Modular Runtime Complexity Analysis of Probabilistic While Programs

Martin Avanzini
INRIA Sophia Antipolis
France
martin.avanzini@inria.fr

Michael Schaper
University of Innsbruck
Austria
michael.schaper@student.uibk.ac.at

Georg Moser
University of Innsbruck
Austria
georg.moser@uibk.ac.at

Introduction We are concerned with the *average case runtime complexity analysis* of a prototypical *imperative language* PWHILE in the spirit of Dijkstra's *Guarded Command Language*. This language is endowed with primitives for *sampling* and *probabilistic choice* so that *randomized algorithms* can be expressed. Complexity analysis in this setting is particularly appealing as *efficiency* is one striking reason why randomized algorithms have been introduced and studied: in many cases, the most efficient algorithm for the problem at hand is randomized [5].

Our staring point towards an *automated analysis* is the ert-*calculus* of Kaminski et al. [4], which constitutes a *sound and complete method* for deriving expected runtimes of probabilistic programs. The ert-calculus has been recently automated within [6], showing encouraging results. Indeed, their prototype Absynth can derive accurate bounds on the expected runtime of a wealth of non-trivial, albeit academic, imperative programs with probabilistic choice. Since the average case runtime of probabilistic programs is inherently non-modular (see e.g. [4]), different program fragments cannot be analysed in general independently within the ert-calculus. This work aims at overcoming this situation, by enriching the calculus with a form of *expected value analysis*. Conceptually, our result rests on the observation that if f and g measure the runtime of deterministic programs C and D as a function in the variables assignment σ before executing the command, then $f(\sigma) + g(\sigma')$ for σ' the store after the execution of C gives the runtime of the composed command C;D. Estimating σ' in terms of C and σ, and ensuring some monotonicity property on g, gives rise to a compositional analysis. When C exhibits some probabilistic behavior though, the command D may be executed after C on several probabilistic branches b, each with probability p_b with a variable assignment σ_b. Assuming bounding functions f and g on the expected runtime of C and D respectively, yields a bound $f(\sigma) + \sum_b p_b \cdot g(\sigma_b)$ on the expected runtime of the probabilistic program C;D. As the number of probabilistic branches b is unbounded for all but the most trivial programs C, estimating all assignments σ_b in terms of σ soon becomes infeasible. The crux of our approach towards a compositional analysis lies in the observation that if we can give the runtime of D in terms of a concave function (i.e., described by a multi-linear polynomial), the expected runtime $\sum_b p_b \cdot g(\sigma_b)$ can be bounded in terms of g and the variable assignment $\sum_b p_b \cdot \sigma_b$ expected after executing C. This way, a compositional analysis is recovered. This observation then also enables some form of modularity for the analysis of nested loops.

To prove this machinery sound, we first give a novel structural operational semantics in terms of *weighted probabilistic ARSs (Abstract Reduction Systems)*. These constitute a refinement to probabilistic ARSs introduced by Bournez and Garnier [2] where operations do not necessarily have uniform cost. Probabilistic ARSs give rise to a reduction relation on (multi-)distributions that is equivalent to the standard operational semantic via stochastic processes [1]. We then *generalise the* ert-*calculus* to one for reasoning about *expected costs* consumed by a command tick($\cdot$), and *expected values in final configurations*. This machinery is proven sound and complete with respect to our new operational semantics. Finally, we conclude with some words on a *prototype implementation* that we are currently developing.

Submitted to:

$$\frac{}{\langle \texttt{tick}(r)\rangle(\sigma) \xrightarrow{r} \sigma}\ [\textsc{Tick}] \qquad \frac{}{\langle x := d\rangle(\sigma) \xrightarrow{0} \{\!\{p_i : \sigma[x := i] \mid i \in \mathbb{Z}, p_i = [\![d]\!](\sigma)(i) > 0\}\!\}}\ [\textsc{Assign}]$$

