Formalising Semantics for Expected Running Time of Probabilistic Programs (Rough Diamond)

Johannes Hölzl

Fakultät für Informatik, TU München, hoelzl@in.tum.de

Abstract. We formalise two semantics observing the expected running time of pGCL programs. The first semantics is a denotational semantics providing a direct computation of the running time, similar to the weakest pre-expectation transformer. The second semantics interprets a pGCL program in terms of a Markov decision process (MDPs), i.e. it provides an operational semantics. Finally we show the equivalence of both running time semantics.
We want to use this work to implement a program logic in Isabelle/HOL to verify the expected running time of pGCL programs. We base it on recent work by Kaminski, Katoen, Matheja, and Olmedo. We also formalise the expected running time for a simple symmetric random walk discovering a flaw in the original proof.

1 Introduction

We want to implement expected running time analysis in Isabelle/HOL based on Kaminski *et al.* [9]. They present semantics and proof rules to analyse the expected running time of probabilistic guarded command language (pGCL) programs. pGCL is an interesting programming language as it admits probabilistic and non-deterministic choice, as well as unbounded while loops [12].

Following [9], in Section 3 we formalise two running time semantics for pGCL and show their equivalence: a denotational one expressed as expectation transformer of type $(\sigma \Rightarrow \mathsf{ennreal}) \Rightarrow (\sigma \Rightarrow \mathsf{ennreal})$, and a operational one defining a Markov decision process (MDP). This proof follows the equivalence proof of pGCL semantics on the expectation of program variables in [4] derived from the pen-and-paper proof by Gretz *et al.* [3].

Based on these formalisations we analyse the simple symmetric random walk, and show that the expected running time is infinite. We started with the proof provided in [9], but we discovered a flaw in the proof of the lower ω-invariant based on the denotational semantics. Now, our solution combines results from the probability measure of the operational semantics and the fixed point solution from the denotational semantics.

Both proofs are based on our formalisation of Markov chains and MDPs [4]. The formalisation in this paper is on BitBucket[1].

[1] https://bitbucket.org/johannes2011/avgrun

2 Preliminaries

The formulas in this paper are oriented on Isabelle's syntax: type annotations are written $t :: \tau$, type variables can be annotated with type classes $t :: \tau :: tc$ (i.e. t has type τ which is in type class tc), and type constructors are written in post-fix notation: e.g. α set. We write int for integers, ennreal for extended non-negative real numbers: $[0, \infty]$, α stream for infinite streams of α, α pmf for probability mass functions (i.e. discrete distributions) on α. The state space is usually the type variable σ. On infinite streams sdrop n ω drops the first n elements from the stream ω: sdrop 0 $\omega = \omega$ and sdrop $(n+1)$ $(s{\cdot}\omega)$ = sdrop n ω.

Least fixed points A central tool to define semantics are least fixed points on complete lattices: $\alpha \Rightarrow (\beta :: \text{complete-lattice})$, bool, enat, and ennreal. Least fixed points are defined as $\mathsf{lfp}\ f = \bigsqcap\{u \mid f\,u \le u\}$. For a monotone function f, we get the equations $\mathsf{lfp}\,f = f\,(\mathsf{lfp}\,f)$. Fixed point theory also gives nice algebraic rules: the rolling rule "rolls" a composed fixed point: $g\,(\mathsf{lfp}\,(\lambda x.\ f\,(g\,x))) = \mathsf{lfp}\,(\lambda x.\ g\,(f\,x))$ for monotone f and g, and the diagonal rule for nested fixed points: $\mathsf{lfp}\,(\lambda x.\ \mathsf{lfp}\,(f\,x)) = \mathsf{lfp}\,(\lambda x.\ f\,x\,x)$, for f monotone in both arguments.

