BiGKAT: An Algebraic Framework forRelational Verification of ProbabilisticPrograms

Leandro Gomes[1](✉), Patrick Baillot[1], and Marco Gaboardi[2]

[1] Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{leandrogomes.moreiragomes,patrick.baillot}@univ-lille.fr
[2] Boston University, Boston, USA
gaboardi@bu.edu

Abstract. This work is devoted to formal reasoning on relational properties of probabilistic imperative programs. Relational properties are properties which relate the execution of two programs (possibly the same one) on two initial memories. We aim at extending the algebraic approach of Kleene Algebras with Tests (KAT) to relational properties of probabilistic programs. For that we consider the approach of Guarded Kleene Algebras with Tests (GKAT), which can be used for representing probabilistic programs, and define a relational version of it, called Bi-guarded Kleene Algebras with Tests (BiGKAT) together with a semantics. We show that the setting of BiGKAT is expressive enough to encode a finitary version of probabilistic Relational Hoare Logic (pRHL) (without the While rule), a program logic that has been introduced in the literature for the verification of relational properties of probabilistic programs. We also discuss the additional expressivity brought by BiGKAT.

Keywords: Kleene algebra with tests · Relational reasoning · Probabilistic programs · Hoare logic

1 Introduction

Formal verification of program properties has triggered a variety of methods, among which the algebraic approach of Kleene Algebra with Tests (KAT) stands out as an elegant, simple and automatizable framework [18,16]. It is closely related to modeling with finite automata and has stimulated the development of techniques from coalgebra for reasoning about program behavior, for instance based on bisimulation checking [12]. It has also been implemented in a library for the Coq proof-assistant [19]. Among the properties one might want to check on programs, some important ones are expressed by relating the execution of two programs on two initial states, or of the same program on two initial states. They are called *relational properties* or *2-properties*. One can think for instance of simulation properties, refinements, or extensional equivalence. Another example is non-interference: assume the variables are divided into public ones and private ones, a program satisfies non-interference if the final value of public variables after an execution only depends on the initial value of public variables.

Actually in a large number of situations the software systems one wants to verify are not deterministic but admit a probabilistic behaviour. Think for instance of randomized algorithms, cryptography, network programming or differential privacy. In those scenarios many crucial properties are also relational ones. For instance in cryptography one can express the fact that a randomized encryption scheme is safe as a probabilistic non-interference property: a public variable is assigned a ciphered value, obtained from a private variable, and we want to ensure that one cannot distinguish between two ciphered values computed from the same private initial state. Similarly in differential privacy (see e.g. [5]), in order to protect private data one might want to verify that two executions of a given program on two databases that differ only by one individual give indistinguishable result.

In order to express and prove relational properties on imperative programs some specific methods have been introduced. First in the deterministic case let us mention Relational Hoare Logic [10], that extends the classic Floyd-Hoare logic approach to reason on pairs of programs. This approach has been upgraded to the setting of probabilistic relational Hoare Logic (pRHL) by Barthe and coauthors [6]. It has then been extensively applied to the verification of cryptographic schemes, in particular through the development of the Certicrypt [9] and Easycrypt [3] tools.

However one would still benefit from additional techniques for the automation and the understandability of such reasoning methods. In particular one difficulty with (probabilistic) relational Hoare Logic is to find a suitable alignment of the two programs in order to be able, in a second step, to find the intermediate properties needed for the proof (see [1]). Algebraic methods coming from Kleene algebra with tests are promising in these respects. In particular they facilitate the reasoning on simple program transformations. Such direction is already addressed with the introduction of BiKAT [1], allowing to apply the KAT approach to reasoning on pairs of programs.

Our goal is thus to introduce a KAT approach to reason on relational properties of probabilistic programs. Unfortunately standard KAT techniques cannot be applied directly to probabilistic programs, since there is no known probabilistic interpretation for KAT. To handle this question, recent progress was made by the introduction of Guarded Kleene Algebra with tests (GKAT) [22], in which non-deterministic union and iteration are replaced by guarded union and iteration, while being sufficiently expressive to model imperative programming languages. The main motivation for the introduction of GKAT was initially to design a more efficient version of KAT where the complexity of the decision procedure is reduced, but it was also shown that GKAT admits a probabilistic model that can be used to interpret probabilistic programs.

Our strategy is thus to adapt the relational extension BiKAT to the setting of GKAT, in order to apply this relational approach to pairs of probabilistic programs. In practice we will consider programs of an imperative language extended with some probabilistic primitives. We call the corresponding calculus BiGKAT and in this paper define for it a syntax, a denotational semantics and a theory, which we prove to be sound. We would like to demonstrate the expressivity of our framework by showing how probabilistic relational Hoare Logic reasoning can be encoded in it, in an analogous way as (standard) Hoare logic can be encoded in KAT [17] and Relational Hoare Logic in BiKAT [1] (see 1). In the present work we make a first step in this direction, by showing that a finitary version of pRHL (without the While rule) can be encoded.

Property nature	Unary	Relational deterministic	Relational probabilistic
Hoare logic	HL	RHL	(finitary) pRHL
Algebra	KAT	BiKAT	**BiGKAT**
Examples of properties	Partial correctness	Validation of program transformations	Probabilistic non-interference

Table 1. Program logics and algebras

Outline. We first recall the definition of Guarded Kleene algebra with tests GKAT (Sect. 2), then introduce BiGKAT (Sect. 3) and describe its syntax, semantics, theory, as well as en encoding of a subset of pRHL. We present en example in Sect. 4 before discussing in Sect. 5 the differences between BiGKAT and pRHL and more generally related work in Sect. 6. We finish in Sect. 7 with conclusions and perspectives of future work.

Because of space constraints some proofs are omitted in this paper but can be found in the Appendix of the technical report [15].

2 Guarded Kleene algebra with tests

This section recalls the language and the semantics of *Guarded Kleene Algebra with Tests (GKAT)* [22], an abstraction of imperative programs where conditionals ($c_1 +_b c_2$) and loops ($c^{(b)}$) are expressions guarded by Boolean predicates b. As explained before, the structure is a restriction of KAT in which we are not allowed to freely use operators $+$ and * to build expressions, i.e. GKAT does not allow nondeterminism. Although less expressive than KAT, GKAT offers two advantages: decidability in (almost) linear time (compared to PSPACE complexity of decidability in KAT), and better foundation for probabilistic applications. Although the first one was the main motivation to introduce the structure [22], we are more interested in the second advantage for the purpose of this paper.

2.1 Syntax

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

$b, b_1, b_2 \in \mathtt{BExp} ::=$

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

$c, c_1, c_2 \in \mathtt{Exp} ::=$

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

where, for any $b, b_1, b_2 \in \mathtt{BExp}$, operators $\cdot$, $+$ and $\bar{\ }$ 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. Command **skip** will be 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 than $\cdot$.[3] To simplify the writing, we often omit the operator $\cdot$ by writing c_1c_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*[4] over S with discrete support, i.e. the set of functions $\mu : S \to [0, 1]$ such that $\mathit{Supp}(\mu) = \{x \in S \mid \mu(x) > 0\}$ is discrete and μ sums up to at most 1, i.e. $\sum_{s \in S} \mu(s) \leq 1$. In particular, the *Dirac* distribution $\delta_s \in \mathcal{D}(S)$ is the following map: $w \to \begin{cases} 1, \text{ if } w = s \\ 0, \text{ otherwise.} \end{cases}$

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:

$$\begin{aligned}
\textit{terms } t \in \mathtt{Terms} &::= x \in \mathtt{Var} \mid r \in \mathbb{R} \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \\
\textit{distributions } d &\in \mathtt{Distr} \\
\textit{tests } b \in \mathtt{Tests} &::= \textbf{false} \mid \textbf{true} \mid t_1 < t_2 \mid t_1 = t_2 \mid \textbf{not } b \mid b_1 \textbf{ and } b_2 \mid b_1 \textbf{ or } b_2 \\
\textit{commands } c \in \mathtt{Comm} &::= \textbf{skip} \mid x \leftarrow t \mid x \overset{\$}{\leftarrow} d \mid c_1; c_2 \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } b \textbf{ do } c
\end{aligned}$$

This language can be modeled in GKAT by taking as sets of actions and primitive tests respectively $\Sigma = \{x \leftarrow t,\ x \overset{\$}{\leftarrow} d \mid x \in \mathtt{Var}, t \in \mathtt{Terms}, d \in \mathtt{Distr}\}$ and $\mathtt{T} = \{t_1 < t_2,\ t_1 = t_2 \mid t_1, t_2 \in \mathtt{Terms}\}$[5] The first action evaluates term t and

[3] For example the GKAT expression $c_1^{(b_1)} \cdot c_2 +_{b_2} c_3$ reads as $((c_1^{(b_1)}) \cdot c_2) +_{b_2} c_3$.

