma \mid \beta \leq b' \text{ for all } bp\overline{b'} = 0 \in \Gamma \text{ such that } \alpha \leq b\}$. □

6.2 Testing Equivalence Modulo a Set of Assumptions

The decision procedure for testing equivalence presented before can be easily adapted. Given a set of assumptions Γ, the set At^Γ is obtained by filtering in At all atoms that satisfy c but do not satisfy c', for all $c \leq c' \in \Gamma$. The function f has to account for the new definition of $\Delta^\Gamma_{\alpha p}$.

We compared this new algorithm, equiv^Γ, with equiv when deciding the PCA presented in Subsection 5.2. First, we constructed expressions r and u from Γ, as described above and proved the equivalence of expressions $t_0p_1t_1p_2t_2(t_3t_2p_3t_4p_4)^*\overline{t_3t_5} + uru$ and $0 + uru$, with function equiv. In this case $|H| = 17$. In other words, equiv needed to derive 17 pairs of expressions in order to reach a conclusion about the correction of program P. Then, we applied function equiv^Γ directly to the pair $(t_0p_1t_1p_2t_2(t_3t_2p_3t_4p_4)^*\overline{t_3t_5}, 0)$ and Γ. In this case, $|H| = 5$. Other tests, that we ran, produced similar results, but at this point we have not carried out a study thorough enough to compare both methods.

7 Conclusion

Considering the algebraic properties of KAT expressions (or even KA expressions) it seems possible to improve the decision procedure for equivalence. The procedure essentially computes a bisimulation (or fails to do that if the expressions are inequivalent); thus it would be interesting to know if, for instance the maximum bisimulation can be obtained. Having a method that reduces the amount of used atoms, or alternatively to resort to an external SAT solver, would also turn the use of KAT expressions in formal verification more feasible. Concerning Hoare logic, it would be interesting to treat the assignment rule within a decidable first-order theory and to integrate the KAT decision procedure in an SMT solver.

References

[1] Kamal Aboul-Hosn & Dexter Kozen (2006): *KAT-ML: An Interactive Theorem Prover for Kleene Algebra with Tests*. *Journal of Applied Non-Classical Logics* 16(1–2), pp. 9–33, doi:10.3166/jancl.16.9-33.

[2] Marco Almeida (2011): *Equivalence of regular languages: an algorithmic approach and complexity analysis*. Ph.D. thesis, Faculdade de Ciências das Universidade do Porto. http://www.dcc.fc.up.pt/~mfa/thesis.pdf.

[3] Marco Almeida, Nelma Moreira & Rogério Reis (2009): *Antimirov and Mosses's rewrite system revisited*. *International Journal of Foundations of Computer Science* 20(04), pp. 669 – 684, doi:10.1142/S0129054109006802.

[4] Valentin M. Antimirov (1996): *Partial Derivatives of Regular Expressions and Finite Automaton Constructions*. *Theoret. Comput. Sci.* 155(2), pp. 291–319, doi:10.1016/0304-3975(95)00182-4.

[5] Valentin M. Antimirov & Peter D. Mosses (1994): *Rewriting Extended Regular Expressions*. In G. Rozenberg & A. Salomaa, editors: *Developments in Language Theory*, World Scientific, pp. 195 – 209.

[6] S. Ajesh Babu & Paritosh K Pandya (2011): *Chop Expressions and Discrete Duration Calculus*. In: *Modern Applications of Automata Theory*, 2, World Scientific, pp. 1–30.

[7] H. Chen & R. Pucella (2004): *A coalgebraic approach to Kleene algebra with tests. Theor. Comp. Sci.* 327(1-2), pp. 23–44, doi:10.1016/j.tcs.2004.07.020.

[8] Ernie Cohen, Dexter Kozen & Frederick Smith (1996): *The complexity of Kleene algebra with tests.* Technical Report TR96-1598, Computer Science Department, Cornell University.

[9] Project FAdo (Access date:1.1.2012): *FAdo: tools for formal languages manipulation.* http://fado.dcc.fc.up.pt.

[10] Maria João Frade & Jorge Sousa Pinto (2011): *Verification conditions for source-level imperative programs. Computer Science Review* 5(3), pp. 252–277, doi:10.1016/j.cosrev.2011.02.002.

[11] C. A. R. Hoare (1969): *An axiomatic basis for computer programming. Comm. of the ACM* 12(10), pp. 576–580, doi:10.1145/357980.358001.

[12] Peter Hófner & Georg Struth (2007): *Automated Reasoning in Kleene Algebra.* In F. Pfenning, editor: *CADE 2007*, *LNAI* 4603, Springer-Verlag, pp. 279–294, doi:10.1007/978-3-540-73595-3-19.

[13] Dexter Kozen (1994): *A completeness theorem for Kleene algebras and the algebra of regular events. Infor. and Comput.* 110(2), pp. 366–390, doi:10.1006/inco.1994.1037.

[14] Dexter Kozen (1997): *Kleene algebra with tests. Trans. on Prog. Lang. and Systems* 19(3), pp. 427–443, doi:10.1145/256167.256195.

[15] Dexter Kozen (2000): *On Hoare logic and Kleene algebra with tests. Trans. Comput. Logic* 1(1), pp. 60–76, doi:10.1145/343369.343378.

[16] Dexter Kozen (2003): *Automata on Guarded Strings and Applications. Matématica Contemporânea* 24, pp. 117–139.

[17] Dexter Kozen (2008): *On the coalgebraic theory of Kleene algebra with tests.* Computing and Information Science Technical Reports http://hdl.handle.net/1813/10173, Cornell University.

[18] Dexter Kozen & Frederick Smith (1996): *Kleene algebra with tests: Completeness and decidability.* In D. van Dalen & M. Bezem, editors: *Proc. 10th CSL*, *LNCS* 1258, Springer-Verlag, pp. 244–259, doi:10.1007/3-540-63172-0-43.

[19] Boris Mirkin (1966): *An Algorithm for Constructing a Base in a Language of Regular Expressions. Engineering Cybernetics* 5, pp. 110–116.

[20] Jan J. M. M. Rutten (1998): *Automata and Coinduction (An Exercise in Coalgebra).* In Davide Sangiorgi & Robert de Simone, editors: *CONCUR'98 Concurrency Theory*, *LNCS* 1466, Springer, pp. 194–218, doi:10.1007/BFb0055624.

[21] The Caml team (Access date: 1.5.2012): *Ocaml.* http://caml.inria.fr/ocaml/.