To use least fixed points in measure theory, countable approximations are necessary. This is possible if the function f is sup-continuous: $f\,(\bigsqcup_i C\,i) = \bigsqcup_i f\,(C\,i)$ for all chains C. Then f is monotone and $\mathsf{lfp}\,f = \bigsqcup_i f^i\,\bot$. For our proofs we also need an induction and a transfer rule[2]:

$$\frac{\mathsf{mono}\,f \qquad \forall x \le \mathsf{lfp}\,f.\ P\,x \longrightarrow P\,(f\,x) \qquad \forall S.\ (\forall x \in S.\ P\,x) \longrightarrow P\,(\bigsqcup S)}{P\,(\mathsf{lfp}\,f)}$$

$$\frac{\mathsf{sup\text{-}continuous}\ f, g, \text{and}\ \alpha \qquad \alpha\,\bot \le \mathsf{lfp}\,g \qquad \alpha \circ f = g \circ \alpha}{\alpha(\mathsf{lfp}\,f) = \mathsf{lfp}\,g}$$

Markov chains (MCs) and Markov decision processes (MDPs) An overview of Isabelle's MC and MDP theory is found in [4,5]. A MC is defined by a transition function $K :: \alpha \Rightarrow \alpha$ pmf, inducing an expectation: $\mathbb{E}^K_s[f]$ is the expectation of f over all traces in K starting in s. A MDP is defined by a transition function $K :: \alpha \Rightarrow \alpha$ pmf set, inducing the maximal expectation: $\hat{\mathbb{E}}^K_s[f]$ is the supremum of all expectation of f over all traces in K starting in s. Both expectations $\mathbb{E}^K_s[f]$ and $\hat{\mathbb{E}}^K_s[f]$ have values in ennreal, which is a complete lattice. Both are sup-continuous on measurable functions (called *monotone convergent* in measure theory), which allows us to apply the transfer rule when f is defined as a least fixed point. Also both expectations support an iteration rule, i.e. we can compute them by first taking a step in K and then continue in the resulting state t:

$$\mathbb{E}^K_s[f] = \int_t \mathbb{E}^K_t[\lambda\omega.\ f(t \cdot \omega)]\mathsf{d}K_s \quad \text{and} \quad \hat{\mathbb{E}}^K_s[f] = \bigsqcup_{D \in K_s} \int_t \hat{\mathbb{E}}^K_t[\lambda\omega.\ f(t \cdot \omega)]\mathsf{d}D.$$

Where $t \cdot \omega$ is the stream constructor and $\int f\mathsf{d}D$ is the integral over the pmf D.

[2] In our formalisation, the transfer rule is stronger: expectation requires measurability, hence we restrict the elements to which we apply α by some predicate P.

$$
\begin{aligned}
\sigma\ \mathsf{pgcl} = {} & \mathsf{Empty} \mid \mathsf{Skip} \mid \mathsf{Halt} \mid \mathsf{Assign}\ (\sigma \Rightarrow \sigma\ \mathsf{pmf}) \\
& \mid \mathsf{Seq}\ (\sigma\ \mathsf{pgcl})\ (\sigma\ \mathsf{pgcl}) \mid \mathsf{Par}\ (\sigma\ \mathsf{pgcl})\ (\sigma\ \mathsf{pgcl}) \\
& \mid \mathsf{If}\ (\sigma \Rightarrow \mathsf{bool})\ (\sigma\ \mathsf{pgcl})\ (\sigma\ \mathsf{pgcl}) \mid \mathsf{While}\ (\sigma \Rightarrow \mathsf{bool})\ (\sigma\ \mathsf{pgcl})
\end{aligned}
$$

Fig. 1. pGCL syntax

$$
\begin{aligned}
&\mathsf{ert} :: \sigma\ \mathsf{pgcl} \Rightarrow (\sigma \Rightarrow \mathsf{ennreal}) \Rightarrow (\sigma \Rightarrow \mathsf{ennreal}) \\
&\mathsf{ert}\ \mathsf{Empty}\ \ f = f \\
&\mathsf{ert}\ \mathsf{Skip}\ \ f = 1 + f \\
&\mathsf{ert}\ \mathsf{Halt}\ \ f = 0 \\
&\mathsf{ert}\ (\mathsf{Assign}\ u)\ \ f = 1 + \lambda x.\ \int_y f\ y\ d(u\ x) \\
&\mathsf{ert}\ (\mathsf{Seq}\ c_1\ c_2)\ f = \mathsf{ert}\ c_1\ (\mathsf{ert}\ c_2\ f) \\
&\mathsf{ert}\ (\mathsf{Par}\ c_1\ c_2)\ f = \mathsf{ert}\ c_1\ f \sqcup \mathsf{ert}\ c_2\ f \\
&\mathsf{ert}\ (\mathsf{If}\ g\ c_1\ c_2)\ f = 1 + \lambda x.\ \mathsf{if}\ g\ x\ \mathsf{then}\ \mathsf{ert}\ c_1\ f\ x\ \mathsf{else}\ \mathsf{ert}\ c_2\ f\ x \\
&\mathsf{ert}\ (\mathsf{While}\ g\ c)\ f = \mathsf{lfp}\ (\lambda W\ x.\ 1 + \mathsf{if}\ g\ x\ \mathsf{then}\ \mathsf{ert}\ c\ W\ x\ \mathsf{else}\ f\ x)
\end{aligned}
$$