[4] 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 $\frac{1}{2p} \exp(\frac{|x-a|}{p})$.

[5] Note that technically speaking according to the definition of GKAT the set $\mathtt{T}$ should be chosen finite, which is not the case here, but as observed in [22] Sect. 2.3 Example 2.5 we can use a finite subset $\mathtt{T}'$ of $\mathtt{T}$ for reasoning on pairwise equivalence of programs which terminate.

assigns the result to x; 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* [22] [6]. We first review some basic concepts needed for the semantics. Given a statement ϕ over a set S, the *Iverson bracket* $[\phi]$ is the function on S taking value 1 on $s \in S$ if $\phi(s)$ is true and 0 if it is false. 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 F, G... i.e. Markov kernels on sub-distributions.

Two particular examples of sub-Markov kernel on S are id_S and 0_S respectively defined by, for any s, $s' \in S$: $id_S(s) = \delta_s$ and $0_S(s)(s') = 0$.

The composition of two sub-Markov kernels F, G is $F; G$ defined by $(F; G)(s)(s') = \sum_{s'' \in S} F(s)(s'') \times G(s'')(s')$. This composition is associative, admits id_S as identity and 0_S as absorbing element: for any F, $F; id_S = id_S; F = F$ and $F; 0_S = 0_S; F = 0_S$.

Definition 1 (Probabilistic interpretation). *Let $i = (S, eval, sat)$ be a triple:*

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

We define $sat^\dagger : \mathtt{BExp} \to 2^S$ as the lifting of $sat : \mathtt{T} \to 2^S$ to arbitrary Boolean expressions over $\mathtt{BExp}$. The *probabilistic interpretation* of an expression c with respect to i is the sub-Markov kernel $\mathcal{P}_i[\![c]\!] : S \to \mathcal{D}(S)$ defined as follows:

$$\mathcal{P}_i[\![a]\!] := eval(a)$$

$$\mathcal{P}_i[\![b]\!](s) := \begin{cases} \delta_s, \text{ if } s \in sat^\dagger(b) \\ 0, \text{ otherwise} \end{cases} \qquad \mathcal{P}_i[\![c_1 +_b c_2]\!](s) := \begin{cases} \mathcal{P}_i[\![c_1]\!](s), \text{ if } s \in sat^\dagger(b) \\ \mathcal{P}_i[\![c_2]\!](s), \text{ if } s \in sat^\dagger(\bar{b}) \end{cases}$$

$$\mathcal{P}_i[\![c^{(b)}]\!](s)(s') := \lim_{n \to \infty} \mathcal{P}_i[\![(c +_b 1)^n \cdot \bar{b}]\!](s)(s')$$

$$\mathcal{P}_i[\![c_1 \cdot c_2]\!] := \mathcal{P}_i[\![c_1]\!]; \mathcal{P}_i[\![c_2]\!]$$

Intuitively $\mathcal{P}_i[\![c]\!](s)(s')$ is the probability that the execution of c on initial state s terminates on state s', and $\sum_{s' \in S} \mathcal{P}_i[\![c]\!](s)(s')$ is the probability that the execution of c on initial state s terminates (we then also say that it is a successful

[6] Note that more interpretations of GKAT are presented in [22], namely a relational model and a trace model.

execution). Observe thus that we really need to consider sub-distributions and not only distributions. Note that the definition implies that $\mathcal{P}_i[\![1]\!] = id_S$ and $\mathcal{P}_i[\![0]\!] = 0_S$. In the sequel, when the interpretation i is fixed we will sometimes write $[\![c]\!]$ instead of $\mathcal{P}_i[\![c]\!]$.

One can observe that:

$$\mathcal{P}_i[\![c \cdot b]\!](s_1)(s_2) = \begin{cases} \mathcal{P}_i[\![c]\!](s_1)(s_2), \text{ if } s_2 \in sat^\dagger(b) \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

$$\mathcal{P}_i[\![b \cdot c]\!](s_1)(s_2) = \begin{cases} \mathcal{P}_i[\![c]\!](s_1)(s_2), \text{ if } s_1 \in sat^\dagger(b) \\ 0, \text{ otherwise} \end{cases} \tag{2}$$

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

Example 2. Let us consider the example of the uniform distribution over the two-elements boolean set $Bool = \{tt, ff\}$ (or unbiased coin), that we call *dbool*: $dbool(tt) = dbool(ff) = 1/2$.

Then we have $eval(x \overset{\$}{\leftarrow} dbool)(m) = 1/2 \cdot \delta_{m[x \leftarrow tt]} + 1/2 \cdot \delta_{m[x \leftarrow ff]}$.

2.3 Axioms

The theory of GKAT introduced in [22] is given by the axioms from Fig. 1. Note

$$c +_b c = c \tag{3}$$
$$c_1 +_b c_2 = c_2 +_{\neg b} c_1 \tag{4}$$
$$(c_1 +_{b_1} c_2) +_{b_2} c_3 = c_1 +_{b_1 \cdot b_2} (c_2 +_{b_2} c_3) \tag{5}$$
$$c_1 +_b c_2 = b \cdot c_1 +_b c_2 \tag{6}$$
$$c_1 \cdot c_3 +_b c_2 \cdot c_3 = (c_1 +_b c_2) \cdot c_3 \tag{7}$$
$$(c_1 \cdot c_2) \cdot c_3 = c_1 \cdot (c_2 \cdot c_3) \tag{8}$$
$$0 \cdot c = 0 \tag{9}$$
$$c \cdot 0 = 0 \tag{10}$$
$$1 \cdot c = c \tag{11}$$
$$c \cdot 1 = c \tag{12}$$
$$c^{(b)} = c \cdot c^{(b)} +_b 1 \tag{13}$$
$$(c +_{b_2} 1)^{(b_1)} = (b_2 \cdot c)^{(b_1)} \tag{14}$$
$$\frac{c_3 = c_1 \cdot c_3 +_b c_2}{c_3 = c_1^{(b)} \cdot c_2} \text{ if } E(c_1) = 0 \tag{15}$$

Fig. 1. Axiomatisation of Guarded Kleene algebra with tests

in particular for the fixpoint axiom (15). 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 (see [22] for more details). In order to overcome such limitation, 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: $E(b) := b$, $E(a) := 0$, $E(c_1 +_b c_2) := b \cdot E(c_1) + \neg b \cdot E(c_2)$, $E(c_1 \cdot c_2) := E(c_1) \cdot E(c_2)$, $E(c^{(b)}) := \neg b$. 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 $c_1; (b; c_2 + \neg b; c_3) = c_1; b; c_2 + c_1; \neg b; c_3$ 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 $\neg b$ and executing c_3. That is why left distributivity does not hold in GKAT for any $c \in \text{Exp}$; it only holds for the particular case of $b \in \text{BExp}$, i.e. if b is a test.

In [15] we list additional derivable equations in GKAT, also given in [22].

We already mentioned that GKAT does not allow to construct an arbitrary program by using freely the nondeterministic choice operator $+$, allowing only guarded choice $+_b$, for any $b \in \text{BExp}$. However, the $+$ operator is included in the grammar of BExp, representing the Boolean disjunction. Since BExp$\subseteq$Exp, the grammar allows to write expressions as $b_1 +_b b_2$, for any $b \in \text{BExp}$.

By Boolean reasoning, we can observe that $b \cdot b + \neg b \cdot \neg b = 1$. Such property will be useful later to prove the soundness of R-Case rule (39).

3 Bi-guarded Kleene algebra with tests

We will define an algebra called BiGKAT, based on GKAT and which will allow us to reason on relations between two probabilistic programs, that we refer to as left and right programs. Just as GKAT it will be defined by a grammar of tests and a grammar of expressions. They can be thought of as tests over a product state space $S \times S$ and probabilistic programs over the same product state space. We will give them a semantics of sub-Markov kernels over $S \times S$.

