KAT and PHL in Coq

David Pereira[1] and Nelma Moreira[1]

LIACC – University of Porto
{dpereira,nam}@ncc.up.pt

Abstract. In this paper we describe an implementation of Kleene Algebras with Tests (KAT) in the Coq theorem prover. We also present the Propositional Hoare Logic (PHL) encoding in KAT, by deriving its deduction rules as theorems of KAT. This work is part of a study of the feasibility of using KAT for to the automatic production of certificates in the context of (source-level) Proof-Carrying-Code (PCC).

1 Introduction

A Kleene Algebra with Tests (KAT) is an algebraic structure developed by Kozen [1] which is the combination of a Boolean Algebra (BA) and Kleene Algebra (KA) [2]. This algebraic structure permits the interaction of Boolean tests and regular events in a very compact way. It was shown to be highly expressive, which allow it to be applied to several formal verification tasks involving communication protocols, basic safety analysis, source-to-source program transformation, concurrency control, compiler optimization and data-flow analysis [3–5].

In this paper we present an approach to program verification using KAT by proving in the Coq theorem prover [6] that Propositional Hoare Logic (PHL) [7, 8] rules are theorems of KAT.

This paper is organized as follows. In Section 2 we briefly present the theory of KAT and PHL. In Section 3 we show how we have implemented KAT and PHL in the Coq theorem prover. In Section 4 we relate the work presented here with the enforcement of security mechanisms for Embedded Systems' software.

Finally, in Section 5 we draw some conclusions and point some directions to future research.

2 Revision of KAT and PHL concepts

As defined by Kozen in [1], a KAT is a KA combined with a BA where the elements of the latter are called *tests*. Therefore, a KAT is an algebraic structure

$$(K, B, +, \cdot, ^*, ^-, 0, 1)$$

with $B \subseteq K$ and where

$$(K, +, \cdot, ^*, 0, 1)$$

is a KA and

$$(B, +, \cdot, ^-, 0, 1)$$

is a BA. Both the KA and the BA satisfy the axioms (1)-(11) presented below. In addition, KA satisfies the axioms (12)-(15) and BA satisfies the usual Boolean axioms (16)-(24). As usual we omit the $\cdot$ operator. The Kleene Star operator * has the higher precedence level, and $+$ the lower precedence level, in the KA context. In the BA context, $^-$ has the higher precedent and the rest remains the same.

Let $p, q, r \in K$ and $b, c \in B$ for any given KAT. The axioms for KAT are the following:

$$p + (q + r) = (p + q) + r \tag{1}$$

$$p + q = q + p \tag{2}$$

$$p + 0 = p \tag{3}$$

$$p + p = p \tag{4}$$

$$p(qr) = (pq)r \tag{5}$$

$$1p = p \tag{6}$$

$$p1 = p \tag{7}$$

$$p(q + r) = pq + pr \tag{8}$$

$$(p + q)r = pr + qr \tag{9}$$

$$0p = 0 \tag{10}$$

$$p0 = 0 \tag{11}$$

$$1 + pp^* = p^* \tag{12}$$

$$1 + p^*p = p^* \tag{13}$$

$$q + pr \leq r \rightarrow p^*q \leq r \tag{14}$$

$$q + rp \leq r \rightarrow qp^* \leq r \tag{15}$$

$$\bar{1} = 0 \tag{16}$$

$$\bar{0} = 1 \tag{17}$$

$$bc = cb \tag{18}$$

$$b + 1 = 1 \tag{19}$$

$$b + \bar{b} = 1 \tag{20}$$

$$b\bar{b} = 0 \tag{21}$$

$$\bar{\bar{b}} = b \tag{22}$$

$$\overline{(b + c)} = \overline{bc} \tag{23}$$

$$\overline{(bc)} = \bar{b} + \bar{c} \tag{24}$$

where $\leq$ is the natural partial order on K, defined as

$$p \leq q \leftrightarrow p + q = q.$$

The language of KAT is very expressive and compressed and permits us to represent both (simple) imperative programs and its properties, such as Propositional Hoare Logic (PHL) rules). The PHL deductive rules are theorems of KAT, *i.e.*, can be derived using the above axioms of KAT and a usual deduction system.

