A Kleene Algebra with Tests for Union Bound Reasoning About Probabilistic Programs

Leandro Gomes
Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000, France

Patrick Baillot
Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000, France

Marco Gaboardi
Boston University, MA, USA

Abstract

Kleene Algebra with Tests (KAT) provides a framework for algebraic equational reasoning about imperative programs. The recent variant Guarded KAT (GKAT) allows to reason on non-probabilistic properties of probabilistic programs. Here we introduce an extension of this framework called approximate GKAT (aGKAT), which equips GKAT with a partially ordered monoid (real numbers) enabling to express satisfaction of (deterministic) properties *except* with a probability up to a certain bound. This allows to represent in equational reasoning "à la KAT" proofs of probabilistic programs based on the union bound, a technique from basic probability theory. We show how a propositional variant of approximate Hoare Logic (aHL), a program logic for union bound, can be soundly encoded in our system aGKAT. We then illustrate the use of aGKAT with an example of accuracy analysis from the field of differential privacy.

2012 ACM Subject Classification Theory of computation → Algebraic semantics; Theory of computation → Pre- and post-conditions; Theory of computation → Logic and verification; Theory of computation → Hoare logic

Keywords and phrases Kleene algebras with tests, Hoare logic, equational reasoning, probabilistic programs, union bound, formal verification

Digital Object Identifier 10.4230/LIPIcs.CSL.2025.35

Related Version *Extended Version*: `https://hal.science/hal-04196675v3` [13]

Funding *Leandro Gomes*: This work is financed by National Funds through FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project IBEX, with reference 10.54499/PTDC/CCI-COM/4280/2021.
Patrick Baillot: Work partially supported by ANR Project HOPR (ANR-24-CE48-5521-01).

1 Introduction

Kleene algebra with tests (KAT) has been introduced in [21] as an algebraic framework for program verification. A KAT is a two-sorted structure, consisting of a Kleene algebra and a Boolean algebra of tests: the Kleene algebra part accounts for programs, with sequential composition, branching and iteration; the Boolean algebra part accounts for the predicates used to build if-then-else instructions, while loops and assertions, as well as, being KAT able to subsume propositional Hoare logic [22], for the pre and post-conditions. This framework allowed to give algebraic proofs corresponding to several approaches in program verification, see e.g. [22, 1, 24], and has been implemented as a library for the Coq proof assistant [28]. It has also been followed by several variants, like NetKAT [2], which allows to reason about software defined networks, Concurrent NetKAT [31] for concurrent networks, CKAO [18] for concurrent programs and more recently TopKAT for reasoning about incorrectness [32].

Recently the variant Guarded KAT (GKAT) [30] has been proposed as a restriction of KAT where all sums and iterations are guarded by tests. It offers several advantages over KAT, including the fact that the complexity of its equational theory is lower (almost linear time, provided that the number of tests is fixed) and the existence of a probabilistic model. The latter paves the way for using GKAT for reasoning about probabilistic programs. However an important feature of this system is that the tests of GKAT remain the same as those of KAT, namely they express Boolean properties on states. Therefore the framework of GKAT allows to encode probabilistic programs, but the assertions about them are non-probabilistic.

In this paper our goal is to extend the GKAT approach to reason about probabilistic programs satisfying properties *with a given probability bound* β. The objective is not to design an expressive framework for advanced probabilistic proofs, but instead to allow for simple probabilistic reasoning with a low technical overhead.

Concretely we target proofs based on the *union bound principle*, a property from basic probability theory, which is a simple consequence of the definition of probability measure, and can be stated as follows: given some properties $A_1, \ldots, A_n$, one has $Pr[\cup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} Pr[A_i]$. This principle is ubiquitous when reasoning about properties of randomized algorithms [27] and in their application in security, privacy [9], learning theory [19], etc.

A previous approach for reasoning about probabilistic imperative programs using the union bound principle had been provided by the *union bound program logic* aHL [5]. This is a Hoare logic for reasoning about probabilistic programs with non-probabilistic assertions but with judgements carrying a numeric index for tracking the failure probability. That is, judgments have the form $\vdash_\beta c : \phi \Rightarrow \psi$ where β is an upper bound on the probability that $\neg\psi$ is true after executing c starting from a memory satisfying ϕ. The authors illustrated how this logic could be used for the verification of accuracy of some algorithms, in particular in the setting of differential privacy. A relational variant of this logic is handled by the Easycrypt tool [8, 4] , which can be used for proving properties of cryptographic protocols as well as differential privacy properties of programs.

A natural idea is thus to adapt the union bound logic aHL to the GKAT framework. To do this and capture the union bound reasoning in an algebraic framework we extend GKAT with an additional relation, denoted $\lhd$, relating GKAT expressions with elements of a partially ordered monoid, typically real numbers on $[0, 1]$. We call this new system *approximate* GKAT (aGKAT). An important feature of this structure is that we want the new setting to subsume standard GKAT, requiring aGKAT to satisfy the theory of GKAT. A second feature is that we want the probabilistic model of sub-Markov kernels to be a model of our new structure, when we consider the monoid of real numbers. For this particular instantiation, the meaning of $\lhd$ will be that $c \lhd \beta$ holds if *the probability of successful execution of program* c *is bounded by* β. The theory of aGKAT extends the one of GKAT, by a small set of axioms characterizing the properties of the new relation $\lhd$. We illustrate how this theory allows for a concise form of equational reasoning for establishing probability bounds on some GKAT programs. Moreover, in order to demonstrate the expressivity of aGKAT, we encode aHL in it. This is inspired by the classical result of Kozen [22] showing that propositional Hoare logic can be encoded in KAT.

Outline. In Sect. 2 we will recall GKAT and its probabilistic model, and in Sect. 3 we will recall the Hoare logic aHL. Then in Sect. 4 we will define our system, aGKAT, its theory and its semantics. After that in Sect. 5 we will provide an encoding of the logic aHL in aGKAT and prove its soundness. Sect. 6 will be devoted to an example, the analysis in aGKAT of the accuracy of the probabilistic algorithm Report-noisy-max. Finally, Sec. 7 overviews related work and Sec. 8 enumerates possible directions for future work.

2 Guarded Kleene algebra with tests

This section recalls the language and the semantics of *Guarded Kleene Algebra with Tests* (GKAT) [30], an abstraction of imperative programs where conditionals and loops are encoded as guarded sums $(c_1 +_b c_2)$ and guarded iterations $(c^{(b)})$, respectively, guarded by Boolean predicates b. The structure is a restriction of KAT in which we are not allowed to freely use operators $+$ and * to build terms. In other words, GKAT does not allow nondeterminism.

2.1 Syntax

The syntax of GKAT is defined with a set of *actions* Σ and a finite set of primitive tests $\mathtt{T}$, which are disjoint. We denote actions by a and primitive tests by p. The sets of Boolean expressions $\mathtt{BExp}$ (also called tests) and GKAT expressions $\mathtt{Exp}$ (also called programs) are then defined by the following grammars:

$b, b_1, b_2 \in \mathtt{BExp} ::=$	
0	**false**
1	**true**
$p \in \mathtt{T}$	p
$b_1 \cdot b_2$	b_1 **and** b_2
$b_1 + b_2$	b_1 **or** b_2
$\bar{b}$	**not** b

$c, c_1, c_2 \in \mathtt{Exp} ::=$	
$a \in \Sigma$	**do** a
$b \in \mathtt{BExp}$	**assert** b
$c_1 \cdot c_2$	$c_1; c_2$
$c_1 +_b c_2$	**if** b **then** c_1 **else** c_2
$c^{(b)}$	**while** b **do** c

where, for any $b, b_1, b_2 \in \mathtt{BExp}$, operators $\cdot$, $+$ and $^-$ denote conjunction, disjunction and negation, respectively, and, for any $c, c_1, c_2 \in \mathtt{Exp}$, the operator $\cdot$ denotes sequential composition. The notations on the r.h.s. are given to help intuition and will sometimes be used when writing programs. We introduce command **skip** as a shorthand for **assert** 1, which is encoded by the Boolean expression 1.
The precedence of the operators is the usual one, i.e. the operator $\cdot$ has higher precedence than operator $+_b$, and $()^{(b)}$ has higher precedence that $\cdot$.[1] To simplify the writing, we often omit the operator $\cdot$ by writing $c_1 c_2$ for the expression $c_1 \cdot c_2$, for any $c_1, c_2 \in \mathtt{Exp}$.