3.1 Syntax

The syntax of BiGKAT is defined using that of GKAT and is additionnally parameterized by a set of actions $\mathbb{\Sigma}$ and a finite set of primitive tests $\mathbb{P}$ which are disjoint. Elements of $\mathbb{\Sigma}$ are denoted as A and those of $\mathbb{P}$ are denoted as P. A typical choice will be to consider in $\mathbb{P}$ some tests relating variables of the two programs on the two state spaces such as for instance $[x = x']$, and to consider in $\mathbb{\Sigma}$ some couplings between random assignments. The language of *Bi-guarded Kleene algebra with tests (BiGKAT)* consists of expressions in $\ddot{\text{E}}\text{xp}$ constructed from GKAT expressions c, c' in Exp, as follows:

$$B, B_1, B_2 \in \ddot{\text{B}}\text{Exp} ::= \ddot{0} \mid \ddot{1} \mid P \mid \langle b| \mid |b'\rangle \mid \neg B \mid B_1 \mathbin{⨾} B_2 \mid B_1 \oplus B_2$$

$$C, C_1, C_2 \in \ddot{\text{E}}\text{xp} ::= A \mid \langle c| \mid |c\rangle \mid \{c \underset{C}{\mid} c'\} \mid B \mid C_1 \mathbin{⨾} C_2 \mid C_1 \oplus_B C_2 \mid C^{(B)}$$

We will sometimes omit $\fatsemi$ and write $C_1 C_2$ for $C_1 \fatsemi C_2$, and $\overline{B}$ for $\neg B$.

We define the notation $\langle \_|\_ \rangle$ as $\langle c|c' \rangle \overset{\text{def}}{=} \langle c| \fatsemi |c' \rangle$.

Note that in the expression $\{c \underset{C}{|} c'\}$, c and c' belong to Exp (so GKAT) while the bottom C belongs to $\ddot{\mathrm{E}}\mathrm{xp}$ (BiGKAT). The intuition behind this expression is that C is a joint kernel over $S \times S$, whose left and right projections are respectively c and c', and that $\{c \underset{C}{|} c'\}$ denotes C itself. We will make this intuition more precise in the following section by defining the semantic interpretation by sub-Markov kernels.

3.2 Semantics of BiGKAT

As for interpreting GKAT we were using sub-Markov kernels over a given set S, now for BiGKAT for interpreting pairs of programs we will consider sub-Markov kernels over the product S^2. We will denote sub-Markov kernels on S by lower-case letters c, c', c_1 ...and those on S^2 by upper-case letters C, C', C_1, D .... In order to recover the interpretation of the left and right programs we will need the notion of projections or marginals. For that, given a sub-distribution μ on S^2 we denote its left and right marginals respectively as: $\Pi_1(\mu)(s) = \sum_{s' \in S} \mu(s, s')$, $\quad \Pi_2(\mu)(s') = \sum_{s \in S} \mu(s, s')$. Moreover if E is a subset of S^2, denote $\Pi_1(E) = \{s / \exists s' \in S, (s, s') \in E\}$ and $\Pi_2(E) = \{s' / \exists s \in S, (s, s') \in E\}$.

Example 3. Let us consider the set $Bool = \{tt, ff\}$ and the distributions on the product $Bool^2$ defined on Fig. 2 (the subscripts p, s, a are respectively for product, symmetric and antisymmetric). The array notation here means that the value of $\mu_p(x, x')$ is given by the coefficient on the line (resp. column) given by x (resp. x').

μ_p :

$x \backslash x'$	tt	ff
tt	1/4	1/4
ff	1/4	1/4

μ_s :

$x \backslash x'$	tt	ff
tt	1/2	0
ff	0	1/2

μ_a :

$x \backslash x'$	tt	ff
tt	0	1/2
ff	1/2	0

Fig. 2. Three distributions on $Bool^2$

By computing the left and right marginals, one obtains the following equalities, where $dbool$ is the uniform distribution on $Bool$ (see Example 2): $\Pi_1(\mu_p) = \Pi_1(\mu_s) = \Pi_1(\mu_a) = dbool, \qquad \Pi_2(\mu_p) = \Pi_2(\mu_s) = \Pi_2(\mu_a) = dbool.$

A simple way of constructing sub-Markov kernels on S^2 is by using products. The product of two sub-Markov kernels c and c' on S is defined as $c \times c' : (s_1, s_1') \to \big((s_2, s_2') \to c(s_1)(s_2) \times c'(s_1')(s_2')\big)$.

Let us show an example of sub-Markov kernel on S^2 in the case where S is a state of memories, as in Sect. 1. Assume μ is a distribution over a product space D^2 where D is a domain for a variable x (for instance $D = Bool$, as in Example 3). We define the sub-Markov kernel C_μ in a way similar to $eval(x \overset{\$}{\leftarrow} d)$ in Sect. 1 but over S^2: $C_\mu(m, m') = \sum_{(t,t') \in Supp(\mu)} \mu(t, t') \cdot \delta_{(m[x \leftarrow t], m'[x' \leftarrow t'])}$. In other words: $C_\mu(m,m')(m_1,m_1') = 0$ if there exists $y \neq x$ such that $m_1(y) \neq m(y)$ or $y' \neq x'$ such that $m_1'(y') \neq m'(y')$, and otherwise $C_\mu(m,m')(m_1,m_1') = \mu(t,t')$ where $t = m_1(x)$, $t' = m_1'(x')$. So Example 3 for instance allows to define the sub-Markov kernels C_{μ_p}, C_{μ_s}, C_{μ_a}.

Let us now consider the marginals of the sub-distributions $C_\mu(m,m')$:

$$\begin{aligned}\Pi_1(C_\mu(m,m'))(m_1) &= \sum_{m_1' \in S} \sum_{(t,t') \in Supp(\mu)} \mu(t,t') \cdot \delta_{(m[x \leftarrow t], m'[x' \leftarrow t'])}(m_1,m_1') \\ &= \begin{cases} 0 & \text{, if } \forall (t,t') \in Supp(\mu), m_1 \neq m[x \leftarrow t] \\ \sum_{t'/(t,t') \in Supp(\mu)} \mu(t,t') = \Pi_1(\mu)(t) & \text{, if } m_1 = m[x \leftarrow t] \end{cases}\end{aligned}$$

$$\begin{aligned}\Pi_1(C_\mu(m,m')) &= \sum_{t \in \Pi_1(Supp(\mu))} \Pi_1(\mu)(t) \cdot \delta_{m[x \leftarrow t]} \\ \Pi_2(C_\mu(m,m')) &= \sum_{t' \in \Pi_2(Supp(\mu))} \Pi_2(\mu)(t') \cdot \delta_{m'[x' \leftarrow t']}\end{aligned}$$

Example 4. In the case where μ is anyone of the three distributions μ_p, μ_s, μ_a of Example 3 one thus obtains, following the definition in Example 2:

$$\Pi_1(C_\mu(m,m')) = eval(x \overset{\$}{\leftarrow} dbool)(m), \;\; \Pi_2(C_\mu(m,m')) = eval(x' \overset{\$}{\leftarrow} dbool)(m')$$

Definition 2. *We say that two sub-Markov kernels C_1, C_2 on S^2 are equivalent, denoted by $C_1 \equiv C_2$, if $\forall_{(s,s') \in S^2}, i = 1, 2, \Pi_i(C_1(s,s')) = \Pi_i(C_2(s,s'))$.*

For instance we can deduce from Example 4 that: $C_{\mu_p} \equiv C_{\mu_s} \equiv C_{\mu_a}$.

Definition 3. *We say that a sub-Markov kernel C on S^2 is separable if there exists sub-Markov kernels $\mathcal{L}^C(.)$, $\mathcal{R}^C(.)$ on S such that, for all (s,s'): $\Pi_1(C(s,s')) = \mathcal{L}^C(s)$ and $\Pi_2(C(s,s')) = \mathcal{R}^C(s')$.*

Example 4 shows that C_{μ_p}, C_{μ_s} and C_{μ_a} are separable. Now, given a probabilistic interpretation $\mathcal{P}_i[\![.]\!]$ of GKAT we want to define a probabilistic interpretation $\overline{\mathcal{P}_i}[\![.]\!]$ of BiGKAT.

Definition 4 (Probabilistic interpretation of BiGKAT). *A probabilistic interpretation of BiGKAT is defined by a probabilistic interpretation $i = (State, eval, sat)$ of GKAT and two functions Eval and Sat such that:*

- *for each action $A \in \Sigma$, $Eval(A) : S^2 \to \mathcal{D}(S^2)$ is a sub-Markov kernel,*
- *for each primitive test $P \in \mathbb{P}$, $Sat(P) \subseteq S^2$ is a set of states.*

We want to define the interpretation $\overline{\mathcal{P}_i}[\![B]\!]$ and $\overline{\mathcal{P}_i}[\![C]\!]$ of expressions in $\ddot{\text{B}}$Exp and $\ddot{\text{E}}$xp. This interpretation will actually only be partially defined, that is to say in some cases $\overline{\mathcal{P}_i}[\![C]\!]$ is undefined.