Let p and q be elements of K, and b elements of B, for any KAT. Consider the following simple *while* language without assertions:

$$\begin{aligned} p ::= {} & p; q \mid \\ & \textbf{if } b \textbf{ then } p \textbf{ else } q \mid \\ & \textbf{while } b \textbf{ do } p \end{aligned}$$

These rules can be easily translated into KAT expressions as follows:

$$p; q = pq$$

$$\textbf{if } b \textbf{ then } p \textbf{ else } q = bp + \bar{b}q$$

$$\textbf{while } b \textbf{ do } p = (bp)^* \bar{b}$$

The basic assertion of Hoare Logic is the *Partial Correctness Assertion* (PCA)

$$\{b\}p\{c\}$$

where b and c are formulas and p is a program. The intuition behind this assertion is that whenever the formula b holds before executing p, then if p halts, it will halt in an output state where c will be true. However, this condition does not force the program p to halt. In KAT, the PCA is written as

$$bp\bar{c} = 0$$

or, equivalently [8], as

$$bpc = bp.$$

The translation of the four deduction rules of PHL is as follows:

Composition rule:

$$\frac{\{\mathrm{b}\}\mathrm{p}\{\mathrm{c}\} \qquad \{\mathrm{c}\}\mathrm{q}\{\mathrm{d}\}}{\{\mathrm{b}\}\mathrm{p};\mathrm{q}\{\mathrm{d}\}}$$

$$\Leftrightarrow$$

$$bp = bcp \wedge cq = cqd \rightarrow bpq = bpqd$$

Conditional rule:

$$\frac{\{\mathrm{b} \wedge \mathrm{c}\}\,\mathrm{p}\,\{\mathrm{d}\} \qquad \{\neg\ \mathrm{b} \wedge \mathrm{c}\}\,\mathrm{q}\,\{\mathrm{d}\}}{\{\mathrm{c}\}\textbf{if } \mathrm{b} \textbf{ then } \mathrm{p} \textbf{ else } \mathrm{q}\,\{\mathrm{d}\}}$$

$$\Leftrightarrow$$

$$bcp = bcpd \wedge \bar{b}cq = \bar{b}cqd \rightarrow c(bp + \bar{b}q) = c(bp + \bar{b}q)d$$

While rule:

$$\frac{\{\mathrm{b} \wedge \mathrm{c}\}\,\mathrm{p}\,\{\mathrm{c}\}}{\textbf{while}\,\mathrm{b}\,\textbf{do}\,\mathrm{p}\,\{\neg\,\mathrm{b} \wedge \mathrm{c}\}}$$

$$\Leftrightarrow$$

$$bpc = bcpc \rightarrow c(bp)^*\bar{b} = c(bp)^*\bar{b}\bar{b}c$$

Weakening rule:

$$\frac{\mathrm{b'} \rightarrow \mathrm{b} \qquad \{\mathrm{b}\}\,\mathrm{p}\,\{\mathrm{c}\} \qquad \mathrm{c} \rightarrow \mathrm{c'}}{\{\mathrm{b'}\}\,\mathrm{p}\,\{\mathrm{c'}\}}$$

$$\Leftrightarrow$$

$$b' \leq b \wedge bp = bpc \wedge c \leq c' \rightarrow b'p = b'pc'$$

3 Implementation of KAT in Coq

A KAT is a combination of BA and KA. Both share a set of axioms for the operations + and ., applied to both elements of B and K, respectively. In the current version of our implementation we use the Coq module system to define modules which represent a BA, a KAT and PHL, respectively.