Fig. 2. Expectation transformer semantics for pGCL running times

3 Probabilistic Guarded Command Language (pGCL)

The probabilistic guarded command language (pGCL) is a simple programming language allowing probabilistic assignment, non-deterministic choice and arbitrary While-loops. A thorough description of it using the weakest pre-expectation transformer (wp) semantics is found in McIver and Morgan [12]. Gretz *et al.* [3] shows the equivalence of wp with a operational semantics based on MDPs. Hurd [8] and Cock [2] provide a shallow embedding of pGCL in HOL4 and Isabelle/HOL. We follow the definition in Kaminski *et al.* [9].

In Figure 1 we define a datatype representing pGCL programs over an arbitrary program state of type σ. Empty has not running time. Halt immediately aborts the program. Seq is for sequential composition. Par is for non-deterministic choice, i.e. both commands are executed and then one of the results is chosen. Assign, If, and While have the expected behaviour, and all three commands require one time step. A probabilistic choice is possible with Assign u, where u is a probabilistic state transformer ($\sigma \Rightarrow \sigma$ pmf). The expected running time of Assign u weights each possible running time with the outcome of u. The assignment is deterministic is u is a Dirac distribution, i.e. assigning probability 1 to exactly one value. We need the datatype to have a deep embedding of pGCL programs, which is necessary for the construction of the MDP.

Expected Running Time The denotational semantics for the running time is given as an expectation transformer, which is similar to the denotational semantics for the expectation of program variables as weakest pre-expectation transformers. Again we follow the definition in Kaminski *et al.* [9]. In Figure 2 we define the expectation transformer ert taking a pGCL command c and an expectation f,

$$
\begin{aligned}
&\mathsf{K} :: (\sigma\ \mathsf{pgcl} \times \sigma) \Rightarrow (\sigma\ \mathsf{pgcl} \times \sigma)\ \mathsf{pmf\ set}\\
&\mathsf{K}(\mathsf{Empty},\ \ s) = \ll \mathsf{Empty}, s \gg\\
&\mathsf{K}(\mathsf{Skip},\ \ s) = \ll \mathsf{Empty}, s \gg\\
&\mathsf{K}(\mathsf{Halt},\ \ s) = \ll \mathsf{Halt}, s \gg\\
&\mathsf{K}(\mathsf{Assign}\ u,\ \ s) = \{[\lambda s'.\ (\mathsf{Empty}, s')]\ (u\ s)\}\\
&\mathsf{K}(\mathsf{Seq}\ c_1\ c_2,\ s) = \left[\lambda(c', s').\ \left(\left\{\begin{array}{ll} c_2 & \text{if } c' = \mathsf{Empty}\\ \mathsf{Halt} & \text{if } c' = \mathsf{Halt}\\ \mathsf{Seq}\ c'\ c_2 & \text{otherwise}\end{array}\right\}, s'\right)\right]\ \mathsf{K}\ (c_1, s)\\
&\mathsf{K}(\mathsf{Par}\ c_1\ c_2,\ s) = \ll c_1, s \gg \cup \ll c_2, s \gg\\
&\mathsf{K}(\mathsf{If}\ g\ c_1\ c_2,\ s) = \mathsf{if}\ g\ s\ \mathsf{then}\ \ \ll c_1, s \gg\ \ \mathsf{else}\ \ \ll c_2, s \gg\\
&\mathsf{K}(\mathsf{While}\ g\ c,\ s) = \mathsf{if}\ g\ s\ \mathsf{then}\ \ \ll \mathsf{Seq}\ c\ (\mathsf{While}\ g\ c), s \gg\ \ \mathsf{else}\ \ \ll \mathsf{Empty}, s \gg
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{cost} :: (\sigma \Rightarrow \mathsf{ennreal}) \Rightarrow \sigma\ \mathsf{pgcl} \Rightarrow \sigma \Rightarrow \mathsf{ennreal} \Rightarrow \mathsf{ennreal}\\
&\mathsf{cost}\ f\ \mathsf{Empty}\ \ \ s\ \ \_ = fs \qquad \mathsf{cost}\ \_\ (\mathsf{Seq\ Empty}\ \_)\ \_\ x = x\\
&\mathsf{cost}\ \_\ \mathsf{Halt}\ \ \ \_\ \ \_ = 0 \qquad \mathsf{cost}\ f\ (\mathsf{Seq}\ c\ \_)\ \ \ s\ \ x = \mathsf{cost}\ f\ c\ s\ x\\
&\mathsf{cost}\ \_\ (\mathsf{Par}\ \_\ \_)\ s\ \ x = x \qquad \mathsf{cost}\ \_\ \_\ \ \ \_\ x = 1 + x
\end{aligned}
$$