We are interested in using GKAT for representing probabilistic programs. For that, let us first fix a few definitions. Given a set S, $\mathcal{D}(S)$ is the set of *probability sub-distributions*[2] over S with countable support, i.e. the set of functions $f : S \to [0, 1]$ such that $Supp(f) = \{x \in S \mid f(x) > 0\}$ is countable and f sums up to at most 1, i.e. $\sum_{s \in S} f(s) \leq 1$. In particular, the *Dirac* distribution $\delta_s \in \mathcal{D}(S)$ is the map $w \to [w = s] = \begin{cases} 1, \text{ if } w = s \\ 0, \text{ otherwise} \end{cases}$

▶ **Example 1** (Imperative programming language)**.** Take a set $\mathtt{Var}$ of variables and a set $\mathtt{Distr}$ of sub-distributions over $\mathbb{R}$ with discrete support. Consider a simple imperative programming language defined by the following grammar:

$$\textit{terms } t \in \mathtt{Terms} ::= x \in \mathtt{Var} \mid r \in \mathbb{R} \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2$$
$$\textit{distributions } d \in \mathtt{Distr}$$
$$\textit{tests } b \in \mathtt{Tests} ::= \textbf{false} \mid \textbf{true} \mid t_1 < t_2 \mid t_1 = t_2 \mid \textbf{not } b \mid b_1 \textbf{ and } b_2 \mid b_1 \textbf{ or } b_2$$

[1] For example the GKAT expression $c_1^{(b_1)} \cdot c_2 +_{b_2} c_3$ reads as $((c_1^{(b_1)}) \cdot c_2) +_{b_2} c_3$.
[2] Some examples of distributions are the tossing of a fair coin, with probability 0.5 for 0 and 1, and the (discrete version of the) Laplacian distribution $\mathcal{L}_p(a)$ centered in a with parameter p. The density function of $\mathcal{L}_p(a)$ is given by $\frac{1}{2p} \exp(\frac{|x-a|}{p})$.

$$\textit{commands } c \in \texttt{Comm} ::= \textbf{skip} \mid x \leftarrow t \mid x \overset{\$}{\leftarrow} d \mid c_1; c_2 \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } b \textbf{ do } c$$

This language can be modeled in GKAT by taking as sets of actions and primitive tests respectively $\Sigma = \{x \leftarrow t,\ x \overset{\$}{\leftarrow} d \mid x \in \texttt{Var}, t \in \texttt{Terms}, d \in \texttt{Distr}\}$ and $\texttt{T} = \{t_1 < t_2,\ t_1 = t_2 \mid t_1, t_2 \in \texttt{Terms}\}$ [3] . The first action evaluates term t and assigns the result to x and the second one samples from d and assigns the result to x. Observe that while programs c may be probabilistic, due to the use of samplings, the tests b as for them are deterministic, i.e. they do not use any probabilistic primitives. In particular the conditional branching in programs is only done on deterministic tests.

2.2 Semantics

We now present the semantic interpretation of GKAT that we will be using, the *Probabilistic model* [30] [4]. We first review some basic concepts needed for the semantics. The *Iverson bracket* $[b]$, for $b \in \texttt{BExp}$, is the function taking value 1 if b is true and 0 otherwise. Typical models of probabilistic imperative programming languages interpret programs as *Markov kernels* on a set S, i.e. maps from S to probability distributions. The semantic model defined below interprets programs as *sub-Markov* kernels, i.e. Markov kernels on sub-distributions.

▶ **Definition 2** (Probabilistic interpretation). *Let $i = (\texttt{State}, eval, sat)$ be a triple where:*

- *$\texttt{State}$ is a set of states,*
- *for each action $a \in \Sigma$, $eval(a) : \texttt{State} \to \mathcal{D}(\texttt{State})$ is a sub-Markov kernel,*
- *for each primitive test $p \in \texttt{T}$, $sat(p) \subseteq \texttt{State}$ is a set of states.*

The *probabilistic interpretation* of an expression c with respect to i is the sub-Markov kernel $\mathcal{P}_i[\![c]\!] : \texttt{State} \to \mathcal{D}(\texttt{State})$ defined as follows:

1. $\mathcal{P}_i[\![a]\!] := eval(a)$
2. $\mathcal{P}_i[\![b]\!](\sigma) := [\sigma \in sat^\dagger(b)] \times \delta_\sigma$
3. $\mathcal{P}_i[\![c_1 \cdot c_2]\!](\sigma)(\sigma') := \sum_{\sigma''} \mathcal{P}_i[\![c_1]\!](\sigma)(\sigma'') \times \mathcal{P}_i[\![c_2]\!](\sigma'')(\sigma')$
4. $\mathcal{P}_i[\![c_1 +_b c_2]\!](\sigma) := [\sigma \in sat^\dagger(b)] \times \mathcal{P}_i[\![c_1]\!](\sigma) + [\sigma \in sat^\dagger(\bar{b})] \times \mathcal{P}_i[\![c_2]\!](\sigma)$
5. $\mathcal{P}_i[\![c^{(b)}]\!](\sigma)(\sigma') := \lim_{n\to\infty} \mathcal{P}_i[\![(c +_b 1)^n \cdot \bar{b}]\!](\sigma)(\sigma')$

where $\text{sat}^\dagger : \texttt{BExp} \to 2^{\texttt{State}}$ is the lifting of $\text{sat} : \texttt{T} \to 2^{\texttt{State}}$ to arbitrary Boolean expressions over $\texttt{BExp}$, and $\times$ denotes both multiplication on real numbers and the pointwise multiplication on sub-distributions. For instance the definition of $\mathcal{P}_i[\![b]\!](\sigma)$ means that it is either δ_σ if σ belongs to $sat^\dagger(b)$, or the constant sub-distribution equal to 0 otherwise. Intuitively $\mathcal{P}_i[\![c]\!](\sigma)(\sigma')$ is the probability that the execution of c on initial state σ terminates on state σ', and $\sum_{\sigma'} \mathcal{P}_i[\![c]\!](\sigma)(\sigma')$ is the probability that the execution of c on initial state σ terminates on a state (we then also say that it is a successful execution). Observe thus that we really need to consider sub-distributions and not only distributions. Let us recall that the existence of the limit in point 5. has been shown in [30].

▶ Remark 3 (Finite state case). In the case where $\texttt{State}$ is a finite set of size n, say $\{s_1, \ldots, s_n\}$ then a sub-Markov kernel f can be represented as an $n \times n$ matrix $\mathcal{M} = (a_{i,j})_{i,j\in[1,n]}$. Each coefficient $a_{i,j}$ is defined as $a_{i,j} = f(s_i)(s_j)$. So in particular the sum over each line is inferior or equal to 1. We denote $f = \mathcal{M}$. For tests b, the matrix $\mathcal{P}_i[\![b]\!]$ has only diagonal coefficients,

[3] Note that technically speaking according to the definition of GKAT the set $\texttt{T}$ should be chosen finite, which is not the case here, but as observed in [30] Sect. 2.3 Example 2.5 we can use a finite subset $\texttt{T}'$ of $\texttt{T}$ for reasoning on pairwise equivalence of programs which terminate.

[4] Note that more interpretations of GKAT are presented in [30], namely a relational model and a language model.

with value $a_{i,i} = 1$ if $s_i \in \mathrm{sat}^\dagger(b)$, $a_{i,i} = 0$ if s_i not in $\mathrm{sat}^\dagger(b)$. In the case of $c_1 \cdot c_2$, the matrix $\mathcal{P}_i[\![c_1 \cdot c_2]\!]$ is obtained by the matrix product of $\mathcal{P}_i[\![c_1]\!]$ and $\mathcal{P}_i[\![c_2]\!]$. See [13] for an example.

In the following we will consider programs over a finite set of variables $\mathtt{Var}$ and the set of states will be the set of *memories*, that is to say functions in $\mathtt{Var} \to D$ where D is the domain of variables (we can take for instance $D = \mathbb{Q}$, the rational numbers). If $x \in \mathtt{Var}$ and σ is a memory, then $\sigma[x \leftarrow t]$ is the memory identical to σ except that it maps x to the evaluation of t in memory σ.
The interpretation of actions $a \in \Sigma$ as sub-Markov Kernels is then given by $eval(x \leftarrow t)(\sigma) := \delta_{\sigma[x \leftarrow t]}$ and $eval(x \overset{\$}{\leftarrow} d)(\sigma) := \sum_{t \in Supp(d)} d(t) \cdot \delta_{\sigma[x \leftarrow t]}$.
In the sequel memories will often be denoted as m.