3.1 Implementation of the BA module

The module `Boolean_Algebra_of_Test` represents a BA. It contains a type `B` representing the elements of B and an inductive type `BExpr` for representing BA expressions.

```
Module Boolean_Algebra_of_Tests.

  Parameter B : Set.

  Inductive BExpr : Set :=
  | BZero : BExpr
  | BOne  : BExpr
  | BTv   : B -> BExpr
  | BNeg  : BExpr -> BExpr
  | BPlus : BExpr -> BExpr -> BExpr
  | BDot  : BExpr -> BExpr -> BExpr.
```

The constants 0 and 1 are represented, respectively, by the constructors `BZero` and `BOne`. The constructor `BTv` establishes that elements of B are Boolean expressions by construction and the constructors `BNeg`, `BPlus` and `BDot` represent, respectively, the operators $^-$, $+$ and . of negation, disjunction and conjunction.

The implementation of this module is completed by establishing the axioms (1)-(11) and (16)-(22) in Coq, by just stating them as primitive properties of `BExpr` expressions.

```
Axiom BPlus_Assoc      : BPlus (BPlus x y) z = BPlus x (BPlus y z).
Axiom BPlus_Comm       : BPlus x y = BPlus y x.
Axiom BPlus_BZero      : BPlus x BZero = x.
Axiom BPlus_Idem       : BPlus x x = x.
Axiom BDot_Assoc       : BDot (BDot x y) z = BDot x (BDot y z).
Axiom BDot_BOne        : BDot x BOne = x.
Axiom BDot_BOne_Comm   : BDot x BOne = BDot BOne x.
Axiom BDot_Assoc_Left  : BDot (BPlus x y) z = BPlus (BDot x z) (BDot y z).
Axiom BDot_Assoc_Right : BDot x (BPlus y z) = BPlus (BDot x y) (BDot x z).
Axiom BDot_BZero       : BDot x BZero = BZero.
Axiom BDot_BZero_Comm  : BDot x BZero  = BDot BZero x.
Axiom BNeg_One         : BNeg BOne  = BZero.
Axiom BNeg_Zero        : BNeg BZero = BOne.
Axiom BIdemp           : BDot b b = b.
Axiom BDot_Comm        : BDot b c = BDot c b.
Axiom BPlus_Comp       : BPlus b (BNeg b) = BOne.
Axiom BDot_Comp        : BDot b (BNeg b) = BZero.
Axiom BDouble_Neg      : BNeg (BNeg b) = b.
Axiom BPlus_Neg        : BNeg (BPlus b c) = BDot (BNeg b) (BNeg c).
Axiom BDot_Neg         : BNeg (BDot b c) = BPlus (BNeg b) (BNeg c).
```

Besides these axioms, some authors use auxiliary axioms to characterize a BA. Two of the most used are the following, which here we proved as theorems:

```
Theorem  BPlus_One : BPlus b BOne = BOne.
Proof.
  intros b.
  rewrite <- (BPlus_Comp b).
  rewrite <- BPlus_Assoc.
  rewrite  BPlus_Idem.
  trivial.
Qed.

Theorem BPlus_Distr : BPlus b (BDot c d) = BDot (BPlus b c) (BPlus b d).
Proof.
  intros b c d.
  rewrite BDot_Assoc_Left.
  repeat rewrite BDot_Assoc_Right.
  rewrite BIdemp.
```

```
  rewrite <- (BDot_BOne b).
  rewrite BDot_Assoc.
  rewrite <- BDot_Assoc_Right.
  rewrite (BPlus_Comm BOne (BDot BOne d)).
  rewrite (BPlus_One).
  rewrite BDot_BOne.
  rewrite <- BPlus_Assoc.
  rewrite (BDot_Comm c b).
  rewrite <- (BDot_BOne b).
  rewrite (BDot_Assoc b BOne c).
  rewrite <- BDot_Assoc_Right.
  rewrite <- (BDot_BOne_Comm c).
  rewrite (BDot_BOne c).
  rewrite (BPlus_Comm BOne c).
  rewrite BPlus_One.
  rewrite BDot_BOne.
  trivial.
Qed.

End Boolean_Algebra_of_Tests.
```