$$\frac{\sigma \vDash \psi \wedge \phi}{\langle \texttt{if [}\psi\texttt{] (}\phi\texttt{) \{C\} \{D\}}\rangle(\sigma) \xrightarrow{0} \langle \texttt{C}\rangle(\sigma)}\ [\textsc{IfT}] \qquad \frac{\sigma \vDash \psi \wedge \neg\phi}{\langle \texttt{if [}\psi\texttt{] (}\phi\texttt{) \{C\} \{D\}}\rangle(\sigma) \xrightarrow{0} \langle \texttt{D}\rangle(\sigma)}\ [\textsc{IfF}]$$

$$\frac{\sigma \vDash \psi \wedge \phi}{\langle \texttt{while [}\psi\texttt{] (}\phi\texttt{) \{C\}}\rangle(\sigma) \xrightarrow{0} \langle \texttt{C;while [}\psi\texttt{] (}\phi\texttt{) \{C\}}\rangle(\sigma)}\ [\textsc{WhT}] \qquad \frac{\sigma \vDash \psi \wedge \neg\phi}{\langle \texttt{while [}\psi\texttt{] (}\phi\texttt{) \{C\}}\rangle(\sigma) \xrightarrow{0} \sigma}\ [\textsc{WhF}]$$

$$\frac{\sigma \vDash \neg\psi}{\langle \texttt{if [}\psi\texttt{] (}\phi\texttt{) \{C\} \{D\}}\rangle(\sigma) \xrightarrow{0} \bot}\ [\textsc{AbortIf}] \qquad \frac{\sigma \vDash \neg\psi}{\langle \texttt{while [}\psi\texttt{] (}\phi\texttt{) \{C\}}\rangle(\sigma) \xrightarrow{0} \bot}\ [\textsc{AbortWhile}]$$

$$\frac{}{\langle \texttt{\{C\} <> \{D\}}\rangle(\sigma) \xrightarrow{0} \langle \texttt{C}\rangle(\sigma)}\ [\textsc{ChoiceL}] \qquad \frac{}{\langle \texttt{\{C\} <> \{D\}}\rangle(\sigma) \xrightarrow{0} \langle \texttt{D}\rangle(\sigma)}\ [\textsc{ChoiceR}]$$

$$\frac{}{\langle \texttt{\{C\}[}p\texttt{]\{D\}}\rangle(\sigma) \xrightarrow{0} \{\!\{p : \langle \texttt{C}\rangle(\sigma), 1-p : \langle \texttt{D}\rangle(\sigma)\}\!\}}\ [\textsc{ProbChoice}]$$

$$\frac{\langle \texttt{C}\rangle(\sigma) \xrightarrow{r} \{\!\{p_i : \gamma_i\}\!\}_{i\in I}}{\langle \texttt{C;D}\rangle(\sigma) \xrightarrow{r} \{\!\{p_i : \mathsf{step}_{\texttt{D}}(\gamma_i)\}\!\}_{i\in I}}\ [\textsc{Compose}] \quad \text{where } \mathsf{step}_{\texttt{D}}(\gamma) \triangleq \begin{cases} \langle \texttt{C;D}\rangle(\sigma) & \text{if } \gamma = \langle \texttt{C}\rangle(\sigma) \\ \langle \texttt{D}\rangle(\sigma) & \text{if } \gamma = \sigma \in \Sigma \\ \bot & \text{if } \gamma = \bot. \end{cases}$$

Figure 1: One-step reduction relation as a weighted probabilistic ARS.

A Probabilistic While Language For a finite set of integer-valued *variables* Var, we denote by $\Sigma \triangleq \mathsf{Var} \to \mathbb{Z}$ the set of *stores*. The syntax of *program commands* Cmd over Var is given as follows:

$$\texttt{C},\texttt{D} ::= \texttt{tick}(r) \mid x := d \mid \texttt{if [}\psi\texttt{] (}\phi\texttt{) \{C\} \{D\}} \mid \texttt{while [}\psi\texttt{] (}\phi\texttt{) \{C\}} \mid \texttt{\{C\} <> \{D\}} \mid \texttt{\{C\}[}p\texttt{]\{D\}} \mid \texttt{C;D}\,.$$