$\ll c, s \gg$ is the singleton set of the singleton distribution (c, s).
$[f]\mu$ maps f over all elements of μ

Fig. 3. MDP semantics for pGCL running times

where f assigns an expected running time to each terminal state of c. This gives a simple recursive definition of the Seq case, for the expected running time of a pGCL program we will set $f = 0$. We proved some validating theorems about expectation transformer ert, i.e. continuity and monotonicity of ert c, closed under constant addition for Halt-free programs, sub-additivitiy, etc.

MDP Semantics For the operational small-step semantics we introduce a MDP constructed per pGCL program, and compute the expected number of steps until the program terminates. In Figure 3 we define the MDP by its transition function K and the per-state cost function cost $f\ c\ s\ x$. The per-state cost cost $f\ c\ s\ x$ computes the running time cost associated with the program c at state s. Here the program is seen as a list of statements, hence we walk along a list of Seq and only look at its left-most leaf. If the program is Empty the MDP is stopped and we return $f\ s$ containing further running time cost we want to associated to a finished state s (in most cases this will be 0, but it is essential in the induction case of Theorem 1). When the execution continues we also add x, c.f. the definition of $\mathsf{cost}_{\mathsf{stream}}$.

The transition function K induces now a set of trace spaces, one for each possible resolution of the non-deterministic choices introduced by Par. We write $\hat{\mathbb{E}}^K_{(c,s)}[f]$ for the maximal expectation of $f :: (\sigma\ \mathsf{pgcl} \times \sigma)\ \mathsf{stream} \Rightarrow \mathsf{ennreal}$ when the MDP starts in (c, s). We define the cost of a trace as the sum of cost over all states in the trace:

$$\mathsf{cost}_{\mathsf{stream}}\ f\ ((c, s) \cdot \omega) \overset{\mathsf{lfp}}{=} \mathsf{cost}\ f\ c\ s\ (\mathsf{cost}_{\mathsf{stream}}\ f\ \omega)$$

Finally the maximal expectation of $\mathsf{cost}_{\mathsf{stream}}$ computes ert:

Theorem 1. $\hat{\mathbb{E}}^K_{(c,s)}[\mathsf{cost}_{\mathsf{stream}}\ f] = \mathsf{ert}\ c\ f\ s$

Proof (Induction on c). The interesting cases are Seq and While. For Seq we prove the equation $\hat{\mathbb{E}}^K_{(\mathsf{Seq}\ a\ b,s)}[\mathsf{cost}_{\mathsf{stream}}\ f] = \hat{\mathbb{E}}^K_{(a,s)}[\mathsf{cost}_{\mathsf{stream}}\ (\lambda s.\ \hat{\mathbb{E}}^K_{(b,s)}[\mathsf{cost}_{\mathsf{stream}}\ f])]$, by fixed point induction in both directions. For While we prove

$$\hat{\mathbb{E}}^K_{(\mathsf{While}\ g\ c,s)}[\mathsf{cost}_{\mathsf{stream}}\ f] = \mathsf{lfp}\ (\lambda F\ s.\ 1 + \mathsf{if}\ g\ s\ \mathsf{then}\ \hat{\mathbb{E}}^K_{(c,s)}[\mathsf{cost}_{\mathsf{stream}}\ f]\ \mathsf{else}\ f\ s)\ s$$

by equating it to a completely unrolled version using fixed point induction and then massaging it in the right form using the rolling and diagonal rules. □

4 Simple Symmetric Random Walk

As an application for the expected running time analysis Kaminski *et al.* [9] chose the simple random walk. As difference to [9] we do not use ω-invariants to prove the infinite running time, but the correspondence of the program with a Markov chain (there is no non-deterministic choice).