3.2 Implementation of KAT module

The module `Kleene_Algebra_With_Tests` represents a KAT. We use the BA module for the Boolean part of KAT and define a new inductive type for representing KAT expressions. This type is named `K` and is presented below:

```
Module Kleene_Algebra_With_Tests.

  Module BAT := Boolean_Algebra_of_Tests.

  Notation B := BAT.B.
  Notation BExpr := BAT.BExpr.

  Parameter Sigma : Set.

  Inductive K : Set :=
  | One  : K
  | Zero : K
  | T    : BExpr -> K
  | S    : Sigma -> K
  | Star : K -> K
  | Dot  : K -> K -> K
  | Plus : K -> K -> K.
```

The constructors `Zero` and `One` represent the constants 0 and 1 respectively. The constructors `Dot`, `Plus` and `Star` represent the operators $+$, . and * re-

spectively. The constructor `S` is used to refer to elements of K in a KAT and the constructor `T` brings a BA expression of type `BExpr` into an element of type `K` which permits us to reason about the interaction of Boolean test and the elements of K.

We have also proved some theorems of KAT which enable us to prove that PHL rules are theorems of KAT. In particular, that the $\leq$ relation is an equivalence an it verifies the following properties.

$$\begin{array}{c} 1 \leq x^* \\ x \leq x^* \\ x^* x^* \leq x^* \end{array} \quad . \tag{25}$$

We proved also the following theorems of equality and of the $\leq$ relation.

$$\begin{array}{ll} x = y \wedge z = t \rightarrow (x + z) = (y + t) & x \leq y \wedge z \leq t \rightarrow (x + z) \leq (y + t) \\ x = y \wedge z = t \rightarrow xz = yt & x \leq y \wedge z \leq t \rightarrow xz \leq yt \\ x = y \rightarrow x^* = y^* & x \leq y \rightarrow x^* \leq y^* \end{array} \tag{26}$$

Here we present the proof of the monotonicity of expressions involving the Kleene's Star operator both on equality and on the $\leq$ relation.

```
Lemma eq_Mon_Star : x = y -> (Star x) = (Star y).
  Proof.
    intros x y H.
    rewrite H.
    reflexivity.
  Qed.

Definition le_Mon_Star : le x y -> le (Star x) (Star y).
  Proof.
    intros x y H.
    unfold le in *.
    rewrite <- H.
    rewrite Denesting. (* (x + y)^* = x^*(yx^*)^*  *)
    rewrite <- (Dot_One (Star x)).
    rewrite Dot_Assoc.
    rewrite <- Dot_One_Comm.
    rewrite Dot_One_Comm.
    rewrite <- (Dot_Assoc y One (Star x)).
    rewrite (Dot_One y).
    rewrite (Dot_One (Star (Dot y (Star x)))).
    rewrite <- Dot_One_Comm.
    rewrite <- Dot_Assoc_Right.
    rewrite One_le_Star. (* 1 <= x^* *)
```

```
    reflexivity.
  Qed.
```

Besides these properties, we also proved other equivalences between KAT expressions such as the useful theorems of *bisimulation*, *sliding* and *denesting*. These three theorems of KAT are respectively defined as follows:

$$xy = yz \rightarrow x^*y = yz^* \tag{27}$$

$$(xy)^*x = x(yx)^* \tag{28}$$

$$(x + y)^* = x^*(yx^*)^* \tag{29}$$

The importance of these theorems is mostrly related to the treatment of the Kleene star operator under the representation of square matrices within KA. If the elements of set where the matrices are built from is a KA, then these matrices become themselves KA, as presented by Kozen in [9].