In this grammar, $\phi \in \mathsf{BExp}$ denotes a *Boolean expression* over Var and $d \in \mathsf{DExp}$ an *Integer-valued distribution expression* over Var. With $[\![\cdot]\!] : \mathsf{DExp} \to \Sigma \to \mathscr{D}(\mathbb{Z})$ we denote the evaluation functions of distribution expressions, i.e., $[\![d]\!](\sigma)$ gives the result of evaluating d under the current store σ, resulting in a *probability distribution* over $\mathbb{Z}$. For Boolean expressions $[\![\phi]\!] \in \mathsf{BExp}$ and $\sigma \in \Sigma$, we indicate with $\sigma \vDash \phi$ that ϕ holds when the variables in ϕ take values according to σ. Program commands are fairly standard. The command $\texttt{tick}(r)$ consumes $r \in \mathbb{Q}^+$ resource units but otherwise acts as a no-op. The command $x := d$ assigns a value sampled from $d(\sigma)$ to x, for σ the current store. This generalises the usual non-probabilistic assignment $x := e$ for e an integer expression. The commands $\texttt{if [}\psi\texttt{] (}\phi\texttt{) \{C\} \{D\}}$ and $\texttt{while [}\psi\texttt{] (}\phi\texttt{) \{C\}}$ have the usual semantics, with ψ being an invariant. Here, an invariant holds along all probabilistic branches (e.g. probabilistic choice over-approximated with non-determinism) and can in practice be inferred with off-the shelf methods. In case that ψ does not hold, the program terminates abnormally. The command $\texttt{\{C\} <> \{D\}}$ executes either $\texttt{C}$ or $\texttt{D}$, in a non-deterministic fashion. In contrast, the probabilistic choice $\texttt{\{C\}[}p\texttt{]\{D\}}$ executes $\texttt{C}$ with probability $0 \le p \le 1$ and $\texttt{D}$ with probability $1-p$.

We give the small step operational semantics for our language via a *(weighted) probabilistic ARS* $\to$ over *configurations* $\mathsf{Conf} \triangleq (\mathsf{Cmd} \times \Sigma) \cup \Sigma \cup \{\bot\}$. Elements $(\texttt{C},\sigma) \in \mathsf{Conf}$ are denoted by $\langle \texttt{C}\rangle(\sigma)$ and signal that the command $\texttt{C}$ is to be executed under the current store σ, whereas $\sigma \in \mathsf{Conf}$ and $\bot \in \mathsf{Conf}$ indicate that the computation has halted, abnormally in the latter case. The *probabilistic ARS* $\to$ is depicted in Figure 1. In this reduction system, rules have the form $\gamma \xrightarrow{w} \mu$ for $\gamma \in \mathsf{Conf}$ and μ a *multidistribution* over Conf, i.e., countable multisets of the form $\{\!\{p_i : \gamma_i\}\!\}_{i\in I}$ for *probabilities* $0 < p_i \le 1$ with $\sum_{i\in I} p_i \le 1$ and $\gamma_i \in \mathsf{Conf}$ $(i \in I)$. A rule $\gamma \xrightarrow{w} \{\!\{p_i : \gamma_i\}\!\}_{i\in I}$ signals that γ reduces with probability p_i to γ_i, consuming cost w. By identifying dirac multidistributions $\{\!\{1 : \gamma\}\!\}$ with γ, we may write $\gamma \xrightarrow{w} \gamma'$ for a reduction step without probabilistic effect. The *weighted one-step reduction relation* of $\to$ is defined by (i) $\mu \overset{0}{\twoheadrightarrow} \mu$,