The simple symmetric random walk (srw) is a Markov chain on $\mathbb{Z}$, in each step i it goes uniformly to $i+1$ or $i-1$ (i.e. in both cases with probability $1/2$). Surprisingly, but well known (and formalised by Hurd [7]), it reaches each point with probability 1. Equally surprising, the expected time for the srw to go from i to $i+1$ is infinite! Kaminski *et al.* [9] prove this by providing a lower ω-invariant. Unfortunately, this proof has a flaw: in Appendix B.1 of [10] (the extended version of [9]), the equation $1+[\![x>0]\!]\cdot 2+[\![1<x\leq n+1]\!]\cdot\infty+[\![0<x\leq n-1]\!]\cdot\infty = 1+[\![x>0]\!]\cdot 2+[\![0<x\leq n+1]\!]\cdot\infty$ does not hold for $n=0$ and $x=1$. The author knows from private communication with Kaminski *et al.* that it still is possible to use a lower ω-invariant. Unfortunately, the necessary invariant gets much more complicated.

After discovering the flaw in the proof, we tried a more traditional proof. The usual approach in random walk theory uses the generating function of the first hitting time. Unfortunately, this would require quite some formalizations in combinatorics, e.g. Stirling numbers and more theorems about generating functions than available in [4]. Finally, we choose an approach similar to [7], i.e. we set up a linear equation system and prove that the only solution is infinity.

Now, $\mathsf{srw} :: \mathsf{int} \Rightarrow \mathsf{int}\ \mathsf{pmf}$ is the transition function for the simple symmetric random walk. The expected time to reach j when started in i is written $H\ i\ j \stackrel{\mathsf{def}}{=} \mathbb{E}^{\mathsf{srw}}_i[f\ j]$, where $f\ j\ (k\cdot\omega) \stackrel{\mathsf{lfp}}{=} \mathsf{if}\ j=k\ \mathsf{then}\ 0\ \mathsf{else}\ 1+f\ j\ \omega$ is the first hitting time. Now we need to prove the following rules: (I) $H\ j\ i = H\ j\ k + H\ k\ i$ if $i\leq j\leq k$, (II) $H\ (i+t)\ (j+t) = H\ i\ j$, (III) $H\ i\ j = H\ j\ i$ and (VI) $H\ i\ j = (\mathsf{if}\ i=j\ \mathsf{then}\ 0\ \mathsf{else}\ 1+(H\ i\ (j+1)+H\ i\ (j-1))/2)$. From these rules we can derive $H\ i\ j = \infty$ for $i\neq j$.

Rule (VI) is derived the expectation transformer semantics. But it is not clear to us how to prove rule (I) by only applying fixed point transformations or induction. Instead we prove (I) in a measure theoretic way:

$$
\begin{aligned}
H\ j\ k + H\ k\ i &= \mathbb{E}_j^{\mathsf{srw}}[f\ j + H\ k\ i] \\
&= \sum_n (n + H\ k\ i) \cdot \Pr_j(f\ k = n) && (1) \\
&= \sum_n \mathbb{E}_j^{\mathsf{srw}}[\lambda\omega.\ (n + f\ i\ (\mathsf{sdrop}\ n\ \omega)) \cdot [\![f\ k\ \omega = n]\!]] \\
&= \sum_n \mathbb{E}_j^{\mathsf{srw}}[f\ i] = H\ j\ i && (2)
\end{aligned}
$$

Equation 1 requires that $f\ k$ is finite with probability 1, we do a case distinction: if it is not finite a.e. the result follows from $H\ j\ i \geq H\ j\ k = \infty$. Equation 2 is now simply proved by induction on n. The proofs for Equations 1 and 2 essentially operate on each trace ω in our probability space, making them inherently dependent on the trace space.

Theorem 2 (The running time of srw is infinite.). $H\ i\ j = \infty$ *if* $i \neq j$.