2.3 Axioms

The theory of GKAT introduced in [30] is given by the axioms from Fig. 1. Note in particular the fixpoint axiom (13). Intuitively, it says that if expression c_3 chooses (using guard b) between executing c_1 and looping again, and executing c_2, then c_3 is a b-guarded loop followed by c_2. However, the rule is not sound in general. In order to overcome this limitation,

$$c +_b c = c \quad (1)$$
$$c_1 +_b c_2 = c_2 +_{\bar{b}} c_1 \quad (2)$$
$$(c_1 +_{b_1} c_2) +_{b_2} c_3 = c_1 +_{b_1 \cdot b_2} (c_2 +_{b_2} c_3) \quad (3)$$
$$c_1 +_b c_2 = b \cdot c_1 +_b c_2 \quad (4)$$
$$c_1 \cdot c_3 +_b c_2 \cdot c_3 = (c_1 +_b c_2) \cdot c_3 \quad (5)$$
$$(c_1 \cdot c_2) \cdot c_3 = c_1 \cdot (c_2 \cdot c_3) \quad (6)$$
$$0 \cdot c = 0 \quad (7)$$
$$c \cdot 0 = 0 \quad (8)$$
$$1 \cdot c = c \quad (9)$$
$$c \cdot 1 = c \quad (10)$$
$$c^{(b)} = c \cdot c^{(b)} +_b 1 \quad (11)$$
$$(c +_{b_2} 1)^{(b_1)} = (b_2 \cdot c)^{(b_1)} \quad (12)$$
$$\frac{c_3 = c_1 \cdot c_3 +_b c_2}{c_3 = c_1^{(b)} \cdot c_2} \text{ if } E(c_1) = 0 \quad (13)$$

Figure 1 Axiomatisation of Guarded Kleene algebra with tests.

following [30] (Section 3.1, Definition 3.2), the side condition $E(c_1) = 0$ is introduced, ensuring that command c_1 is *productive*, i.e. that it performs some action. To this end, the function E is inductively defined as follows: $E(b) := b$, $E(a) := 0$, $E(c_1 +_b c_2) := b \cdot E(c_1) + \bar{b} \cdot E(c_2)$, $E(c_1 \cdot c_2) := E(c_1) \cdot E(c_2)$, $E(c^{(b)}) := \bar{b}$.
We can see $E(c)$ as the weakest test that guarantees that command c terminates successfully but does not perform any action.

Moreover, note particularly the following observation: in KAT the encoding $c_1(bc_2 + \bar{b}c_3) = c_1bc_2 + c_1\bar{b}c_3$ is not an **if-then-else** statement; it is rather a nondeterministic choice between executing c_1, then testing b and executing c_2, and executing c_1, then testing $\bar{b}$ and executing c_3. The corresponding encoding in GKAT would be $c_1(c_2 +_b c_3) = c_1c_2 +_b c_1c_3$, an equality which is actually *not* valid in GKAT. Since GKAT is restricted to deterministic programs, there is no valid correspondence between the KAT encoding, which is not an **if-then-else** statement, and the hypothetical correspondent GKAT encoding, which is not valid. That is why left distributivity does not hold in GKAT for any $c \in \mathtt{Exp}$; it only holds for the particular case of $c_1 \in \mathtt{BExp}$, i.e. if c_1 is a test.

We define the relation $\leq$ on tests as: $b_1 \leq b_2$ iff $b_1 + b_2 = b_2$. Contrarily to KAT [21], the relation $\leq$ is not defined on an arbitrary GKAT expression, only on tests. In [13] we recall additional derivable equations in GKAT from [30].

Since any test is a program ($\mathtt{BExp} \subseteq \mathtt{Exp}$), the grammar also allows to write expressions as $b_1 +_b b_2$, for any $b \in \mathtt{BExp}$. We thus establish the following proposition[5] (proof in [13]) which expresses the guarded sum $+_b$, for any $b \in \mathtt{BExp}$, in terms of the disjunction $+$ on tests.

▶ **Proposition 4.** *For any tests b, b_1, b_2 one has: $b_1 +_b b_2 = bb_1 + \bar{b}b_2$.*

By Boolean reasoning, we can observe that $bb + \bar{b}\bar{b} = 1$. This observation will be useful later to prove the soundness of some aHL rules in aGKAT.

We also state the following proposition (see [13] for the proof):

▶ **Proposition 5.** *For any tests b_1, b_2 one has: $b_1 + b_2 = b_1 +_{b_1} b_2$.*

3 Union bound logic - Approximate Hoare logic

In this section we recall *Approximate Hoare logic (aHL)* [5], a logic based on the union bound, a tool from probability theory for analyzing randomised algorithms. A judgment in aHL is of the form $\vdash_\beta c : \phi \Rightarrow \psi$ where: ϕ, ψ are first-order formulas representing non probabilistic pre- and post-conditions[6], respectively; β is a value in $[0, 1]$ and it is an upper bound on the probability that the post-condition ψ does *not* hold on the output distribution, assuming that ϕ holds on an initial memory m. We assume a probabilistic interpretation i and we will denote $m \models \phi$ if ϕ is valid in memory m. The validity of the judgement is thus stated by:

▶ **Definition 6** (Validity of aHL judgment)**.** *A judgment $\vdash_\beta c : \phi \Rightarrow \psi$ is valid if for every memory m such that $m \models \phi$, we have $\mathcal{P}_i[\![c]\!](m)[\bar{\psi}] \leq \beta$.*

Figure 2 presents the deduction rules of aHL. Let us comment on some of these rules. The rule (*Rand*) handles sampling from a distribution d; we can assume a postcondition ψ after the sampling, provided that under the assumption of precondition ϕ, the statement ψ fails with probability at most β.

The other rules are similar to standard Hoare logic rules annotated with suitable probability indexes β. The rule (*Seq*) says that when composing two programs c and c', the failure probabilities of the two programs with respect to their postconditions add together. The (*Cond*) rule states that if the two branches of the conditional have the same index β, then we can keep the index β for the conditional. In rule (*Weak*) the premise $\models \phi' \Rightarrow \phi$ means that, in any model, ϕ' implies ϕ. This (*Weak*) rule allows to strengthen the precondition, weaken the postcondition, and increase the index β (which means overapproximating the failure probability). The (*And*) rule can be seen as an application of the union bound principle. It enables to combine two postconditions by a conjunction, provided we add up the failure probabilities. As to the (*Or*) rule, it allows to take the disjunction of two preconditions, if they have the same failure probability, and keep this index for the disjunction. Note that thanks to the (*Weak*) rule we could also in (*Or*) consider two indexes β and β' in the premises, and their maximum in the conclusion (the same is also true for (*Cond*)). The rule (*False*) might first seem a bit strange as it allows to conclude false, but note that its index is 1, which means that false holds in the final memory with probability 0. Finally, considering the

[5] We thank the anonymous reviewer of another paper for pointing out to us the fact that this property is derivable in GKAT.

[6] Note that ϕ and ψ are properties of memories rather that properties of distributions over memories.

(*While*) rule, observe that it is slightly more restrictive than the corresponding classical one of Hoare logic. Its side conditions ensure that the loop terminates in at most k iterations except with probability $k\beta$. Its first side condition states that the variable b_v only takes non-negative integer values.

- *Skip*:

$$\frac{}{\vdash_0 \mathtt{skip} : \phi \Rightarrow \phi}$$

- *Assn*:

$$\frac{}{\vdash_0 \mathbf{do}\ x \leftarrow t : \phi[t/x] \Rightarrow \phi}$$

- *Seq*:

$$\frac{\vdash_\beta c : \phi \Rightarrow \phi' \quad \vdash_{\beta'} c' : \phi' \Rightarrow \phi''}{\vdash_{\beta+\beta'} c; c' : \phi \Rightarrow \phi''}$$

- *Weak*:

$$\frac{\models \phi' \Rightarrow \phi \quad \vdash_\beta c : \phi \Rightarrow \psi \quad \models \psi \Rightarrow \psi' \quad \beta \leq \beta'}{\vdash_{\beta'} c : \phi' \Rightarrow \psi'}$$

- *Or*:

$$\frac{\vdash_\beta c : \phi \Rightarrow \psi \quad \vdash_\beta c : \phi' \Rightarrow \psi}{\vdash_\beta c : \phi \vee \phi' \Rightarrow \psi}$$

- *While*:

$$\frac{b_v : \mathbb{N}, \qquad \models (\phi \wedge b_v = 0 \to \bar{b}),\ \vdash_\beta c : \phi \Rightarrow \phi,\ \forall_{\eta>0} : \vdash_0 c : \phi \wedge b \wedge (b_v = \eta) \Rightarrow (b_v < \eta)}{\vdash_{k\cdot\beta} \mathbf{while}\ b\ \mathbf{do}\ c : \phi \wedge (b_v \leq k) \Rightarrow \phi \wedge \bar{b}}$$

- *Rand*:

$$\frac{\forall m : m \models \phi \Rightarrow \mathcal{P}_i[\![x \xleftarrow{\$} d]\!](m)[\bar{\psi}] \leq \beta}{\vdash_\beta \mathbf{do}\ x \xleftarrow{\$} d : \phi \Rightarrow \psi}$$

- *Cond*:

$$\frac{\vdash_\beta c : \phi \wedge b \Rightarrow \psi \quad \vdash_\beta c' : \phi \wedge \bar{b} \Rightarrow \psi}{\vdash_\beta \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c' : \phi \Rightarrow \psi}$$

- *And*:

$$\frac{\vdash_\beta c : \phi \Rightarrow \psi \quad \vdash_{\beta'} c : \phi \Rightarrow \psi'}{\vdash_{\beta+\beta'} c : \phi \Rightarrow \psi \wedge \psi'}$$

- *False*:

$$\frac{}{\vdash_1 c : \phi \Rightarrow \bot}$$

Figure 2 Approximate Hoare Logic rules (aHL).