$$
\begin{aligned}
\mathsf{et}_c[\mathtt{skip}](f) &\triangleq f \\
\mathsf{et}_c[\mathtt{tick}(r)](f) &\triangleq [c]\cdot \mathbf{r} + f \\
\mathsf{et}_c[\mathtt{abort}](f) &\triangleq \mathbf{0} \\
\mathsf{et}_c[x := d](f) &\triangleq \lambda\sigma.\mathbb{E}_{[\![d]\!](\sigma)}(\lambda i.f(\sigma[x := i])) \\
\mathsf{et}_c[\mathtt{if}\ [\psi]\ (\phi)\ \{\mathtt{C}\}\ \{\mathtt{D}\}](f) &\triangleq [\psi\wedge\phi]\cdot \mathsf{et}_c[\mathtt{C}](f) + [\psi\wedge\neg\phi]\cdot\mathsf{et}_c[\mathtt{D}](f) \\
\mathsf{et}_c[\mathtt{while}\ [\psi]\ (\phi)\ \{\mathtt{C}\}](f) &\triangleq \mu F.[\psi\wedge\phi]\cdot\mathsf{et}_c[\mathtt{C}](F) + [\psi\wedge\neg\phi]\cdot f \\
\mathsf{et}_c[\{\mathtt{C}\}\ \texttt{<>}\ \{\mathtt{D}\}](f) &\triangleq \mathbf{max}(\mathsf{et}_c[\mathtt{C}](f),\mathsf{et}_c[\mathtt{D}](f)) \\
\mathsf{et}_c[\{\mathtt{C}\}\,[p]\,\{\mathtt{D}\}](f) &\triangleq \mathbf{p}\cdot\mathsf{et}_c[\mathtt{C}](f) + (\mathbf{1}-\mathbf{p})\cdot\mathsf{et}_c[\mathtt{D}](f) \\
\mathsf{et}_c[\mathtt{C};\mathtt{D}](f) &\triangleq \mathsf{et}_c[\mathtt{C}](\mathsf{et}_c[\mathtt{D}](f))
\end{aligned}
$$

Figure 2: Definition of expectation transformer $\mathsf{et}_c[\cdot]\colon \mathtt{C}\to\mathbb{T}\to\mathbb{T}$.

(ii) $\{\!\{1:\gamma\}\!\} \xrightarrow{w}\!\!\!\!\rightarrow \mu$ if $\gamma \xrightarrow{w} \mu$, and (iii) $\biguplus_{i\in I} p_i\cdot\mu_i \xrightarrow{w}\!\!\!\!\rightarrow \biguplus_{i\in I} p_i\cdot\nu_i$ where $w = \sum_{i\in I} p_i\cdot w_i$, $\mu_i \xrightarrow{w_i} \nu_i$ for all $i\in I$ and $\sum_{i\in I} p_i \le 1$ for probabilities $0 < p_i \le 1$. Here, $\biguplus_{i\in I} p_i\cdot\mu_i$ denotes the *countable convex union of multidistributions* μ_i $(i\in I)$, e.g., $\frac{1}{2}\cdot\{\!\{1:a\}\!\}\uplus\frac{1}{2}\cdot\{\!\{\frac{1}{3}:a,\frac{1}{2}:b\}\!\} = \{\!\{\frac{1}{2}:a,\frac{1}{6}:a,\frac{1}{4}:b\}\!\}$. Finally, with $\xrightarrow{w}\!\!\!\!\rightarrow^*$ we denote the *weighted multi-step reduction relation* defined by $\mu \xrightarrow{w}\!\!\!\!\rightarrow^* \nu$ if $\mu = \mu_0 \xrightarrow{w_1}\!\!\!\!\rightarrow \cdots \xrightarrow{w_n}\!\!\!\!\rightarrow \mu_n = \nu$ and $w = \sum_{i=1}^{n} w_i$. *Expected cost* and *value functions* for $f\colon\Sigma\to\mathbb{R}^\infty_{\ge 0}$ are defined by

$$\mathsf{ec}[\mathtt{C}](\sigma) \triangleq \sup\{w \mid \langle\mathtt{C}\rangle(\sigma) \xrightarrow{w}\!\!\!\!\rightarrow^* \mu\} \qquad \mathsf{ev}[\mathtt{C}](\sigma)(f) \triangleq \sup\{\mathbb{E}_{\mu\restriction\Sigma}(f) \mid \langle\mathtt{C}\rangle(\sigma) \xrightarrow{w}\!\!\!\!\rightarrow^* \mu\}\ ,$$

where $\mu\restriction\Sigma$ denotes the restriction of μ to elements of Σ and $\mathbb{E}_\nu(f) \triangleq \sum_{i\in I} p_i\cdot f(a_i)$ gives the expected value of f with respect to $\nu = \{\!\{p_i : a_i\}\!\}_{i\in I}$.