For B in $\ddot{\text{B}}$Exp, $\overline{\mathcal{P}_i}[\![B]\!]$ is defined in the same way as $\mathcal{P}_i[\![b]\!]$ for b in BExp, using Sat.

For $A \in \Sigma$, $\overline{\mathcal{P}_i}[\![A]\!]$ is defined as $\overline{\mathcal{P}_i}[\![A]\!] = \text{Eval}(A)$. The interpretations of constructs $\overline{\mathcal{P}_i}[\![C_1 \mathbin{;} C_2]\!]$, $\overline{\mathcal{P}_i}[\![C_1 \oplus_B C_2]\!]$, $\overline{\mathcal{P}_i}[\![C^{(B)}]\!]$ are defined in the same way as the interpretations of the similar constructs of GKAT expressions with $\mathcal{P}_i[\![.]\!]$, except that the state is now S^2 instead of S.

There thus only remains to define $\overline{\mathcal{P}_i}[\![\langle c|]\!]$, $\overline{\mathcal{P}_i}[\![|c\rangle]\!]$ and $\overline{\mathcal{P}_i}[\![\{c \underset{C}{|} c'\}]\!]$.

$\overline{\mathcal{P}_i}[\![\langle c|]\!]$ is defined as $\overline{\mathcal{P}_i}[\![\langle c|]\!] = \mathcal{P}_i[\![c]\!] \times id_S$.
$\overline{\mathcal{P}_i}[\![|c\rangle]\!]$ is defined as $\overline{\mathcal{P}_i}[\![|c\rangle]\!] = id_S \times \mathcal{P}_i[\![c]\!]$.
$\overline{\mathcal{P}_i}[\![\{c \underset{C}{|} c'\}]\!]$ is defined only if $\overline{\mathcal{P}_i}[\![C]\!]$ is defined and if we have for all s, $s' \in S$:

$$\Pi_1(\overline{\mathcal{P}_i}[\![C]\!](s,s')) = \mathcal{P}_i[\![c]\!](s), \quad \Pi_2(\overline{\mathcal{P}_i}[\![C]\!](s,s')) = \mathcal{P}_i[\![c']\!](s'), \tag{16}$$

and in this case we define $\overline{\mathcal{P}_i}[\![\{c \underset{C}{|} c'\}]\!] = \overline{\mathcal{P}_i}[\![C]\!]$.

As a consequence we have:

Lemma 1. *For all c, c' in Exp we have $\overline{\mathcal{P}_i}[\![\langle c|c'\rangle]\!] = \mathcal{P}_i[\![c]\!] \times \mathcal{P}_i[\![c']\!]$.*

Now, remember the interpretation of composition by pre- and postconditions in GKAT, $\mathcal{P}_i[\![c \cdot b]\!]$ and $\mathcal{P}_i[\![b \cdot c]\!]$, given in (1) and (2). It also holds in the same way for BiGKAT, as the interpretations are the same as sub-Markov kernels over S^2. As a consequence we have, for the sub-distributions of Example 3:

Example 5. $C_{\mu_s} = C_{\mu_s}\ \overline{\mathcal{P}_i}[\![[x = x']]\!]$, $\quad C_{\mu_a} = C_{\mu_a}\ \overline{\mathcal{P}_i}[\![[x = \overline{x'}]]\!]$ (where $\overline{x'}$ is the negation of x).

3.3 Theory of BiGKAT

Now we give a list of axioms on BiGKAT expressions, using predicates $=$ and $\equiv$, that will be interpreted by equality and equivalence on sub-Markov kernels.

- The functions $\langle \_| : \text{Exp} \to \ddot{\text{Exp}}$, $|\_\rangle : \text{Exp} \to \ddot{\text{Exp}}$ satisfy, for any $b_1, b_2, b \in B$, c_1, c_2, $c \in C$, the following properties:

$$\langle 0| = |0\rangle = \ddot{0} \tag{17}$$
$$\langle 1| = |1\rangle = \ddot{1} \tag{18}$$
$$\langle b_1 + b_2| = \langle b_1| \oplus \langle b_2| \tag{19}$$
$$\langle \neg b| = \ddot{\neg}\langle b| \tag{20}$$
$$\langle c_1 \cdot c_2| = \langle c_1| \mathbin{;} \langle c_2| \tag{21}$$
$$\langle c_1 +_b c_2| = \langle c_1| \oplus_{\langle b|} \langle c_2| \tag{22}$$
$$\langle c^{(b)}| = \langle c|^{\langle b|} \tag{23}$$

  Similarly for $|\_\rangle$. We say that $\langle \_|$ and $|\_\rangle$ are *homomorphisms.* The operators have the same precedence as in GKAT.
- The following property on $\langle .|$ and $|.\rangle$:

$$\forall_{c_1,c_2 \in \text{Exp}},\ \langle c_1| \mathbin{;} |c_2\rangle = |c_2\rangle \mathbin{;} \langle c_1| \tag{24}$$

  The operators have the same precedence as in GKAT. For readability we use interchangeably the same notation for operators in GKAT and BiGKAT, i.e. operators $\cdot$, $\neg$ and $+_e$, for any $e \in B$, and constants 1 and 0 in GKAT

stand for $\,\mathring{,}$, $\neg$, $\oplus_{\langle e|}$ ($\oplus_{|e\rangle}$), $\ddot{1}$ and $\ddot{0}$, respectively. Often we go even further and omit the operator $\cdot$ and we write $\langle c_1|\langle c_2|$ ($|c_1\rangle|c_2\rangle$) for $\langle c_1|\cdot\langle c_2|$ ($|c_1\rangle\cdot|c_2\rangle$).
Note that property (24) states a commutativity property between programs that run in parallel, but we do not have in general $\langle c_1| \mathbin{;} \langle c_2| = \langle c_2| \mathbin{;} \langle c_1|$ for c_1, c_2 in $\ddot{\mathrm{E}}\mathrm{xp}$ (and similarly for operator $|\_\rangle$).

– The following properties on $\{.\mid.\}$:

$$\{c \underset{C}{|} c'\} = C \tag{25}$$

$$\{c_1 \underset{C_1}{|} c_1'\} \mathbin{;} \{c_2 \underset{C_2}{|} c_2'\} = \{c_1 \cdot c_2 \underset{C_1 ; C_2}{|} c_1' \cdot c_2'\} \tag{26}$$

– GKAT axioms for = (see Fig. 1) on GKAT expressions,
– GKAT axioms for = (see Fig. 1) written in the language of BiGKAT for BiGKAT expressions:
for instance equation (3) becomes $C \oplus_B C = C$ and (8) becomes $(C_1 \mathbin{;} C_2) \mathbin{;} C_3 = C_1 \mathbin{;} (C_2 \mathbin{;} C_3)$.
– Axioms for $\equiv$:

$$C_1 = C_2 \Rightarrow C_1 \equiv C_2 \tag{27}$$

$$C_1 \equiv C_2 \Rightarrow C_3 \mathbin{;} C_1 \equiv C_3 \mathbin{;} C_2 \tag{28}$$

$$C_1 \equiv C_2 \Rightarrow C_1 \mathbin{;} \{c \underset{C_3}{|} c'\} \equiv C_2 \mathbin{;} \{c \underset{C_3}{|} c'\} \tag{29}$$

We call this list of axioms the *theory of BiGKAT*. We say that a formula $C_1 = C_2$ (resp. $C_1 \equiv C_2$) is well-defined for an interpretation $\overline{\mathcal{P}_i}[\![.]\!]$ if $\overline{\mathcal{P}_i}[\![C_1]\!]$ and $\overline{\mathcal{P}_i}[\![C_2]\!]$ are both defined. An implicative formula $F_1 \Rightarrow F_2$ is well-defined for $\overline{\mathcal{P}_i}[\![.]\!]$ if both F_1 and F_2 are well-defined.

Now we say that an interpretation $\overline{\mathcal{P}_i}[\![.]\!]$ satisfies a formula F if F is well-defined for $\overline{\mathcal{P}_i}[\![.]\!]$ and if moreover:

1. if F is of the fom $C_1 = C_2$ (resp. $C_1 \equiv C_2$) then $\overline{\mathcal{P}_i}[\![C_1]\!] = \overline{\mathcal{P}_i}[\![C_2]\!]$ (resp. $\overline{\mathcal{P}_i}[\![C_1]\!] \equiv \overline{\mathcal{P}_i}[\![C_2]\!]$)
2. if F is of the form $C_1 = C_2 \Rightarrow C_3 \equiv C_4$ and if $\overline{\mathcal{P}_i}[\![C_1]\!] = \overline{\mathcal{P}_i}[\![C_2]\!]$, then $\overline{\mathcal{P}_i}[\![C_3]\!] \equiv \overline{\mathcal{P}_i}[\![C_4]\!]$ (and similarly for other implicative formulas built with = and $\equiv$).