4 Approximate Guarded Kleene algebra with tests (aGKAT)

4.1 Definition and theory of aGKAT

Recalling that GKAT encodes only Boolean assertions on probabilistic programs, we want to extend this kind of reasoning in order to capture aHL properties. We want to define a structure which would allow to express the fact that a probabilistic program c satisfies a deterministic postcondition, *except* with a probability up to a certain bound. For that we will extend GKAT with a relation between a GKAT expression and a value β from a partially ordered set. Such a set is defined as follows:

▶ **Definition 7.** *A preordered double monoid (pod-monoid) is a* $\mathcal{M} = (M, \leq, \cdot, 1, +, 0)$ *where:*

- $\leq$ *is a preorder on* M*,*

- $(M, \cdot, 1)$ *and* $(M, +, 0)$ *are two monoid structures, whose operations* $\cdot$ *and* $+$ *are monotone w.r.t.* $\leq$.

Note that we do not include any axiom relating $\cdot$ and $+$. This structure is thus sufficient to model the probability bounds from aHL. In the sequel we will consider the pod-monoid consisting of the real unit interval $[0, 1]$ equipped with multiplication and addition truncated to 1, that is to say $\min((\beta_1 + \beta_2), 1)$, where $+$ is the ordinary addition.

We then give the main definition of this section.

▶ **Definition 8.** *Approximate GKAT, denoted as aGKAT, is an extension of GKAT with a pod-monoid* $\mathcal{M}$ *and a predicate symbol* $\triangleleft$ *on* $\mathtt{Exp} \times \mathcal{M}$. *The theory of aGKAT is the union of axioms of pod-monoid, and those of Fig. 1 and Fig. 3.*

$$(c_1 = c_2 \ \wedge \ c_1 \triangleleft \beta) \ \Rightarrow \ c_2 \triangleleft \beta \quad (14)$$

$$(c \triangleleft \beta_1 \ \wedge \ \beta_1 \leq \beta_2) \ \Rightarrow \ c \triangleleft \beta_2 \quad (15)$$

$$(c \cdot c_1 \triangleleft \beta_1 \ \wedge \ c \cdot c_2 \triangleleft \beta_2) \ \Rightarrow \ c \cdot (c_1 +_b c_2) \triangleleft \beta_1 + \beta_2 \quad (16)$$

$$(c_1 \triangleleft \beta_1 \ \wedge \ c_2 \triangleleft \beta_2) \ \Rightarrow \ c_1 \cdot c_2 \triangleleft \beta_1 \cdot \beta_2 \quad (17)$$

$$(c_1 \triangleleft \beta \ \wedge \ c_2 \triangleleft \beta) \ \Rightarrow \ c_1 +_b c_2 \triangleleft \beta \quad (18)$$

$$c \triangleleft 1 \quad (19)$$

$$0 \triangleleft 0 \quad (20)$$

Figure 3 Axioms on the relation $\triangleleft$.

Recall that the intended meaning of $\triangleleft$ in the case where $\mathcal{M} = ([0, 1], \leq, \cdot, 1, +, 0)$ is that $c \triangleleft \beta$ holds if the probability of successful execution of program c is bounded by β. Observe that the $\triangleleft$-axioms of Fig. 3 are arguably simple, as they are Horn clauses and none deal with guarded iteration $c^{(b)}$.

Let us explain some intuitions underlying these axioms. Axiom (19) simply says that any program has a probability of successful execution bounded by 1, while (20) states that program 0 (which is **assert false**) has probability 0 of successful execution. Axiom (15) says that the statement still holds if we increase the probability β_1. Axiom (14) states that programs which are equal (up to the GKAT axioms of Fig. 1) admit the same probability of successful execution. Axiom (18) says that if the two branches of a conditional admit a bound β for their successful execution, so does the conditional itself. As to Axiom (17), its meaning is that the probability of successful execution of the composition of two programs c_1 and c_2 is bounded by the product of the probabilities of successful execution of respectively c_1 and c_2.

Maybe the less intuitive axiom is Axiom (16). Note that the difference with Axiom (18) is that for Axiom (18) any initial state s either satisfies b or $\overline{b}$, and so only one branch of $c_1 +_b c_2$ is explored. By contrast in Axiom (16) any initial state s might lead by the probabilistic execution of c *both* to states satisfying b and to states satisfying $\overline{b}$, so triggering both branches of the conditional. We will come back to this axiom in Remark 12 below.

After these intuitive considerations, we now formally define a semantic interpretation of aGKAT as follows:

▶ **Definition 9.** *A probabilistic interpretation of aGKAT is obtained by extending a probabilistic interpretation* $\mathcal{P}_i$ *of GKAT given in Sect. 2.2 in the following way:*

- *we consider the triple* $i = (\mathtt{State}, eval, sat)$ *of Def. 2 interpreting GKAT,*
- *the pod-monoid* $\mathcal{M}$ *is interpreted as indicated above by* $([0, 1], \leq, \cdot, 1, +, 0)$ *where* $\cdot$ *is the product and* $+$ *the truncated sum,*
- *the predicate* $\triangleleft$ *is interpreted by the relation between sub-Markov kernels* f *and* $[0, 1]$*-reals* β *consisting in the pairs* (f, β) *satisfying* $\forall s \in \mathtt{State}$, $\Sigma_{s' \in \mathtt{State}} f(s)(s') \leq \beta$, *i.e. for any* s, *the total mass of the sub-distribution* $f(s)$ *is bounded by* β.

We still denote the interpretation of an expression c as $\mathcal{P}_i[\![c]\!]$.

If i is an interpretation and F a 1st-order formula on the signature consisting of terms in Exp and in real numbers and predicates $=$ and $\triangleleft$, we write $i \models F$ if F is valid in the model defined by i. By abuse we will simply write $\models F$ if i is clear from the context. So by the definition above we have in particular that $i \models c \triangleleft \beta$ if $\forall s \in \texttt{State}$, $\sum_{s' \in \texttt{State}} \mathcal{P}_i[\![c]\!](s)(s') \leq \beta$.

We now establish the following proposition.

▶ **Proposition 10** (Soundness of aGKAT). *Any probabilistic interpretation of aGKAT is a model of its theory, i.e.:*

1. *the interpretation of aGKAT expressions satisfies the axioms of GKAT (Fig. 1) and that of $([0,1], \leq, \cdot, 1, +, 0)$ satisfies the axioms of pod-monoid,*
2. *the axioms of Fig. 3 (axioms (14) to (20)) are satisfied.*

Proof. (Prop. 10) See [13]. The most delicate case is that of Axiom (16). ◀

▶ Remark 11. Proposition 10 implies that if i is a probabilistic interpretation and if a statement $c \triangleleft \beta$ is derivable from the aGKAT axioms and possibly some semantic hypothesis of the shape $i \models A$, then $i \models c \triangleleft \beta$ holds. Note that Prop. 10 implies in particular that if i is a probabilistic interpretation and $A \Rightarrow B$ is an instance of an axiom in Fig. 3, then if $i \models A$ we can deduce that $i \models B$. This is because as i is a model, classical logic rules are sound in it.

▶ Remark 12. Note that by analogy with Axiom (18), one could have expected an axiom stronger than Axiom (16), namely that if $(c \cdot c_1 \triangleleft \beta \ \wedge \ c \cdot c_2 \triangleleft \beta)$ then one would have $c \cdot (c_1 +_b c_2) \triangleleft \beta$ (this would then generalize Axiom (18) when taking $c = 1$). However it turns out that this candidate additional axiom is *not* valid in the probabilistic model. A counter-example is given in [13].

▶ **Proposition 13.** *The following property is derivable in aGKAT:*

$$(c \cdot b_1 \triangleleft \beta_1 \ \wedge \ c \cdot b_2 \triangleleft \beta_2) \Rightarrow c \cdot (b_1 + b_2) \triangleleft \beta_1 + \beta_2$$

Proof. Observe that $b_1 + b_2 = b_1 +_{b_1} b_2$ by Prop. 5 and use Axiom (16). ◀

The proposition below refines in some sense Axiom (16).

▶ **Proposition 14.** *The following property is derivable in aGKAT:*

$$(c \cdot b \cdot c_1 \triangleleft \beta_1 \ \wedge \ c \cdot \bar{b} \cdot c_2 \triangleleft \beta_2) \Rightarrow c \cdot (c_1 +_b c_2) \triangleleft \beta_1 + \beta_2$$

Proof. Observe that $c_1 +_b c_2 = b \cdot c_1 +_b \bar{b} \cdot c_2$ by Axiom (4), Axiom (2), applied two times. Then apply Axiom (16) to c, $c'_1 = bc_1$ and $c'_2 = \bar{b}c_2$. ◀

▶ Remark 15 (Axiom (16), left distributivity and union bound). Recall that KAT [21] has an axiom of *left distributivity* $c \cdot (c_1 + c_2) = c \cdot c_1 + c \cdot c_2$. It does not hold in GKAT with the guarded sum $+_b$ though. In some sense axiom (16) (or its refinement Prop. 14) can be seen as a kind of compensation for this lack of left distributivity because it allows, when one is reasoning about an expression $c \cdot (c_1 +_b c_2)$ (in order to establish a bound β), to continue the proof with two branches, respectively on $c \cdot c_1$ and on $c \cdot c_2$. If one obtains two bounds $c \cdot c_i \triangleleft \beta_i$, for $i = 1, 2$ then one can deduce that $c \cdot (c_1 +_b c_2) \triangleleft \beta_1 + \beta_2$.