Expectation Transformers To overcome the problems concerning composability, Kaminski et al. [4] express the expected runtime in continuation passing style, via an *expectation transformer* $\mathsf{ert}[\cdot]\colon \mathtt{C}\to\mathbb{T}\to\mathbb{T}$ over *expectations* $\mathbb{T}\triangleq\Sigma\to\mathbb{R}^\infty_{\ge 0}$. Given the cost f of executing a program fragment $\mathtt{D}$, $\mathsf{ect}[\mathtt{C}](f)$ computes the cost of first executing $\mathtt{C}$ and then $\mathtt{D}$. We suite this transformer to two transformers $\mathsf{ect}[\mathtt{C}]$ and $\mathsf{evt}[\mathtt{C}]$ that compute the *expected cost* and *expected value function* of the program $\mathtt{C}$, respectively. Their definition coincide up to the case where $\mathtt{C} = \mathtt{tick}(r)$, the former taking into account the cost r while the latter is ignoring it. We thus generalise $\mathsf{ect}[\cdot]\colon \mathsf{Cmd}\to\mathbb{T}\to\mathbb{T}$ and $\mathsf{evt}[\cdot]\colon\mathsf{Cmd}\to\mathbb{T}\to\mathbb{T}$ to a function $\mathsf{et}_c[\mathtt{C}]$ and set $\mathsf{ect}[\mathtt{C}]\triangleq\mathsf{et}_\top[\mathtt{C}]$ and $\mathsf{evt}[\mathtt{C}]\triangleq\mathsf{et}_\bot[\mathtt{C}]$, where $\mathsf{et}_c[\mathtt{C}]$ is given in Figure 2. Here, functions $f\colon(\mathbb{R}^\infty_{\ge 0})^k\to\mathbb{R}^\infty_{\ge 0}$ are extended pointwise on expectations and denoted in bold face, e.g., for each $r\in\mathbb{R}^\infty_{\ge 0}$ we have a constant function $\mathbf{r}(\sigma)\triangleq r$, $f+g\triangleq\lambda\sigma.f(\sigma)+g(\sigma)$ for $f,g\in\mathbb{T}$ etc. For $\phi\in\mathsf{BExp}$ we use Iverson's bracket $[\phi]$ to denote the expectation function $[\phi](\sigma)\triangleq 1$ if $\sigma\vDash\phi$, and $[\phi](\sigma)\triangleq 0$ otherwise. Finally, with $\mu F.e$ we denote the least fixed point of the function $\lambda F.e\colon\mathbb{T}\to\mathbb{T}$ with respect to the *pointwise ordering* $\preceq$ *on expectations*. It can be shown that $(\mathbb{T},\preceq)$ forms an ω-CPO with bottom element $\mathbf{0}$ and top element $\boldsymbol{\infty}$, and that the transformer $\mathsf{et}_c[\mathtt{C}]$ is ω-continuous. Consequently, $\mathsf{et}_c[\mathtt{C}]$ is well-defined.

We note that $\mathsf{evt}[\mathtt{C}]$ coincides with the weakest precondition transformer $\mathsf{wp}[\mathtt{C}]$ of Olmedo et al. [7] on *fully probabilistic programs*, i.e., those without non-deterministic choice. In contrast to $\mathsf{evt}[\mathtt{C}]$, $\mathsf{wp}[\mathtt{C}]$ minimises over non-deterministic choice.