An example of formula which can in some cases not be well-defined is equation (25), because for $\overline{\mathcal{P}_i}[\![\{c \underset{C}{|} c'\}]\!]$ to be defined its projections need to satisfy the equations (16).

Proposition 1. *The interpretation $\overline{\mathcal{P}_i}[\![.]\!]$ of BiGKAT expressions by sub-Markov kernels on the product space S^2 defined in Sect. 3.2 satisfies all well-defined axioms of the theory of BiGKAT.*

Proof. The properties (17) - (23) are obtained directly from the semantics of BiGKAT (Definition 4). The proofs of properties (24), (26), (27) - (29) are in [15].

For GKAT axioms (Fig. 1) formulated for BiGKAT, note that since the probabilistic interpretation of BiGKAT expressions $\overline{\mathcal{P}_i}[\![.]\!]$ is the same as the one of GKAT (on product space S^2), we naturally have that such interpretation satisfies the axioms of Fig. 1 written in the language of BiGKAT. □

We give in [15] some properties as consequences of the theory of BiGKAT.

We now derive proofs in BiGKAT in the following way. We assume given an interpretation $\overline{\mathcal{P}_i}[\![.]\!]$ and a finite subset of elements A of Σ together with some equations: $A = \{c \underset{A}{|} c'\}$ and $B \mathbin{;} A = B \mathbin{;} A \mathbin{;} B'$ which are well-defined and satisfied by $\overline{\mathcal{P}_i}[\![.]\!]$. Then we can use these equations and any formula of the theory of BiGKAT which is well-defined and satisfied by $\overline{\mathcal{P}_i}[\![.]\!]$ to derive new formulas. Proposition 1 then ensures that any formula derived in this way is satisfied by $\overline{\mathcal{P}_i}[\![.]\!]$.

Note that additionnally, if we allow ourselves to use in the proof any formula $C = C'$ which is satisfied by $\overline{\mathcal{P}_i}[\![.]\!]$ (semantic hypothesis), we still preserve the fact that the formulas derived are satisfied by $\overline{\mathcal{P}_i}[\![.]\!]$.

Example 6. Consider again the set space of memories. Consider two elements A_s, A_a of Σ with equations:

$$A_s = \{x \xleftarrow{\$} dbool \underset{A_s}{|} x' \xleftarrow{\$} dbool\} \qquad A_s = A_s \mathbin{;} [x = x'] \tag{30}$$

$$A_a = \{x \xleftarrow{\$} dbool \underset{A_a}{|} x' \xleftarrow{\$} dbool\} \qquad A_a = A_a \mathbin{;} [x = \overline{x'}] \tag{31}$$

Then if we choose $\overline{\mathcal{P}_i}[\![A_s]\!] = C_{\mu_s}$ and $\overline{\mathcal{P}_i}[\![A_a]\!] = C_{\mu_a}$, we know by Example 4 and Example 5 that $\overline{\mathcal{P}_i}[\![.]\!]$ satisfies the formulas above. Let us give a small example of proof:

$$\begin{aligned} A_s \oplus_{\langle y=tt|} (A_a \mathbin{;} \langle x \leftarrow \overline{x}|) &= (A_s \mathbin{;} [x = x']) \oplus_{\langle y=tt|} (A_a \mathbin{;} [x = \overline{x'}] \mathbin{;} \langle x \leftarrow \overline{x}|) (30), (31) \\ &= (A_s \mathbin{;} [x = x']) \oplus_{\langle y=tt|} (A_a \mathbin{;} \langle x \leftarrow \overline{x}| \mathbin{;} [x = x']) \\ &= (A_s \oplus_{\langle y=tt|} (A_a \mathbin{;} \langle x \leftarrow \overline{x}|)) \mathbin{;} [x = x'] \ (7) \text{ for BiGKAT} \end{aligned}$$

3.4 Encoding of a subsystem of pRHL into BiGKAT

We want to encode (a part of) probabilistic relational Hoare logic (pRHL) in BiGKAT. Recall that Hoare logic can be encoded in KAT [17] by encoding a Hoare triple $\{\phi\}\ c\ \{\psi\}$ as a KAT equation $\phi \cdot c = \phi \cdot c \cdot \psi$. The intuitive meaning of the equation is that testing ψ after executing $\phi \cdot c$ is always redundant. This approach has been extended to the relational setting with RHL and BiKAT in [1]. To define such an encoding for pRHL and BiGKAT we need first to recall the definition and semantics of pRHL judgements and proofs [6]. We consider the language of probabilistic programs of Example 1, a state space S of memories and a probabilistic interpretation $\mathcal{P}_i[\![.]\!]$.

Let us first define the *range* of a subdistribution μ on S^2: $range(\mu) = \{(m, m') \in S \ / \ \mu(m, m') > 0\}$.

A pRHL *judgement* is a tuple of the form $c \sim c' : \phi \Rightarrow \psi$ where c, c' are programs and ϕ, ψ are predicates on S^2 (relations on states). For simplicity we will also denote as ϕ the subset of elements in S^2 satisfying ϕ.

One says that the pRHL judgement is *valid in the interpretation* $\mathcal{P}_i[\![.]\!]$, denoted as $\models_i c \sim c' : \phi \Rightarrow \psi$ if:

for any $(m, m') \in \phi$, there exists a subdistribution μ on S^2 such that: $\Pi_1(\mu) = \mathcal{P}_i[\![c]\!](m)$, $\Pi_2(\mu) = \mathcal{P}_i[\![c']\!](m')$, and $range(\mu) \subseteq \psi$. One then says that programs c and c' are equivalent with respect to precondition ϕ and postcondition ψ. If the interpretation i is fixed we write $\models$ instead of $\models_i$.

Following the above interpretation, we encode the pRHL judgment in BiGKAT as follows:

$$\exists_{C \in \ddot{\mathrm{E}}\mathrm{xp}} \cdot \phi \mathbin{;} \{c \underset{C}{|} c'\} = \phi \mathbin{;} \{c \underset{C}{|} c'\} \mathbin{;} \psi \tag{32}$$

where $\phi, \psi \in \ddot{\mathrm{B}}\mathrm{Exp}$ and $c, c' \in \mathrm{Exp}$.

– Note that we do not use the encoding $\phi \mathbin{;} \{c \underset{C}{|} c'\} \leq \phi \mathbin{;} \{c \underset{C}{|} c'\} \mathbin{;} \psi$ since in GKAT and BiGKAT there is no natural notion of order $\leq$ as in KAT [18,16];
– We do not use either the encoding $\phi \mathbin{;} \{c \underset{C}{|} c'\} \mathbin{;} \neg\psi = 0$. In KAT, $\phi \cdot c = \phi \cdot c \cdot \psi$ is equivalent to $\phi \cdot c \cdot \neg\psi = 0$, but this cannot be proved in the same way in GKAT and the equivalence might not hold. We only have the implication $(\phi \cdot c = \phi \cdot c \cdot \psi) \Rightarrow (\phi \cdot c \cdot \neg\psi = 0)$, and we choose as encoding the stronger property. This encoding aligns with the intuitive interpretation of the validity of a Hoare triple, i.e. from a state satisfying the pre-condition ϕ, each execution of c, c', if it halts, it leads to a state satisfying the post-condition ψ.

Observe that the semantic interpretation of (32) is the same as $\models_i c \sim c' : \phi \Rightarrow \psi$.

We display on Fig. 3 the rules of pRHL defined in [6][7], except the rule for *While*, that we replace by an iteration rule (*R-Iter rule*) which we will explain below. This is a subsystem of pRHL, but we keep here the name pRHL for convenience.