Moreover if c_1 and c_2 are tests b_1 and b_2, then by Prop. 5 $b_1 + b_2 = b_1 +_{b_1} b_2$. So $c \cdot (b_1 + b_2) = c \cdot (b_1 +_{b_1} b_2) \triangleleft \beta_1 + \beta_2$. So the probability that after execution of c the test $(b_1 + b_2)$ is satisfied is inferior to the sum of the probability that b_1 is satisfied and of the probability that b_2 is satisfied. This is the application of the binary union bound principle on post-conditions, and it can easily be applied to an arbitrary union bound.

4.2 Semantic reasoning

When reasoning about concrete programs, we want to establish properties on their semantic interpretations. That might sometimes require, besides the axioms of aGKAT, the use of some semantic properties. One such example is that some actions can be commuted without changing the semantics of the program. We establish thus some notations:

▶ **Definition 16.** *Given two GKAT program c and c' and a probabilistic interpretation i, we write $c \equiv c'$ if $i \models c = c'$, i.e. $\mathcal{P}_i[\![c]\!] = \mathcal{P}_i[\![c']\!]$.*

This definition is required to establish the following proposition.

▶ **Proposition 17.** *Consider GKAT programs c and c', and a probabilistic interpretation i.*

1. *If b is a test which only depends on the values of some variables $x_1, \ldots, x_n$ and if c leaves the values of those variables unchanged, then we have $c \cdot b \equiv b \cdot c$,*
2. *If $c \equiv c'$ and $i \models c \triangleleft \beta$, then $i \models c' \triangleleft \beta$.*

Observe that (1) holds because the syntax of programs does not allow any form of aliasing and (2) because the property $i \models c \triangleleft \beta$ only depends on the semantic interpretation $\mathcal{P}_i[\![c]\!]$.

Let us now illustrate the use of aGKAT on a small example.

▶ **Example 18** (Double tossing)**.** Consider the program c below:

$$c \quad = \quad (x \overset{\$}{\leftarrow} Coin) \cdot (c_1 +_{(x=1)} y \leftarrow 0), \qquad \text{where } c_1 = (x \overset{\$}{\leftarrow} Coin) \cdot (y \leftarrow 1 +_{(x=1)} y \leftarrow 0)$$

Consider the interpretation i where $Coin$ is the distribution of a fair coin, that takes value 0 (resp. 1) with probability 1/2 (resp. 1/2). This can be represented either by adding to the theory two axioms describing the behaviour of $Coin$, namely axioms $(x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \triangleleft 1/2$ and $(x \overset{\$}{\leftarrow} Coin) \cdot (x \neq 1) \triangleleft 1/2$, or by using the following semantic properties of i: $\models (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \triangleleft 1/2$ and $\models (x \overset{\$}{\leftarrow} Coin) \cdot (x \neq 1) \triangleleft 1/2$. We want to prove that after the execution of c, the probability that y equals 0 is below 3/4, and the probability that y equals 1 is below 1/4, i.e. $\models c \cdot (y = 0) \triangleleft 3/4$ and $\models c \cdot (y = 1) \triangleleft 1/4$.

Recall first that as i is a model, by Prop. 10, it satisfies all axioms of Fig. 1 and Fig. 3, and all classical logic rules are sound in it (see Remark 11). Now, by using Axiom (5), $c \cdot (y = 0)$ can be rewritten as follows:

$$\begin{aligned} c \cdot (y = 0) \quad &= \quad (x \overset{\$}{\leftarrow} Coin) \cdot (c'_1 +_{(x=1)} y \leftarrow 0(y = 0)) \\ c'_1 \quad &= \quad (x \overset{\$}{\leftarrow} Coin) \cdot (y \leftarrow 1(y = 0) +_{(x=1)} y \leftarrow 0(y = 0)) \end{aligned}$$

Let us name the following expressions, corresponding to the various possible branches of executions of $c \cdot (y = 0)$:

$$\begin{aligned} c_2 \quad &= \quad (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \cdot (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \cdot (y \leftarrow 1) \cdot (y = 0) \\ c_3 \quad &= \quad (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \cdot (x \overset{\$}{\leftarrow} Coin) \cdot (x \neq 1) \cdot (y \leftarrow 0) \cdot (y = 0) \\ c_4 \quad &= \quad (x \overset{\$}{\leftarrow} Coin) \cdot (x \neq 1) \cdot (y \leftarrow 0) \cdot (y = 0) \end{aligned}$$

First, from the model we know that $(y \leftarrow 1) \cdot (y = 0) \equiv 0$, so $c_2 \equiv 0$, so $\models c_2 \triangleleft 0$.

Then, as $\models (x \overset{\$}{\leftarrow} Coin) \cdot (x \neq 1) \triangleleft 1/2$, by Axioms (19) and (17) we have $\models c_4 \triangleleft 1/2$.

Then, as $\models (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \triangleleft 1/2$, by Axioms (17) and (19) we have $\models c_3 \triangleleft 1/4$.

$$\text{By applying Prop. 14 to } c_2 \text{ and } c_3 \text{ we get: } \quad \models (x \overset{\$}{\leftarrow} Coin) \cdot (x = 1) \cdot c'_1 \quad \triangleleft \quad 1/4 \qquad (21)$$

By applying again Prop. 14, this time to (21) and by $\models c_4 \triangleleft 1/2$ we finally obtain $\models c \cdot (y = 0) \triangleleft 3/4$ $(= 1/4 + 1/2)$. We give in [13] a step-by-step fully explicit version of the proof above. The proof that $\models c \cdot (y = 1) \triangleleft 1/4$ holds is similar.

5 Encoding aHL in aGKAT

We want to relate deduction in aHL and reasoning in aGKAT, by following the approach of [22] on the encoding of propositional Hoare logic in KAT. We consider the programming language of Example 1 but the results remain valid if we consider extended grammars of terms, distributions, tests and commands, where the class of tests is closed by substitution of terms t (as in Example 1). We will encode aHL derivations consisting of judgements $\vdash_\beta c : \phi \Rightarrow \phi'$ where ϕ and ϕ' belong to the class of tests.

Concretely the idea will be to encode the aHL judgement $\vdash_\beta c : \phi \Rightarrow \phi'$ by the aGKAT statement $\phi \cdot c \cdot \overline{\phi'} \lhd \beta$. Similarly to [22], showing that an aHL rule is sound in aGKAT will consist in proving that the conjunction of the aGKAT equations encoding the premises of the aHL rule implies the equation encoding the conclusion of the rule.

Observe that similarly as for Hoare logic, some rules of aHL, namely axiom rules (*Assn*) and (*Rand*), do not depend on aHL judgements as premises but rather on an interpretation of actions and predicates, and possibly a semantic condition (for (*Rand*)). Thus we do not expect to derive their encoding as an equation valid in the theory of aGKAT. Instead, one could add new axioms corresponding to (*Assn*) and (*Rand*) for specific distributions (as mentioned in Example 18), or alternatively when dealing with examples consider a particular interpretation i and thus reason on equalities of expressions in the model.

Fig. 4 lists the interpretations of the rules of aHL (Figure 2) in aGKAT, by encoding aHL judgments as aGKAT equations. Note that the rule (*Assn*) uses the test $\phi[t/x]$ obtained by substituting the term t in ϕ, which does belong to the class of tests by definition.

- Skip:
$$\phi 1 \bar{\phi} \lhd 0 \tag{22}$$
- Assn:
$$\phi[t/x](x \leftarrow t)\bar{\phi} \lhd 0 \tag{23}$$
- Rand:
$$(\forall m, m \models \phi \Rightarrow \mathcal{P}_i[\![x \xleftarrow{\$} d]\!](m)[\bar{\psi}] \leq \beta) \Rightarrow \phi(x \xleftarrow{\$} d)\bar{\psi} \lhd \beta \tag{24}$$
- Seq:
$$(\phi c \bar{\phi'} \lhd \beta) \wedge (\phi' c' \bar{\phi''} \lhd \beta') \Rightarrow \phi c c' \bar{\phi''} \lhd \beta + \beta' \tag{25}$$
- Cond:
$$(\phi b c \bar{\psi} \lhd \beta) \wedge (\phi \bar{b} c' \bar{\psi} \lhd \beta) \Rightarrow \phi(c +_b c')\bar{\psi} \lhd \beta \tag{28}$$
- Weak:
$$(\phi \leq \phi') \wedge (\phi c \bar{\psi} \lhd \beta) \wedge (\psi' \leq \psi) \wedge (\beta \leq \beta') \Rightarrow \phi' c \bar{\psi'} \lhd \beta' \tag{26}$$
- And:
$$(\phi c \bar{\psi} \lhd \beta) \wedge (\phi c \bar{\psi'} \lhd \beta') \Rightarrow \phi c \psi \bar{\psi'} \lhd \beta + \beta' \tag{29}$$
- Or:
$$(\phi c \bar{\psi} \lhd \beta) \wedge (\phi' c \bar{\psi} \lhd \beta) \Rightarrow (\phi + \phi') c \bar{\psi} \lhd \beta \tag{27}$$
- False:
$$\phi c \bar{\bot} \lhd 1 \tag{30}$$
- While:
$$(\models b_v \in \mathbb{N}) \wedge (\models (\phi \wedge (b_v \leq 0)) \to \bar{b}) \Rightarrow (\phi c \bar{\phi} \lhd \beta) \wedge (\forall_{\eta > 0} \cdot \phi b [b_v = \eta] c \overline{[b_v < \eta]} \lhd 0) \Rightarrow \phi[b_v \leq k] c^{(b)} \overline{\phi \bar{b}} \lhd k\beta \tag{31}$$

Figure 4 Interpretation of aHL rules in aGKAT.