For expectations f, we suite the function $\mathsf{et}_c[\cdot](f)\colon\mathsf{Cmd}\to\Sigma\to\mathbb{R}^\infty_{\ge 0}$ to a function $\underline{\mathsf{et}}_c(f)\colon\mathsf{Conf}\to\mathbb{R}^\infty_{\ge 0}$ by $\underline{\mathsf{et}}_c(f)(\langle\mathtt{C}\rangle(\sigma))\triangleq\mathsf{et}_c[\mathtt{C}](f)(\sigma)$, $\underline{\mathsf{et}}_c(f)(\sigma)\triangleq f(\sigma)$ and $\underline{\mathsf{et}}_c(f)(\bot)\triangleq 0$. The following constitutes our first technical result. What it tells us is that $\mathsf{et}_c[\cdot](f)$ decreases in expectation along reductions, taking into account the cost of steps in the case of $\mathsf{ect}[\cdot](f)$.

Theorem 1. $\mathbb{E}_\mu(\underline{\mathsf{et}}_c(f)) = \sup\{[c] \cdot w + \mathbb{E}_\nu(\underline{\mathsf{et}}_c(f)) \mid \mu \xrightarrow{w}\!\!\!\rightarrow^* \nu\}$.

To prove this theorem, we first show its variations based on the probabilistic ARS $\rightarrow$ and the single-step reduction relation$\twoheadrightarrow$. Both of these intermediate results follow by a straight forward induction on the corresponding reduction relation. The following is then immediate:

Corollary 1 (Soundness and Completeness of Expectation Transformers)**.** *For all commands* $\mathtt{C} \in \mathsf{Cmd}$ *and stores* $\sigma \in \Sigma$*, (i)* $\mathsf{ec}[\mathtt{C}](\sigma) = \mathsf{ect}[\mathtt{C}](\mathbf{0})(\sigma)$ *and (ii)* $\mathsf{ev}[\mathtt{C}](\sigma)(f) = \mathsf{evt}[\mathtt{C}](f)(\sigma)$.

By (i), the expected cost of running $\mathtt{C}$ is given by $\mathsf{ect}[\mathtt{C}](\mathbf{0})$. When $\mathtt{C}$ does not contain loops, the latter is easily computable. To treat loops, Kaminski et al. [4] propose to search for *upper invariants*: $I_f \colon \mathbb{T}$ is an *upper invariant* for $\mathtt{C} = \mathtt{while}\ [\psi]\ (\phi)\ \{\mathtt{D}\}$ with respect to $f \in \mathbb{T}$ if it is a pre-fixpoint of the cost through which $\mathsf{et}_c[\mathtt{C}](f)$ is defined.

Proposition 1 ([4])**.** $[\psi \wedge \phi] \cdot \mathsf{et}_c[\mathtt{D}](I_f) + [\psi \wedge \neg\phi] \cdot f \preceq I_f \implies \mathsf{et}_c[\mathtt{while}\ [\psi]\ (\phi)\ \{\mathtt{D}\}](f) \preceq I_f$.

This immediately suggests the following two stage approach towards an automated expected runtime analysis of a program $\mathtt{C}$ via Corollary 1(i): In the first stage, one evaluates $\mathsf{et}_c[\mathtt{C}](\mathbf{0})$ symbolically on some form of *cost expressions* CExp, generating constraints according to Proposition 1 whenever a while-loop is encountered. Based on the collection of generated constraints, in the second phase concrete upper invariants can be synthesised. From these, a symbolic upper bound to the expected cost $\mathsf{ec}[\mathtt{C}]$ can be constructed. Conceptually, this is the approach taken by $\mathtt{Absynth}$ [6], where $\mathsf{ert}[\mathtt{C}]$ is formulated in terms of a Hoare style calculus, and CExp is amendable to Linear Programming.

Towards A Compositional Analysis With Proposition 1 alone it is in general not possible to modularize this procedure so that individual components can be treated separately. In particular, nested loops generate mutual constraints that cannot be solved independently. Of course, this situation is in general unavoidable as the problem itself is inherently non-modular. Nevertheless, with Theorem 2 drawn below, we give conditions under which this global analysis can be broken down into a local one.