We use different notation for pre and post conditions (ϕ, ψ) and for guards ($\langle b|$, $|b'\rangle$). Note in particular the side condition $\phi \Rightarrow b \doteqdot b'$ in rule *R-Cond*, where the right-hand side $b \doteqdot b'$ is equivalent to $\langle b|b'\rangle + \langle \neg b|\neg b'\rangle$, so the following holds

$$\phi\langle b|\neg b'\rangle = 0 \quad \phi\langle \neg b|b'\rangle = 0$$

These equalities assure that the predicates b and b' are evaluated to the same value on both left and right programs [1]. In particular, for the *R-Cond* rule it means that the same branch is executed for right-hand side and left-hand side programs. Observe that similarly as for Hoare logic, some rules of pRHL, namely axiom rules *R-Assign*, *R-Assign left* and *R-Rand assign* (see [15]) do not depend on pRHL judgements as premises but rather on an interpretation of actions and predicates, and a semantic condition (for *R-Rand assign*). Thus

[7] There are also one-sided versions of some of these rules, which we list in [15].

– *R-Assign rule*:

$$\frac{}{x \leftarrow v \sim x' \leftarrow v' : \phi[v/x, v'/x'] \Rightarrow \phi}$$

– *R-Seq rule*:

$$\frac{c_1 \sim c_1' : \phi \Rightarrow \psi \quad c_2 \sim c_2' : \psi \Rightarrow \xi}{c_1 \cdot c_2 \sim c_1' \cdot c_2' : \phi \Rightarrow \xi}$$

– *R-Rand assign rule*:

$$\frac{h \lhd (d, d') \quad \phi = \forall v \in Supp(d).\psi[v/x, h(v)/x']}{x \overset{\$}{\leftarrow} d \sim x' \overset{\$}{\leftarrow} d' : \phi \Rightarrow \psi}$$

– *R-Cond rule*:

$$\frac{\phi \Rightarrow b \doteq b' \quad c_1 \sim c_1' : \phi \wedge \langle b| \wedge |b'\rangle \Rightarrow \psi \quad c_2 \sim c_2' : \phi \wedge \langle \neg b| \wedge |\neg b'\rangle \Rightarrow \psi}{\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \ \sim \textbf{if } b' \textbf{ then } c_1' \textbf{ else } c_2' \ : \phi \Rightarrow \psi}$$

– *R-Iter rule*:

$$\frac{n \in \mathbb{N} \quad c \sim c' : \phi \Rightarrow \phi}{c^n \sim (c')^n : \phi \Rightarrow \phi}$$

– *R-Sub rule*:

$$\frac{\phi' \Rightarrow \phi \quad c \sim c' : \phi \Rightarrow \psi \quad \psi \Rightarrow \psi'}{c \sim c' : \phi' \Rightarrow \psi'}$$

– *R-Case rule*:

$$\frac{c \sim c' : \phi \wedge \phi' \Rightarrow \psi \quad c \sim c' : \phi \wedge \overline{\neg}\phi' \Rightarrow \psi}{c \sim c' : \phi \Rightarrow \psi}$$

Fig. 3. Probabilistic Relational Hoare Logic rules (pRHL)

we do not expect to derive their encoding as an equation valid in the theory of BiGKAT. Instead, when we deal with examples we will consider a particular interpretation and thus reason on equalities of expressions in the model. The first rule derives a valid Hoare triple with the substitution of variables x, x' by expressions v, v', respectively; the second one derives a valid triple with samplings over distributions d, d'. The *R-Iter* rules means that if the execution of program c does not change ϕ, then the composition of c n times does not change ϕ. In *R-Iter* on Fig. 3, $c^n = c \cdot c^{n-1}$, where $c^1 = c$.

Let us explain the R-Rand assign rule: $h \lhd (d, d')$ means that h is a *coupling* between distributions d, d', that is to say a bijective function from $Supp(d)$ to $Supp(d')$ such that: for every $v \in Supp(d)$, $P_{x\sim d}[x = v] = P_{x\sim d'}[x = h(v)]$. For instance μ_s and μ_a in Ex.6 respectively correspond to couplings $h = id$ and negation on *Bool*.

Now, to show that the rules of Fig. 3 are sound in BiGKAT, we interpret them as in Fig. 4[8], by using the encoding of pRHL judgements as BiGKAT equations defined previously.

Our goal is now to prove that the rules above are valid in BiGKAT. In this approach, showing that a pRHL rule is sound in BiGKAT will consist in proving

[8] Note that the encoding of the one-sided rules are listed in [15]

– *R-Assign rule*:

$$\phi[v/x, v'/x']\{x \leftarrow v \underset{C}{|} x' \leftarrow v'\} = \phi[v/x, v'/x'] \cdot \{x \leftarrow v \underset{C}{|} x' \leftarrow v'\} \cdot \phi \tag{33}$$

– *R-Rand assign rule*:

$$\begin{aligned} & h \lhd (d, d') \wedge \phi = \forall v \in Supp(d).\psi[v/x, h(v)/x'] \\ & \Rightarrow \phi \cdot \{x \xleftarrow{\$} d \underset{C}{|} x' \xleftarrow{\$} d'\} = \phi \cdot \{x \xleftarrow{\$} d \underset{C}{|} x' \xleftarrow{\$} d'\} \cdot \psi \end{aligned} \tag{34}$$

– *R-Seq rule*:

$$\begin{aligned} & \phi \cdot \{c_1 \underset{C_1}{|} c_1'\} = \phi \cdot \{c_1 \underset{C_1}{|} c_1'\} \cdot \psi \;\; \wedge \;\; \psi \cdot \{c_2 \underset{C_2}{|} c_2'\} = \psi \cdot \{c_2 \underset{C_2}{|} c_2'\} \cdot \xi \\ & \Rightarrow \;\; \phi \cdot \{c_1 \cdot c_2 \underset{C_1 C_2}{|} c_1' \cdot c_2'\} = \phi \cdot \{c_1 \cdot c_2 \underset{C_1 C_2}{|} c_1' \cdot c_2'\} \cdot \xi \end{aligned} \tag{35}$$

– *R-Cond rule*:

$$\begin{aligned} & \phi \,\overline{\leq}\, b \doteqdot b' \;\wedge\; \phi \cdot \langle b|b'\rangle \cdot \{c_1 \underset{C_1}{|} c_1'\} = \phi \cdot \langle b|b'\rangle \cdot \{c_1 \underset{C_1}{|} c_1'\} \cdot \psi \;\wedge \\ & \phi \cdot \; \langle \neg b|\neg b'\rangle \cdot \{c_2 \underset{C_2}{|} c_2'\} = \phi \cdot \; \langle \neg b|\neg b'\rangle \cdot \{c_2 \underset{C_2}{|} c_2'\} \cdot \psi \\ & \;\Rightarrow \; \phi \cdot \{c_1 +_b c_2 \underset{C_1 \oplus_{\langle b|b'\rangle} C_2}{|} c_1' +_{b'} c_2'\} = \phi \cdot \{c_1 +_b c_2 \underset{C_1 \oplus_{\langle b|b'\rangle} C_2}{|} c_1' +_{b'} c_2'\} \cdot \psi \end{aligned} \tag{36}$$

– *R-Iter rule*:

$$n \in \mathbb{N} \wedge \phi \cdot \{c \underset{C}{|} c'\} = \phi \cdot \{c \underset{C}{|} c'\} \cdot \phi \Rightarrow \phi \cdot \{c^n \underset{C^n}{|} (c')^n\} = \phi \cdot \{c^n \underset{C^n}{|} (c')^n\} \cdot \phi \tag{37}$$

– *R-Sub rule*:

$$\phi' \leq \phi \;\;\wedge\;\; \phi \cdot \{c \underset{C}{|} c'\} = \phi \cdot \{c \underset{C}{|} c'\} \cdot \psi \;\;\wedge\;\; \psi \leq \psi' \;\;\Rightarrow\;\; \phi' \cdot \{c \underset{C}{|} c'\} = \phi' \cdot \{c \underset{C}{|} c'\} \cdot \psi' \tag{38}$$

– *R-Case rule*:

$$\begin{aligned} & \phi \cdot \phi' \cdot \{c \underset{C}{|} c'\} = \phi \cdot \phi' \cdot \{c \underset{C}{|} c'\} \cdot \psi \;\; \wedge \phi \cdot \neg\phi' \cdot \{c \underset{C}{|} c'\} = \phi \cdot \neg\phi' \cdot \{c \underset{C}{|} c'\} \cdot \psi \\ & \Rightarrow \phi \cdot \{c \underset{C}{|} c'\} = \phi \cdot \{c \underset{C}{|} c'\}\psi \end{aligned} \tag{39}$$

Fig. 4. Encoding of pRHL rules in BiGKAT

that the conjunction of the BiGKAT equations encoding the premises of the pRHL rule implies the equation encoding the conclusion of the rule.

Finally we obtain the main result on the soundness of pRHL rules in BiGKAT.