The next theorem establishes the main result of the paper.

▶ **Theorem 19.** *All the rules of the system aHL, union bound logic, except* (Assn) *and* (Rand)*, have an aGKAT interpretation (in Fig. 4) that is derivable from the axioms of aGKAT.*

Note that the interpretation of the (**While**) rule, (31) in Fig. 4, is not a plain aGKAT formula, but has some semantic premises. This is because the aHL (**While**) rule itself is expressed with semantic premises. The proof of Theorem 19 can be found in [13]. The most interesting cases are (**Seq**) and (**And**) rules, and the most difficult one is that of (**While**).

Observe that an interesting feature of aGKAT is that none of its $\triangleleft$-axioms (Fig. 3) refers to guarded iteration $c^{(b)}$, and nevertheless aGKAT is as expressive as aHL and allows to derive its (**While**) rule. Another interesting specificity of aGKAT w.r.t. to aHL is that in aGKAT the axioms for reasoning on program equivalence (those of GKAT, Fig. 1) are disjoint from those for reasoning on probabilities (Fig. 3).

6 Example

We now consider the example of the *Report-noisy-max* algorithm, which has been analysed in [5] with the logic aHL. Our analysis here using aGKAT will be similar, but the equational approach of aGKAT will simplify some steps. The full proof can be found in [13].

We consider a finite set $\mathcal{R}$ and a quality score function *qscore*, which takes as input a pair of an element r of $\mathcal{R}$ and a database d, and returns a real number. The goal of the algorithm is to find an element r^* of $\mathcal{R}$ which approximately minimizes the function *qscore* on d. The algorithm is randomized and only computes an approximate minimization because it is designed to satisfy a differential privacy property (see [9]). The algorithm proceeds by computing for each element r of $\mathcal{R}$ the quality score $qscore(r, d)$ and adding to it a Laplacian noise (according to the Laplace mechanism for differential privacy [9]) and returning the element r^* with the highest noisy value.

Here we do not deal with the privacy property of this program, but instead our objective is to study its *accuracy*, that is to say to bound the difference between the value of $qscore(r^*, d)$ and the real minimum of $qscore(\cdot, d)$ on $\mathcal{R}$. The algorithm is encoded in GKAT as the program $c = (flag \leftarrow 1); (best \leftarrow 0); (\mathcal{R}_0 \leftarrow \mathcal{R}); (\mathcal{R}' \leftarrow \emptyset); c'^{[\mathcal{R} \neq \emptyset]}; return(r^*)$ where

$$c' = (r \leftarrow pick(\mathcal{R})); (noisy[r] \overset{\$}{\leftarrow} \mathcal{L}_{\epsilon/2}(qscore(r, d))); (c_1 +_b 1); (\mathcal{R} \leftarrow \mathcal{R} \backslash \{r\}); (\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\})$$
$$\text{where } c_1 = (flag \leftarrow 0); (r^* \leftarrow r); (best \leftarrow noisy[r]) \qquad b = (noisy[r] > best) + (flag == 1)$$

The variable $flag$ has Boolean values ($\{0, 1\}$), $\mathcal{R}$, $\mathcal{R}_0$ and $\mathcal{R}'$ are sets, r, r^* range over elements of $\mathcal{R}$, $noisy[r]$ and *best* range over reals. The variable *flag* is used for initialization purpose. The notation $noisy[r]$ is an array-like notation for representing n variables, where n is the size of the set $\mathcal{R}$. Note that variable $\mathcal{R}'$ does not play any role in the algorithm, it is just used to express properties of the execution.

This program uses the following kinds of actions and tests:

- actions for operations on sets: picking an (arbitrary) element r from a set $(r \leftarrow pick(\mathcal{R}))$, removing ($\mathcal{R} \leftarrow \mathcal{R} \backslash \{r\}$)) and adding an element $(\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\})$,
- sampling from a Laplacian distribution centered in a with parameter p : $(x \overset{\$}{\leftarrow} \mathcal{L}_p(a))$,
- tests: inequalities for reals, equality for Boolean value, comparison to empty set for sets $[\mathcal{R} \neq \emptyset]$; we will also need a finite number of additional tests for expressing properties on the execution, that we will see later.

We recall the following accuracy property of the Laplace distribution [5]:

▶ **Lemma 20.** *Let $\beta \in [0, 1]$, ν a sample from $\mathcal{L}_p(a)$. Then $Pr_{\mathcal{L}_p(a)}[|\nu - a| > \frac{1}{p} \log(\frac{1}{\beta})] < \beta$.*

Therefore we have $\models (x \xleftarrow{\$} \mathcal{L}_p(a))[|x-a| > \frac{1}{p}\log(\frac{1}{\beta})] \triangleleft \beta$. Hence for the sampling in c:

$$\models (noisy[r] \xleftarrow{\$} \mathcal{L}_{\epsilon/2}(qscore(r,d))) \cdot \overline{b_1} \triangleleft \frac{\beta}{|\mathcal{R}_0|} \tag{32}$$

where $\overline{b_1} = [\,|noisy[r] - qscore(r,d)| > \frac{2}{\epsilon}\log(\frac{|\mathcal{R}_0|}{\beta})]$.

Now we want to establish a property for the whole program c'. Observe that $\overline{b_1}$ only depends on the values of $noisy[r]$ and $qscore(r,d)$. Moreover $noisy[r]$ and $qscore(r,d)$ are not changed by the last 3 actions of c'. Therefore by applying Prop.17.1 we get $c'; \overline{b_1} \equiv c''$, where c'' is obtained by inserting in c' the test $\overline{b_1}$ just after $(noisy[r] \xleftarrow{\$} \mathcal{L}_{\epsilon/2}(qscore(r,d)))$. As we know by (19) that for any c_0 we have $\models c_0 \triangleleft 1$, by combining this with axiom (17) and (32) we get $\models c'' \triangleleft \frac{\beta}{|\mathcal{R}_0|}$. This is a step where aGKAT has provided us a concise and simple reasoning. Therefore, as $c'; \overline{b_1} \equiv c''$, we get by Prop. 17.2: $\models c' \cdot \overline{b_1} \triangleleft \frac{\beta}{|\mathcal{R}_0|}$.

We want to prove an invariant for the body c' of the *while* loop in c. For that consider the test b_2 corresponding to the predicate $\phi_2 = \forall r \in \mathcal{R}',\ |noisy[r] - qscore(r,d)| \leq \frac{2}{\epsilon}\log(\frac{|\mathcal{R}_0|}{\beta})$.

$$\text{We have:} \quad b_2 \cdot b_1 \cdot (\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\}) \cdot \overline{b_2} \quad \equiv \quad 0 \tag{33}$$

Let c_2 be c' deprived of the last action, i.e. $c' = c_2 \cdot (\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\})$. The reasoning we did on c' before can be repeated for c_2, and so as for c' we get: $\models c_2 \cdot \overline{b_1} \triangleleft \frac{\beta}{|\mathcal{R}_0|}$. Therefore by axioms (17) and (19) we get $\models b_2 \cdot c_2 \cdot \overline{b_1} \triangleleft \frac{\beta}{|\mathcal{R}_0|}$ (here again aGKAT helps us with conciseness). Moreover as c_2 does not modify $\mathcal{R}'$, by using Prop.17.1 we get $\models b_2 \cdot c_2 \cdot \overline{b_2} \triangleleft 0$. Thus by using the aGKAT encoding (Theorem 19) of the aHL rule (And) we obtain from the two previous statements: $\models b_2 \cdot c_2 \cdot \overline{b_1 \cdot b_2} \triangleleft \frac{\beta}{|\mathcal{R}_0|}$. Equation (33) gives us $\models (b_1 \cdot b_2) \cdot (\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\}) \cdot \overline{b_2} \triangleleft 0$.

By using the aGKAT encoding of the aHL rule (Seq) we get from the two last statements: $\models b_2 \cdot c' \cdot \overline{b_2} \triangleleft \frac{\beta}{|\mathcal{R}_0|}$. By using the aGKAT encoding of the aHL rule (While) we get from this last statement, since the loop runs for $|\mathcal{R}_0|$ iterations, $\models b_2 \cdot c'^{[\mathcal{R} \neq \emptyset]} \cdot \overline{b_2} \triangleleft \beta$.