3.3 Implementation of PHL module

The module `Propositional_Hoare_Logic` represents the PHL implementation in the language of KAT. This module is an extension of the `Kleene_Algebra_With_Tests` where we proved that rules of PHL are theorems of KAT. For purposes of simplifying the implementation we defined new notations to the sets K and B, and to the Boolean and KAT inductive types.

```
Module Propositional_Hoare_Logic.

  Module KAT := Kleene_Algebra_With_Tests.

  Notation K := KAT.K.
  Notation B := KAT.B.
  Notation BExpr := KAT.BExpr.
  Notation Sigma := KAT.Sigma.
```

The base for the implementation of PHL is the PCA condition. This condition can be represented in two equivalent ways in KAT. We take one as definition and we proved their equivalence.

```
Definition PCA (b c : BExpr) (x : K):= Dot (Dot (T b) x) (T (BNeg c)) = Zero.

Lemma PCA_Equiv_Left : PCA b c x -> Dot (T b) x = Dot (Dot (T b) x) (T c).
Proof.
  intros b c x H.
  rewrite KAT.Dot_Assoc.
  rewrite <- (KAT.Dot_One (Dot (T b) x)).
  rewrite <- KAT.Eq_Expr_Bool_KAT_1.
  rewrite <- (KAT.BAT.BPlus_Comp c).
```

```
  rewrite KAT.Eq_Expr_Bool_KAT_4.
  rewrite KAT.Dot_Assoc_Right.
  rewrite H.
  rewrite KAT.Plus_Zero.
  rewrite KAT.Dot_Assoc.
  trivial.
Qed.

Lemma PCA_Equiv_right : Dot (T b) x = Dot (Dot (T b) x) (T c) -> PCA b c x.
Proof.
  intros b c x H.
  unfold PCA.
  rewrite H.
  repeat rewrite KAT.Dot_Assoc.
  rewrite <- KAT.Eq_Expr_Bool_KAT_3.
  rewrite KAT.BAT.BDot_Comp.
  rewrite KAT.Eq_Expr_Bool_KAT_2.
  repeat rewrite KAT.Dot_Zero.
  trivial.
Qed.
```

The rest of the implementation consisted on the proofs that PHL rules are theorems of KAT. Here we present the case for composition.

```
Definition Composition :
    Dot (T b) x = Dot (Dot (T b) x) (T c) /\ Dot (T c) y = Dot (Dot (T c) y) (T d)
                          ->
    Dot (Dot (T b) x) y = Dot (Dot (T b) x) (Dot y (T d)).
  Proof.
    intros b c d x y H.
    elim H.
    intros H0 H1.
    rewrite H0.
    rewrite KAT.Dot_Assoc.
    rewrite H1.
    repeat rewrite <- KAT.Dot_Assoc.
    trivial.
  Qed.

 Definition Conditional :
     Dot (Dot (T b) (T c)) x = Dot (Dot (Dot (T b) (T c)) x) (T d)
                             /\
     Dot (Dot (T (BNeg b)) (T c)) y = Dot (Dot (Dot (T (BNeg b)) (T c)) y) (T d)
                             ->
     Dot (T c) (Plus (Dot (T b) x) (Dot (T (BNeg b)) y))
                             =
```

```
    Dot (Dot (T c) (Plus (Dot (T b) x) (Dot (T (BNeg b)) y))) (T d).
Proof.
  intros a b c x y H.
  elim H.
  intros H0 H1.
  rewrite KAT.Dot_Assoc_Right.
  repeat rewrite  <- KAT.Dot_Assoc.
  rewrite <- KAT.Eq_Expr_Bool_KAT_3.
  rewrite (KAT.BAT.BDot_Comm c b).
  rewrite KAT.Dot_Assoc_Left.
  rewrite KAT.Eq_Expr_Bool_KAT_3.
  rewrite H0.
  rewrite (KAT.Dot_Assoc (Dot (Dot (T b) (T c)) x) (T d) (T d)).
  repeat rewrite <- KAT.Eq_Expr_Bool_KAT_3.
  rewrite (KAT.BAT.BIdemp).
  rewrite (KAT.BAT.BDot_Comm c (BNeg b)).
  repeat rewrite KAT.Eq_Expr_Bool_KAT_3.
  rewrite H1.
  rewrite (KAT.Dot_Assoc (Dot (Dot (T (BNeg b)) (T c)) y) (T d) (T d)).
  repeat rewrite <- KAT.Eq_Expr_Bool_KAT_3.
  rewrite (KAT.BAT.BIdemp).
  repeat rewrite KAT.Eq_Expr_Bool_KAT_3.
  trivial.
Qed.
```

4 Applications

We have developed a set of Coq modules that formally implement the KAT theory and shown its application to program verification using PHL. We did this with the aim of studying the feasibility of using KAT in the context of enforcement of security mechanisms for Embedded Systems, in particular, Proof-Carrying-Code systems.