Theorem 1 (Soundness of pRHL in BiGKAT). *The encoding of pRHL rules (Fig. 3) R-Seq, R-Cond, R-Sub, R-Case and R-Iter displayed in Fig. 4 can be derived by proofs in BiGKAT.*

4 Example

In this section we use the framework presented before to reason about invariance features of probabilistic programs.

Example 7. Consider a program c[9] encoded as the GKAT term

$$c = \big(b \xleftarrow{\$} dbool \cdot ((y \leftarrow y\ xor\ tt) +_{[b=tt]} 1) +_{[x=tt]} (b \leftarrow ff)\big) \cdot (y \leftarrow y\ xor\ b)$$

Consider also a second copy denoted as c'. We prove the invariance of variables y, y', relational predicate $[y = y']$, over executions of c, c', which corresponds to the following pRHL judgment $\vdash c \sim c' : [y = y'] \Rightarrow [y = y']$. In order to simplify the writing we denote $d_1 = b \xleftarrow{\$} dbool; ((y \leftarrow y\ xor\ tt) +_{[b=tt]} 1)$, $d_2 = b \leftarrow ff$ and $c_2 = (y \leftarrow y\ xor\ b)$, so that $c = (d_1 +_{[x=tt]} d_2) \cdot c_2$. We then use some equational reasoning to obtain in GKAT $(d_1 +_{[x=tt]} d_2) \cdot c_2 =_{(7)} (d_1 \cdot c_2) +_{[x=tt]} (d_2 \cdot c_2)$.

In order to define C the BiGKAT expression showing the analog of the pRHL judgement above, we will distinguish 4 subcases depending on the evaluation of $\langle [x = tt] | [x' = tt] \rangle$: (1)$x = tt, x' = tt$, (2)$x \neq tt, x' = tt$, (3)$x = tt, x' \neq tt$ and (4)$x \neq tt, x' \neq tt$.

For that we will define 4 expressions C_{ij} $(i = 0, 1,\ j = 0, 1)$ and C as:

$$C = (C_{11} \oplus_{|[x'=tt]\rangle} C_{10}) \oplus_{\langle [x=tt]|} (C_{01} \oplus_{|[x'=tt]\rangle} C_{00})$$

Assume temporarily that this is done, then we have:

$$\begin{aligned}
[y = y']C &= [y = y'](C_{11} \oplus_{|[x'=tt]\rangle} C_{10}) \oplus_{\langle [x=tt]|} (C_{01} \oplus_{|[x'=tt]\rangle} C_{00}) \\
&= [y = y'](\langle [x = tt] | [x' = tt] \rangle\ C_{11} \oplus_{|[x'=tt]\rangle} C_{10}) \oplus_{\langle [x=tt]|} (C_{01} \oplus_{|[x'=tt]\rangle} C_{00}) \\
&= ([y = y'] \langle [x = tt] | [x' = tt] \rangle\ C_{11} \oplus_{|[x'=tt]\rangle} [y = y']C_{10}) \\
&\quad \oplus_{\langle [x=tt]|} ([y = y']C_{01} \oplus_{|[x'=tt]\rangle} [y = y']C_{00})
\end{aligned}$$

where the last equality is obtained by one of the GKAT derivable equations listed in [15]. So if we can prove that

$$[y = y'] \langle [x = tt] | [x' = tt] \rangle\ C_{11} = [y = y'] \langle [x = tt] | [x' = tt] \rangle\ C_{11} [y = y'],$$

and similarly for the other C_{ij}, then we will be able to deduce that $[y = y']C = [y = y']C[y = y']$, by using repeatedly axiom (7) for BiGKAT.

We present here the proof of the property for C_{11} (subcase (1)), the most delicate one, and leave the proofs for the other C_{ij} to [15].

subcase (1):

Using assumptions $x = tt$, $x' = tt$ for the left and right programs we obtain $[x = tt]((d_1 \cdot c_2) +_{[x=tt]} (d_2 \cdot c_2)) = [x = tt](d_1 \cdot c_2)$ and $[x' = tt]((d_1' \cdot c_2') +_{[x'=tt]} (d_2' \cdot c_2')) = [x' = tt](d_1' \cdot c_2')$. As d_1 and d_1' contain a sampling we choose to use a coupling in order to obtain a postcondition. We use the constant A_s defined in Example 6 with its interpretation $A_s = \{b \xleftarrow{\$} dbool \underset{A_s}{|} b' \xleftarrow{\$} dbool\}$, $A_s = A_s \mathbin{⨾} [b = b']$.

[9] Its code written in the programming language of Example 1 is in [15].

Denote $e_1 = (y \leftarrow y\ xor\ tt) +_{[b=tt]} 1$, $e'_1 = (y' \leftarrow y'\ xor\ tt) +_{[b'=tt]} 1$ and let $C_{11} = A_s \fatsemi \langle e_1 \cdot c_2 | e'_1 \cdot c'_2 \rangle = A_s \fatsemi \langle e_1 | e'_1 \rangle \fatsemi \langle c_2 | c'_2 \rangle = A_s \fatsemi [b = b'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi \langle c_2 | c'_2 \rangle$. By using R-Cond rule (36) one can obtain: $[b = b'] \fatsemi \langle e_1 | e'_1 \rangle = [b = b'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi [b = b']$. We thus get $C_{11} = A_s \fatsemi [b = b'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi [b = b'] \fatsemi \langle c_2 | c'_2 \rangle$. Moreover we have by semantic hypothesis: $[y = y'] \fatsemi A_s = [y = y'] \fatsemi A_s \fatsemi [y = y']$ and $[y = y'] \fatsemi \langle e_1 | e'_1 \rangle = [y = y'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi [y = y']$, so we obtain $[y = y']C_{11} = [y = y'] \fatsemi A_s \fatsemi [b = b'] \fatsemi [y = y'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi [b = b'] \fatsemi [y = y'] \fatsemi \langle c_2 | c'_2 \rangle$. Now, recall that $c_2 = (y \leftarrow y\ xor\ b)$. We thus have $[b = b'] \fatsemi [y = y'] \fatsemi \langle c_2 | c'_2 \rangle = [b = b'] \fatsemi [y = y'] \fatsemi \langle c_2 | c'_2 \rangle \fatsemi [y = y']$. So from the two last equalities we deduce: $[y = y']C_{11} = [y = y'] \fatsemi A_s \fatsemi [b = b'] \fatsemi [y = y'] \fatsemi \langle e_1 | e'_1 \rangle \fatsemi [b = b'] \fatsemi [y = y'] \fatsemi \langle c_2 | c'_2 \rangle \fatsemi [y = y']$. So finally: $[y = y'] \fatsemi C_{11} = [y = y'] \fatsemi C_{11} \fatsemi [y = y']$. This was the property expected for C_{11}.

5 Discussion: comparison between BiGKAT and pRHL

We have seen that BiGKAT is at least as expressive as pRHL without *While*, since rules of the latter can be encoded in the former. Here we want to illustrate that BiGKAT is in some aspects more expressive than pRHL.

First, BiGKAT allows to derive some rules on pRHL judgements that are valid in the probabilistic model but that cannot be derived within the system pRHL, as defined in [6]. Consider the candidate rule below:

$$\frac{c \sim (c'_1 +_b c'_2)c'_3 : \phi \Rightarrow \psi}{c \sim c'_1 c'_3 +_b c'_2 c'_3 : \phi \Rightarrow \psi}$$

It can be derived in BiGKAT, however it cannot be derived as a sequence of pRHL rules (Fig. 3), simply because if we read any pRHL rule bottom-up the programs in the premises are subterms of the programs in the conclusion. More generally we can derive in BiGKAT:

$$\frac{c_1 \sim c'_1 : \phi \Rightarrow \psi \qquad c_1 = c_2 \qquad c'_1 = c'_2}{c_2 \sim c'_2 : \phi \Rightarrow \psi}$$

where premises $c_1 = c_2$ are given by any GKAT axioms. These rules extend in some sense pRHL with rewriting of programs according to GKAT axioms. This might be useful for some usages of pRHL (see for instance [2]).