Finally as $(\mathcal{R}' \leftarrow \emptyset) \cdot \overline{b_2} \equiv 0$ we deduce from this statement using (Seq) that $\models c \cdot \overline{b_2} \triangleleft \beta$. So we have proven using aGKAT that the property corresponding to the following judgement holds: $\vdash_\beta c : \top \Rightarrow \forall r \in \mathcal{R}',\ |noisy[r] - qscore(r,d)| \leq \frac{2}{\epsilon}\log(\frac{|\mathcal{R}_0|}{\beta})$.

In [13] we continue the proof to finally obtain an accuracy bound for the algorithm.

7 Related work

Several works have explored the use of program logics for the verification of probabilistic programs. Some of these works have explored approaches based on Hoare-like logics [16] while some other ones have developed the approach of weakest-pre-expectations, e.g. [25, 17]. The paper [7] has extended standard Hoare logic to deal with a language containing a probabilistic choice operator, and in which predicates express claims about the state of a probabilistic program. In this work, a semantics for the language is given and a Hoare-style deduction system presented, and proven to be correct w.r.t. the semantics. Another example is the union bound logic aHL [5] that we already presented. More recently, Graded Hoare logic (GHL) was introduced in [10] as a parameterisable framework for extending Hoare logic with a preordered monoidal analysis, with a few examples of applications: the union bound logic aHL [5]; logics for analysis of computation time; or the logic for reasoning about program counter security [26]. Other works even explore extensions of propositional dynamic logic to probabilistic programs, as [23]. This article proposes a probabilistic analog of PDL, which generalises the non-deterministic logical constructs and proof rules in PDL to arithmetic analogs in the probabilistic version.

Other approaches to probabilistic program verification were also introduced in the literature, relying on algebraic structures to wrap the apparatus of logical systems into more elegant frameworks. Some of these approaches are extension of Kleene algebra with tests, of which we give a few examples. A probabilistic extension of GKAT (ProbGKAT) was introduced in [29] for reasoning about imperative programs with probabilistic branching. One difference to our approach is the syntactic introduction of the probability, by the operator $\oplus_r$, where r is a probability; we rather do it semantically, by taking samplings as basic instructions, more in the style of the probabilistic assignment operator of the language `pIMP` [15] for instance. Additionally, [29] provides an axiomatisation of bisimilarity of ProbGKAT expressions and proves its completeness.

Another KAT extension was given in [11] which aimed at capturing fuzzy programs by replacing the Boolean algebra of KAT by a lattice, with the goal of being able to reason on fuzzy (non Boolean) properties [14].

A variant of GKAT was introduced in [12] as a relational structure to reason about properties of pairs of probabilistic programs (e.g. non-interference between variables), in the style of the system BiKAT [3] in the non-probabilistic setting. Still in the domain of relational reasoning, we can mention relational differential dynamic logic [20], which is specifically designed for the verification of *cyber-physical systems*, in a process that the authors called *synchronizing* the dynamics for comparing two systems.

8 Discussion and future work

We believe that a promising aspect of aGKAT and of the axiomatic presentation we introduced in this paper, is that they can contribute to extend the range of applicability of (co)-algebraic techniques of verification illustrated e.g. in [1, 24, 30, 2] to the realm of approximate reasoning on program effects [5, 6, 10]. This suggests several exciting research directions, which we discuss below.

Towards decision procedures. Recall that the paper [30] has given a decision procedure for the equivalence of GKAT programs whose complexity is almost linear time, assuming that the number of tests is fixed. This procedure is based on a new automata construction. One could investigate in an analogous way decision problems in aGKAT for statements of the form $c \triangleleft \beta$. The problem could be expressed with some semantic hypothesis, typically some probabilistic assumptions on the randomized primitives used by the program. Our axioms on the relation $\triangleleft$ (Fig. (3)) are quite promising in this respect since they are Horn clauses. Combining automata methods [30] and Horn clauses deduction techniques might lead to some efficient procedures. In [29], the authors present a decision procedure for demonstrating the existence of bisimilarity between two ProbGKAT expressions. Note that the probabilistic constructs of ProbGKAT can be encoded in aGKAT. Consider for $r \in [0, 1]$ the distribution $\mathtt{flip}(r)$ which returns 0 (resp. 1) with probability r (resp. $(1 - r)$). Then by using a fresh variable x, $c_0 \oplus_r c_1$ can be encoded as $(x \overset{\$}{\leftarrow} \mathtt{flip}(r)) \cdot (c_0 +_{[x=0]} c_1)$, and $c^{[r]}$ as $(x \overset{\$}{\leftarrow} \mathtt{flip}(r)) \cdot ((x \overset{\$}{\leftarrow} \mathtt{flip}(r)) \cdot c)^{[x=0]}$. Using this encoding and axiom (16) one obtains that from $c_i \triangleleft \beta_i$ for $i = 0, 1$ one can derive $c_0 \oplus_r c_1 \triangleleft r\beta_0 + (1 - r)\beta_1$.

Lower bounds. The system aGKAT has been defined to derive probabilistic upper bounds on successful termination, which allows to obtain upper bounds on the satisfaction or failure of a postcondition. It would be interesting to investigate if this approach can be adapted to derive probabilistic lower bounds, to prove that a postcondition holds with probability *at least* β.

Extension to other effects. In the present paper we restricted ourselves to a specific pod-monoid with specific interval of values and set of operators, which were enough to capture aHL and handle the initial intended goals of reasoning on probabilistic properties. However we would like to push this approach further. By using a generic and external structure to the main algebraic model of programs, we could follow a parametric approach, and obtain more freedom on the structure chosen to capture a wider range of quantitative analysis of effects, like for instance: the analysis of computation time model, by taking the natural numbers and the arithmetic sum as the monoidal composition; the program counter security model, by taking a set of binary values and the string concatenation as the composition; and the union bound logic itself. Those are a few concrete models considered for a generic version of Hoare Logic, analysed in [10].

We want to stress however that it is not trivial to capture in the same generic setting both the union bound logic and the logic for analysis of computation time. The system aGKAT as it stands does not allow to do that. In particular axiom (19) implies that the neutral element of the first monoid is also maximal for the order; this is not the case in the monoid $(\mathbb{N}, +)$ (or even $(\mathbb{N}^\infty, +)$) used for the Hoare logic for analysis of computation time [10].

The framework of pRHL could also benefit from an algebraic approach, calling for a structure taking into account the parametric reasoning about judgments themselves. One would need to embed the parameters into the structure itself, resorting, for example, to a relation between algebraic terms and the elements from the structure which model these parameters.

Towards a stronger completeness. Another possible direction for future work would be to study completeness of aGKAT with respect to some class of Horn clauses which embed aHL rules. That would mean to prove that the theory of aGKAT could always derive equations that represent valid aHL rules. We could draw inspiration from Kozen's classical work [22], in which an analogous result was proven for KAT with respect to a class of Horn clauses which embed propositional Hoare logic.

Towards relational properties. In this paper we have considered properties on single executions of a program, but some important questions can be expressed as relational properties on pairs of execution, for instance non-interference, continuity or sensitivity properties. An extension of Kleene algebra with tests called BiKAT for relational properties was introduced in [3] and another framework for probabilistic relational properties was proposed in [12]. It would interesting to explore if the approximation construction we defined in the present paper could be applied to the probabilistic relational setting of [12]. This would be analogous to the move in the relational Hoare logic setting, from pRHL to apRHL.

Non-determinism and probabilities. One of the advantages of the syntactical restriction of GKAT is to facilitate the inclusion of probabilistic models, by neglecting nondeterminism. While usually avoided, and always difficult, one possible direction for future work could be to consider a language with both nondeterminism and probabilities, capturing more application scenarios.

References

1 D. Kozen A. Angus. Kleene algebra with tests and program schematology. Technical report, Computer Science Department, Cornell University, Ithaca, NY 14853-7501, USA, July 2001. Technical Report TR2001-1844.

2 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 113–126. ACM, 2014. doi:10.1145/2535838.2535862.

3 Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. An algebra of alignment for relational verification. *Proc. ACM Program. Lang.*, 7(POPL):573–603, 2023. doi:10.1145/3571213.

4 Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier López, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013. doi:10.1007/978-3-319-10082-1_6.

5 Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. A program logic for union bounds. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 107:1–107:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.107.

6 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9:1–9:49, 2013. doi:10.1145/2492061.

7 Jerry den Hartog and Erik P. de Vink. Verifying probabilistic programs using a Hoare like logic. *Int. J. Found. Comput. Sci.*, 13(3):315–340, 2002. doi:10.1142/S012905410200114X.

8 Easycrypt development team. Easycrypt, 2024. URL: https://formosa-crypto.org/projects/.

9 Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

10 Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, and Tetsuya Sato. Graded hoare logic and its categorical semantics. In Nobuko Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 234–263. Springer, 2021. doi:10.1007/978-3-030-72019-3_9.

11 L. Gomes, A. Madeira, and L. S. Barbosa. Generalising KAT to verify weighted computations. *Scient. Annals of Comp. Sc.*, 29(2):141–184, 2019. doi:10.7561/SACS.2019.2.141.