5 Coupon Collector

Another example we formalised is the coupon collector example from [9]. The idea is to compute the expected time until we collect N different coupons from a uniform, independent and infinite source of coupons. The left side of Figure 4 shows our concrete implementation $\mathtt{CC}_N$, the right side is its refinement (there is no array cp necessary). By fixed point transformations we show that the (refined) inner loop's running time has a Geometric distribution, and hence the expected running time for $\mathtt{CC}_N$ is: $\mathsf{ert}\ \mathtt{CC}_N\ 0\ s = 2 + 4N + 2N \sum_{i=1}^{N} \frac{1}{i}$ for $N > 0$.

$$
\begin{array}{l}
x := 0, cp := [\overbrace{F, \ldots, F}^{N \text{ times}}], i := 0 \\
\texttt{WHILE}\ x < N\ \texttt{DO} \\
\quad \texttt{WHILE}\ cp[i]\ \texttt{DO}\ i :\sim \mathrm{Unif}\{0, \ldots, N\} \\
\quad cp[i] := T, x := x + 1
\end{array}
\qquad
\begin{array}{rcl}
x & \to & c \\
cp[i] & \to & b \\
|cp| & = & x
\end{array}
\qquad
\begin{array}{l}
c := 0, b := F \\
\texttt{WHILE}\ c < N\ \texttt{DO} \\
\quad \texttt{WHILE}\ b\ \texttt{DO}\ b :\sim \mathrm{Bern}(x/N) \\
\quad b := T, c := c + 1
\end{array}
$$

Fig. 4. The Coupon Collector in pGCL and its refinement

6 Related Work

The first formalisation of probabilistic programs was by Hurd [7] in `hol98`, formalising a trace space for a stream of probabilistic bits. Hurd *et al.* [8] is different approach, formalising the weakest pre-expectation transformer semantics of pGCL in HOL4. Both formalisations are not related. Audebaud and Paulin-Mohring [1] use a shallow embedding of a probability monad in Coq.

Cock [2] provides a VCG for pGCL in Isabelle/HOL. Hölzl and Nipkow [5, 6] formalises MCs and analyses the expected running time of the ZeroConf protocol. On the basis of [5] formalises MDPs and shows the equivalence of the weakest pre-expectation transformer (based on the pen-and-paper proof in [3]).

Unlike Theorem 1, these formalisations either define denotational semantics [1, 2, 8], or operational semantics [5–7], none of them relate both semantics.

7 Conclusion and Future Work

While formalising the random walk example in [9] we found an essential flaw in the proof in [10]. Our solution seams to indicate, that for the verification of expected running times an ω-invariant approach is not enough. While the expectation transformer gives us a nice verification condition generator (e.g. [2]), the trace space might be required to get additional information i.e. fairness and termination. The equivalence between the expectation transformer semantics and the MDP semantics provides the required bridge between both worlds. Also we might require a probabilistic, relational Hoare logic (maybe based on [11]) to automate tasks like Figure 4.

References

1. Audebaud, P., Paulin-Mohring, C.: Proofs of randomized algorithms in Coq. In: Special Issue on MPC 2006. Science of Computer Programming, vol. 74. 568–589
2. Cock, D.: Verifying probabilistic correctness in Isabelle with pGCL. In: SSV 2012. EPTCS, vol. 102, pp. 167–178 (2012)
3. Gretz, F., Katoen, J., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Performance Evaluation 73, 110–132 (2014)
4. Hölzl, J.: Markov chains and Markov decision processes in Isabelle/HOL. Submitted to JAR in December 2015 (`http://in.tum.de/~hoelzl/mdptheory`).
5. Hölzl, J.: Construction and Stochastic Applications of Measure Spaces in Higher-Order Logic. PhD thesis, Technische Universität München (2013)
6. Hölzl, J., Nipkow, T.: Interactive verification of Markov chains: Two distributed protocol case studies. In: QFM 2012. EPTCS, vol. 103 (2012)
7. Hurd, J.: Formal Verification of Probabilistic Algorithms. PhD thesis (2002)
8. Hurd, J., McIver, A., Morgan, C.: Probabilistic guarded commands mechanized in HOL. Theoretical Computer Science 346(1), 96–112 (2005)
9. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run-times of probabilistic programs. In: ESOP 2016. LNCS, vol. 9632, pp. 364–389 (2016)
10. Kaminski, B.L., Katoen, J., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected run-times of probabilistic programs. CoRR abs/1601.01001v1 (2016) (Extended version).
11. Lochbihler, A.: Probabilistic functions and cryptographic oracles in higher order logic. In: ESOP 2016. LNCS, vol. 9632, pp. 503–531. Springer (2016)
12. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer (2004)