A second point is that as BiGKAT, contrarily to pRHL, explicitly indicates for couplings the "witnesses" Markov kernels on S^2, it allows to express rules that cannot be written in pRHL. For instance, the following rule, displayed in pRHL format, is (trivially) derivable in BiGKAT:

$$\frac{\phi \fatsemi \{c \underset{C}{|} c'\} = \phi \fatsemi \{c \underset{C}{|} c'\} \fatsemi \psi_1 \qquad \phi \fatsemi \{c \underset{C}{|} c'\} = \phi \fatsemi \{c \underset{C}{|} c'\} \fatsemi \psi_2}{\phi \fatsemi \{c \underset{C}{|} c'\} = \phi \fatsemi \{c \underset{C}{|} c'\} \fatsemi (\psi_1 \wedge \psi_2)} \tag{40}$$

However the corresponding candidate pRHL rule is unsound:

$$\frac{c \sim c' : \phi \Rightarrow \psi_1 \qquad c \sim c' : \phi \Rightarrow \psi_2}{c \sim c' : \phi \Rightarrow \psi_1 \wedge \psi_2}$$

As a counter-example consider Example 6 and take $c = x \xleftarrow{\$} \mathit{dbool}$, $c' = x \xleftarrow{\$} \mathit{dbool}$, $\phi = 1$, $\psi_1 = [x = x']$, $\psi_2 = [x = \overline{x'}]$. Then $\psi_1 \wedge \psi_2$ is false, hence the conclusion is not valid. But the BiGKAT rule (40) could not be applied because the two witnesses A_s and A_a are not the same C. On this point we plan to study the possible links between BiGKAT and [7].

Finally, a third point is that BiGKAT allows to express judgements with assertions which cannot be expressed by a single pRHL judgement. Consider for example an equality $\phi \fatsemi \langle c_1|c_1'\rangle \fatsemi [x = x'] \fatsemi \langle c_2|c_2'\rangle = \phi \fatsemi \langle c_1|c_1'\rangle \fatsemi [x = x'] \fatsemi \langle c_2|c_2'\rangle \fatsemi \psi$. It says that, assuming precondition ϕ for the pair of programs c_1c_2 and $c_1'c_2'$, if moreover after execution of $\langle c_1|c_1'\rangle$ the assertion $[x = x']$ holds, then postcondition ψ is satisfied. This cannot be expressed by a single pRHL judgement.

6 Related work

A seminal approach to relational analysis of programs is due to [10], with the introduction of Relational Hoare logic (RHL). The system takes as the central ingredient the interpretation of program properties as relations over memories, allowing, for example, to prove correctness of program transformations for compiler optimisations. An algebraic approach to this system was taken in [1], by introducing BiKAT, a relational extension of KAT to model relational reasoning on programs, being able to express *all-exists* properties, i.e. 'for any run of one program there exists a run of the other such that ...'. It was shown in this paper that both the syntax and the deductive system of RHL, including *all-exists* corresponding versions [13,11], can be interpreted in BiKAT.

Probabilistic relational Hoare logic (pRHL), a probabilistic variant of RHL, was introduced by Barthe and coauthors in [6], motivated by the certification of cryptographic proofs. One goal of our approach is to subsume pRHL into algebraic reasoning, taking inspiration from BiKAT. Rather than using KAT, we build our approach on GKAT [22]. The main advantage of this structure for the purpose of this paper is the easier representation of probabilistic programming languages due to the absence of nondeterminism. The work of [22] also introduced the probabilistic model of GKAT based on sub-Markov kernels. The main model of the structure presented in this paper is naturally a relational version of such model. GKAT was also investigated further in [21], which in particular provides a semantics for which the equational theory is complete. The work [14] addressed the application of GKAT to unary (non-relational) properties of probabilistic programs, by investigating the relationships with the probabilistic Hoare logic aHL of [4].

In this work we have considered one probabilistic interpretation of GKAT to model the class of programs we wanted to address. By considering other more complex structures we would be able to increase the set of possible programs to analyse, an therefore capture a greater variety of examples. We could take, for instance, ProbGKAT [20], as our base structure, allowing to represents programs with probabilistic branching.

Another approach to relational reasoning of programs is approximate probabilistic relational Hoare logic (apRHL) [8], for the verification of differential privacy. The term ‘approximate’ refers here to the parameters associated to reasoning on judgments, which are related to the distance between the probabilistic distributions generated by the probabilistic programs.

7 Conclusion and perspectives

In this work we have introduced BiGKAT, a variant of KAT allowing to reason on relational properties of probabilistic programs, based on GKAT, provided a semantics for it based on sub-Markov kernels and a theory allowing to derive proofs. We have illustrated the expressivity of BiGKAT by proving how a subsystem of probabilistic relational Hoare logic [6] (with the *while* rule replaced by an iteration rule) can be soundly encoded in it.

In future work we want to extend this encoding to the full pRHL, including the *while* rule. Another interesting path, aligned with what we expressed in the previous section, would be to build a relational version of ProbGKAT, with the goal of extending the set of possible examples to programs with probabilistic branching. Moreover, while usually avoided, and always difficult, one possible direction for future work could be to consider a language with both nondeterminism and probabilities [23], capturing also more application scenarios. That could be indeed another direction to pursue in the future.

For the purpose of this work, we were focused on reasoning about properties of probabilistic non-interference over pairs of programs, i.e. *2-properties*. However, we believe that the generalization to a n-relational framework could be possible to handle properties such as n-safety.

Another natural path would be an algebraic approach to subsume approximate probabilistic relational Hoare logic (apRHL) [8], in order to provide an equational way of reasoning on differential privacy. This direction could use the approach of [14] which gives a way to define an approximate version of GKAT.

Acknowledgements Work partially funded by ANR Project HOPR (ANR-24-CE48-5521-01) from the French National Research Agency, by the project R-TALENT-21-001-BAILLOT from I-Site Université Lille Nord-Europe and 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.

References

1. 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.

2. Martin Avanzini, Gilles Barthe, Benjamin Grégoire, Georg Moser, and Gabriele Vanoni. Hopping proofs of expectation-based properties: Applications to skiplists and security proofs. *Proc. ACM Program. Lang.*, 8(OOPSLA1):784–809, 2024. doi:10.1145/3649839.
3. 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.
4. 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.
5. Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. Programming language techniques for differential privacy. *ACM SIGLOG News*, 3(1):34–53, 2016. doi:10.1145/2893582.2893591.
6. Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101. ACM, 2009. doi:10.1145/1480881.1480894.
7. Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Coupling proofs are probabilistic product programs. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 161–174. ACM, 2017. doi:10.1145/3009837.3009896.
8. 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.
9. Santiago Zanella Béguelin, Gilles Barthe, Benjamin Grégoire, and Federico Olmedo. Formally certifying the security of digital signature schemes. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 237–250. IEEE Computer Society, 2009. doi:10.1109/SP.2009.17.
10. Nick Benton. Simple relational correctness proofs for static analyses and program transformations. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 14–25. ACM, 2004. doi:10.1145/964001.964003.
11. Raven Beutner. Automated software verification of hyperliveness. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 196–216, Cham, 2024. Springer Nature Switzerland.
12. Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 457–468. ACM, 2013. doi:10.1145/2429069.2429124.

13. Robert Dickerson, Qianchuan Ye, Michael K. Zhang, and Benjamin Delaware. RHLE: Modular deductive verification of relational forall exists properties. In Ilya Sergey, editor, *Programming Languages and Systems - 20th Asian Symposium, APLAS 2022, Auckland, New Zealand, December 5, 2022, Proceedings*, volume 13658 of *Lecture Notes in Computer Science*, pages 67–87. Springer, 2022. doi:10.1007/978-3-031-21037-2_4.
14. Leandro Gomes, Patrick Baillot, and Marco Gaboardi. A Kleene algebra with tests for union bound reasoning about probabilistic programs. In *33rd EACSL Annual Conference on Computer Science Logic, CSL 2025*, volume 326. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. to appear. URL: https://hal.science/hal-04196675.
15. Leandro Gomes, Patrick Baillot, and Marco Gaboardi. BiGKAT: an algebraic framework for relational verification of probabilistic programs. Technical report, 2025. URL: https://hal.science/hal-04017128.
16. D. Kozen. Kleene algebra with tests. *ACM Trans. on Prog. Lang. and Systems*, 19(3):427–443, 1997. doi:10.1145/256167.256195.
17. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. on Comp. Logic*, 1(212):1–14, 2000. URL: http://dl.acm.org/citation.cfm?id=343378, doi:10.1109/LICS.1999.782610.
18. Dexter Kozen. Kleene algebra with tests and commutativity conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1996. doi:10.1007/3-540-61042-1_35.
19. 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.
20. 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.
21. Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Coequations, coinduction, and completeness. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 142:1–142:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.142.
22. 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.
23. Daniele Varacca and Glynn Winskel. Distributing probability over nondeterminism. *Math. Struct. Comput. Sci.*, 16(1):87–113, 2006. doi:10.1017/S0960129505005074.