12 Leandro Gomes, Patrick Baillot, and Marco Gaboardi. BiGKAT: an algebraic framework for relational verification of probabilistic programs. working paper or preprint, March 2023. URL: https://hal.science/hal-04017128.

13 Leandro Gomes, Patrick Baillot, and Marco Gaboardi. A Kleene algebra with tests for union bound reasoning about probabilistic programs. working paper or preprint, July 2024. URL: https://hal.science/hal-04196675v3.

14 Leandro Gomes, Alexandre Madeira, and Luís Soares Barbosa. A semantics and a logic for fuzzy arden syntax. *Soft Comput.*, 25(9):6789–6805, 2021. doi:10.1007/s00500-021-05593-9.

15 Ichiro Hasuo, Yuichiro Oyabu, Clovis Eberhart, Kohei Suenaga, Kenta Cho, and Shin-ya Katsumata. Control-data separation and logical condition propagation for efficient inference on probabilistic programs. *J. Log. Algebraic Methods Program.*, 136:100922, 2024. doi:10.1016/J.JLAMP.2023.100922.

16 Claire Jones. *Probabilistic non-determinism.* PhD thesis, University of Edinburgh, UK, 1990. URL: https://hdl.handle.net/1842/413.

17 Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65(5):30:1–30:68, 2018. doi:10.1145/3208102.

18 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent kleene algebra with observations: From hypotheses to completeness. In *Proceedings of FOSSACS 2020*, volume 12077 of *LNCS*, pages 381–400. Springer, 2020. doi:10.1007/978-3-030-45231-5_20.

19 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

20 Juraj Kolcák, Jérémy Dubut, Ichiro Hasuo, Shin-ya Katsumata, David Sprunger, and Akihisa Yamada. Relational differential dynamic logic. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2020. doi:10.1007/978-3-030-45190-5_11.

21 D. Kozen. Kleene algebra with tests. *ACM Trans. on Prog. Lang. and Systems*, 19(3):427–443, 1997. doi:10.1145/256167.256195.

22 D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. on Comp. Logic*, 1(212):1–14, 2000. doi:10.1109/LICS.1999.782610.

23 Dexter Kozen. A probabilistic PDL. *J. Comput. Syst. Sci.*, 30(2):162–178, 1985. doi:10.1016/0022-0000(85)90012-1.

24 Dexter Kozen and Maria-Christina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, volume 1861 of *Lecture Notes in Computer Science*, pages 568–582. Springer, 2000. doi:10.1007/3-540-44957-4_38.

25 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005. doi:10.1007/B138392.

26 David Molnar, Matt Piotrowski, David Schultz, and David A. Wagner. The program counter security model: Automatic detection and removal of control-flow side channel attacks. In Dongho Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, volume 3935 of *Lecture Notes in Computer Science*, pages 156–168. Springer, 2005. doi:10.1007/11734727_14.

27 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. doi:10.1017/cbo9780511814075.

28 Damien Pous. Kleene algebra with tests and coq tools for while programs. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 180–196. Springer, 2013. doi:10.1007/978-3-642-39634-2_15.

29 Wojciech Rozowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. Probabilistic guarded KAT modulo bisimilarity: Completeness and complexity. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 136:1–136:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.136.

30 Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proc. ACM Program. Lang.*, 4(POPL):61:1–61:28, 2020. doi:10.1145/3371129.

31 Jana Wagemaker, Nate Foster, Tobias Kappé, Dexter Kozen, Jurriaan Rot, and Alexandra Silva. Concurrent NetKAT - modeling and analyzing stateful, concurrent networks. In Ilya Sergey, editor, *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13240 of *Lecture Notes in Computer Science*, pages 575–602. Springer, 2022. doi:10.1007/978-3-030-99336-8_21.

32 Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. On incorrectness logic and Kleene algebra with top and tests. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022. doi:10.1145/3498690.

APPENDIX

A An example showing that aGKAT is more expressive than aHL

We have shown that aGKAT allows to encode aHL, but in this section we will show that aGKAT is more expressive than aHL, in the sense that in can prove some bounds that aHL cannot.

▶ **Example 21** (While program)**.** Let d be the distribution corresponding to a fair dice with three outcomes, that is to say that d has support $\{0,1,2\}$ and $d(0) = d(1) = d(2) = 1/3$.

We consider the program c below:

$$c \quad = \quad (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} \cdot [x = 1]$$

So c can be described as follows:

it samples d a first time and assigns the result to x; then until it obtains $(x \neq 0)$ it repeats sampling d and assigning the result to x; if at some point it obtains $(x \neq 0)$, then if $(x = 1)$ it terminates successfully, otherwise (that is to say if $(x = 2)$) it aborts.

The analysis of the probability of successful termination of c goes as follows:

with the first sample one obtains $(x = 1)$ with probability $1/3$ and then the program terminates successfully; or one obtains $(x = 2)$ with probability $1/3$ and then the program aborts; or one obtains $(x = 0)$ with probability $1/3$ and we execute c again.

So the probability of successful termination is:

$$\Sigma_{i=1}^{+\infty}(\frac{1}{3})^i = \frac{1}{3} \cdot \frac{1}{1-\frac{1}{3}} = \frac{1}{3} \cdot \frac{3}{2} = \frac{1}{2}$$

Let us now proceed with an analysis in aGKAT. We represent the properties of d with the following 4 axioms:

$$(x \xleftarrow{\$} d)[x = i] \triangleleft 1/3 \text{ for } i = 0,1,2 \ , (x \xleftarrow{\$} d)[x \neq 0,1,2] \triangleleft 0.$$

We can then derive the following proof by using GKAT axioms:

$$\begin{aligned}
& c \\
= \ & (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} \cdot [x = 1] \\
= \ & (x \xleftarrow{\$} d) \cdot ((x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} +_{[x=0]} 1) \cdot [x = 1] && \text{by ax. (11)} \\
= \ & (x \xleftarrow{\$} d) \cdot (1 +_{[x\neq 0]} (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])}) \cdot [x = 1] && \text{by ax. (2)} \\
= \ & (x \xleftarrow{\$} d) \cdot ([x \neq 0] +_{[x\neq 0]} [x = 0] \cdot (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])}) \cdot [x = 1] && \text{by ax. (4) and (2)} \\
= \ & (x \xleftarrow{\$} d) \cdot ([x \neq 0] \cdot [x = 1] +_{[x\neq 0]} [x = 0] \cdot (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} \cdot [x = 1]) && \text{by ax. (5)} \\
= \ & (x \xleftarrow{\$} d) \cdot ([x = 1] +_{[x\neq 0]} [x = 0] \cdot (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} \cdot [x = 1]) && \text{by properties of tests} \\
= \ & (x \xleftarrow{\$} d) \cdot ([x = 1] +_{[x\neq 0]} [x = 0] \cdot c)
\end{aligned}$$

We know that $(x \xleftarrow{\$} d) \cdot [x = 1] \triangleleft 1/3$ and $(x \xleftarrow{\$} d) \cdot [x = 0] \triangleleft 1/3$. By using axioms (19) and (17) we deduce that $(x \xleftarrow{\$} d) \cdot [x = 0] \cdot c \triangleleft 1/3$.

By applying axiom (16) to the last line of the derivation we obtain $c \triangleleft 1/3 + 1/3 = 2/3$.

We can then refine this bound by proceeding by recursion. Denote $\beta_0 = 1$ and $\beta_{i+1} = \frac{1}{3} \cdot (1 + \beta_i)$ for $i \geq 0$. That is to say that $\beta_i = \Sigma_{j=1}^{i} (\frac{1}{3})^j + (\frac{1}{3})^i$. Let us prove by recursion on i that one can derive $c \triangleleft \beta_i$ for any $i \geq 0$.

The property holds for $i = 0$. Let us assume it holds for i. By applying as before axiom (16) to the last line of the derivation and by using the recursion hypothesis we can derive $c \triangleleft 1/3 + 1/3\beta_i = \beta_{i+1}$. So by recursion we conclude that for any $i \geq 0$ we can derive $c \triangleleft \beta_i$.

As moreover the sequence (β_i) converges to $\Sigma_{i=1}^{+\infty} (\frac{1}{3})^i = \frac{1}{2}$ we can deduce meta-theoretically that $c \triangleleft \frac{1}{2}$ (although we cannot derive this limit bound within our system).

However we can verify that one cannot derive in aHL the property $c \triangleleft \beta_2$, that is to say $c \triangleleft \frac{5}{9}$. Indeed the aHL judgement corresponding to $c \triangleleft \frac{5}{9}$ is $\vdash_{5/9} (x \xleftarrow{\$} d) \cdot (x \xleftarrow{\$} d)^{([x=0])} : T \Rightarrow [x \neq 1]$. However in order to be able to apply a $(While)$ rule in aHL one needs to have an integer variable b_v that strictly decreases at each execution of the body of the loop, which is not the case here. So one cannot apply any $(While)$ rule, and thus one cannot prove this bound.

This example thus shows that aGKAT is more expressive than aHL, in the sense that it can prove probability bounds that aHL cannot.