Proof-Carrying-Code aims at providing static and decentralized security enforcement mechanisms based on the notion of verifiable evidence, usually defined as certificate. The key idea of Proof-Carrying-Code is to attach to a (mobile) code file, an easily checkable proof (certificate) that its execution does not violate certain safety policies.

One possibility is to use KAT as the formal system to write certificates for programs, even at the source code level of the application. KAT has a very compact representation and there exists automatic procedures to decide KAT equations, as presented by Worthington in [10] and Kozen in [11]. This means that we can automate the production of certificates in Coq by implementing a tactic which automatically proves KAT properties which we might be interessed in the contect of mobile code security. The study of the computational costs of having a Proof-Carrying-Code system whose certificate production is based on

KAT, using the Coq interactive theorem prover. The computational cost of the implementation of such a system will be subject to future research.

5 Conclusion

In this paper we have presented an implementation of KAT in the Coq theorem prover. We also presented how the implementation of the KAT model served as the basis for the implementation of a Coq module for verifying simple imperative programs using KAT by applying the PHL rules written in the language of KAT.

As we have seen, simple imperative languages have somehow direct translations into KAT expressions. Moreover, KAT is very compact and needs small computation power for being analysed and processed for proof correctness. This aspect is fundamental when considering Embedded Systems, which are now more in the mobile systems concept and with considerable limitations either in computational power and lack of energy for keeping alive and executing. The implementation we have presented here can be seen as the starting point for such a system.

For future work, we intend to augment the implementation we have, by proving more properties and develop a certified algorithm which decides if two KAT expressions are equivalent, in an automatic way, probably in the line of the work presented by Worthington [10]. Such algorithm will facilitate the introduction of KAT in the realm of new computer paradigms, as is the case of Proof-Carrying-Code.

6 Acknowledgment

This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) and program POSI.

References

1. Kozen, D.: Kleene algebra with tests. Transactions on Programming Languages and Systems **19**(3) (May 1997) 427–443
2. Kozen, D.: On Kleene algebras and closed semirings. In Rovan, ed.: Proc. Math. Found. Comput. Sci. Volume 452 of Lecture Notes in Computer Science., Banska-Bystrica, Slovakia, Springer-Verlag (1990) 26–47
3. Barth, A., Kozen, D.: Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report TR2002-1865, Computer Science Department, Cornell University (June 2002)
4. Kozen, D., Patron, M.C.: Certification of compiler optimizations using kleene algebra with tests. In: Computational Logic. (2000) 568–582
5. Angus, A., Kozen, D.: Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Cornell University (2001)
6. Bertot, Y., Castéran, P.: Interactive theorem proving and program development. coq'art: The calculus of inductive constructions (2004)

7. Kozen, D., Tiuryn, J.: On the completeness of propositional hoare logic. In: RelMiCS. (2000) 195–202
8. Kozen, D.: On Hoare logic and Kleene algebra with tests. Trans. Computational Logic **1**(1) (July 2000) 60–76
9. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Infor. and Comput. **110**(2) (May 1994) 366–390
10. Worthington, J.: Automatic Proof Generation in Kleene Algebra. In: Relations and Kleene Algebra in Computer Science. Volume 4988 of LNCS. Springer Berlin / Heidelberg (2008) 382–396
11. Kozen, D.: On the coalgebraic theory of kleene algebra with tests. Computing and information science technical reports, Cornell University (March 2008)