For expectations $\vec{g} = g_1, \dots, g_k$ and $f \colon (\mathbb{R}^\infty_{\geq 0})^k \to \mathbb{R}^\infty_{\geq 0}$, let us denote the composition $\lambda\sigma.f(\vec{g}(\sigma))$ by $f \circ \vec{g}$. Call f *concave* if $f(p \cdot \vec{r} + (1-p) \cdot \vec{s}) \geq p \cdot f(\vec{r}) + (1-p) \cdot f(\vec{s})$ (where $0 \leq p \leq 1$) and (weakly) *monotone* if $\vec{r} \geq \vec{s}$ implies $f(\vec{r}) \geq f(\vec{s})$. The following presents our central observation:

Lemma 1. $\mathsf{ect}[\mathtt{C}](g \circ (g_1, \dots, g_k)) \preceq \mathsf{ec}[\mathtt{C}] + g \circ (\mathsf{evt}[\mathtt{C}](g_1), \dots, \mathsf{evt}[\mathtt{C}](g_k))$ *if g is monotone and concave.*

The intuition behind this lemma is as follows. The functions $g_i \colon \sigma \to \mathbb{R}^\infty_{\geq 0}$, also referred to as *norms*, represent an abstract view on program stores σ. In the most simple case, g_i could denote the absolute value of the i^{th} variable. If g measures the expected resource consumption of $\mathtt{D}$ in terms of g_i, i.e., $\mathsf{ec}[\mathtt{D}](\sigma) \leq g(g_1(\sigma), \dots, g_k(\sigma))$, by monotonicity of $\mathsf{ect}[\mathtt{C}]$ this lemma tells us then that

$$\mathsf{ec}[\mathtt{C;D}](\sigma) \leq \mathsf{ect}[\mathtt{C}](g \circ (g_1, \dots, g_k))(\sigma) \leq \mathsf{ec}[\mathtt{C}](\sigma) + g(\mathsf{evt}[\mathtt{C}](g_1)(\sigma), \dots, \mathsf{evt}[\mathtt{C}](g_k)(\sigma))\ .$$

The expected cost of $\mathtt{C;D}$ is thus the expected cost of $\mathtt{C}$, plus the expected cost of $\mathtt{D}$ measured in the values $\mathsf{evt}[\mathtt{C}](g_i)$ of the norms g_i expected after executing $\mathtt{C}$. Note that concavity can be dropped when $\mathtt{C}$ admits no probabilistic behaviour. Combining this lemma with Proposition 1 then yields:

Theorem 2. *For monotone and concave g,*

$$\begin{aligned}[\psi \wedge \phi] \cdot \big(\mathsf{ec}[\mathtt{C}] + g \circ (\mathsf{evt}[\mathtt{C}](g_1), \dots, \mathsf{evt}[\mathtt{C}](g_k))\big) &\preceq g \circ (g_1, \dots, g_k) \\ \wedge [\psi \wedge \neg\phi] \cdot f \preceq g \circ (g_1, \dots, g_k) &\implies \mathsf{ect}[\mathtt{while}\ [\psi]\ (\phi)\ \{\mathtt{C}\}](f) \preceq g \circ (g_1, \dots, g_k)\ .\end{aligned}$$

Implementation At the moment, we are working on a prototype implementation that accepts pWHILE programs with finite distributions over integer expressions a in probabilistic assignments. *Integer* and *cost expressions* $c, d \in \mathsf{CExp}$ over variables $x \in \mathsf{Var}$, $z \in \mathbb{Z}$, constants $q \in \mathbb{Q}_{\geq 0}$ are given as follows:

$$a, b ::= x \mid z \mid a + b \mid a * b \mid \dots \qquad c, d ::= q \mid \mathsf{nat}(a) \mid [\phi] \cdot c \mid c + d \mid c \cdot d \mid \mathsf{max}(c, d)$$

Norms $\mathsf{nat}(a)$ lift expressions that depend on the store to cost expressions. For brevity, the interpretation of norms is fixed to $\mathsf{nat}(a) \triangleq \max(0,a)$. All other operations are interpreted in the expected way. We denote the evaluation function of cost expressions also by $[\![\cdot]\!] \colon \mathsf{CExp} \to \Sigma \to \mathbb{Q}_{\geq 0}$. Notice that $[\![c]\!] \in \mathbb{T}$. To automate the cost inference of programs we provide a variation of the expectation transformer, $\mathsf{et}_c^\sharp[\cdot] \colon \mathsf{Cmd} \to \mathsf{CExp} \to \mathsf{CExp}$ (as well as $\mathsf{ect}^\sharp$ and $\mathsf{evt}^\sharp$), sound in the following sense:

Theorem 3. $\mathsf{et}_c[C]([\![f]\!]) \preceq [\![\mathsf{et}_c^\sharp[C](f)]\!]$*, for all commands* $\mathtt{C} \in \mathsf{Cmd}$ *and cost expressions* $f \in \mathsf{CExp}$.

The function $\mathsf{et}_c^\sharp[\cdot]$ is defined along the way of $\mathsf{et}_c[\cdot]$ from Figure 2. As an example consider the assignment which is defined by $\mathsf{et}_c^\sharp[x := \{p_1 \colon a_1, \dots, p_2 \colon a_k\}](f) \triangleq \sum_{1 \leqslant i \leqslant k} p_i \cdot f[x/a_i]$. To obtain closed-form expressions on while loops we make use of decomposition (cf. Theorem 2) and should that fail upper invariants (cf. Proposition 1). Notably, using decomposition we can define a recursive strategy that infers bounds on loops individually. We comment on the application of Theorem 2 in the implementation. Assume that we want to compute $\mathsf{ect}^\sharp[\mathtt{while}\ [\psi]\ (\phi)\ \{\mathtt{C}\}](f)$. First, we compute $g = \mathsf{ect}^\sharp[\mathtt{C}](0)$. We heuristically select norms $g_1, \dots, g_k$ based on the invariants and conditions of the program (e.g. $\mathsf{nat}(x-y)$ for condition $x > y$). Second, we recursively compute $h_i = \mathsf{evt}^\sharp[\mathtt{C}](g_i)$ for all g_i. We have $\mathsf{ect}[C](\mathbf{0}) \preceq [\![g]\!]$ and $\mathsf{evt}[C]([\![g_i]\!]) \preceq [\![h_i]\!]$. Third, we express the necessary conditions as constraints over cost expressions:

$$\psi \wedge \phi \vDash g + \mathtt{h} \circ (h_1, \dots, h_k) \leqslant \mathtt{h} \circ (g_1, \dots, g_k)$$
$$\psi \wedge \neg\phi \vDash f \leqslant \mathtt{h} \circ (g_1, \dots, g_k)\ .$$

A constraint $\phi \vDash c \leqslant d$ holds if $[\![\phi]\!] \vDash [\![c]\!] \preceq [\![d]\!]$ holds for all states. When generating constraints only $\mathtt{h}$ is unknown. To obtain a concrete cost expression for $\mathtt{h}$ we follow the method presented in [3]. Here $\mathtt{h}$ is a *template* expression with undetermined coefficients q_i (e.g. $\lambda h_i \to \sum q_i \cdot h_i$), and we search for an assignment such that all constraints hold and $q_i \geqslant 0$. We apply *case-elimination* and *case-distinction* to reduce the problem $\phi \vDash c \leqslant d$ to inequality constraints of polynomials. For example, given a norm $\mathsf{nat}(a) = \max(0,a)$ we eliminate max when we can show that $[\![a]\!] \geqslant 0$ for all states that satisfy ϕ. The obtained inequality constraints of polynomials have undetermined coefficient variables. We reduce the problem to certification of non-negativity, which can then be solved using SMT solvers.

References

[1] M. Avanzini, U. Dal Lago & A. Yamada (2018): *On Probabilistic Term Rewriting*. In: *Proc. of 14th FLOPS*, *LNCS* 10818, Springer, pp. 132–148.

[2] O. Bournez & F. Garnier (2005): *Proving Positive Almost-Sure Termination*. In: *Proc. of 16th RTA*, *LNCS* 3467, Springer, pp. 323–337.

[3] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann & H. Zankl (2007): *SAT Solving for Termination Analysis with Polynomial Interpretations*. In: *Proc. of 10th SAT*, *LNCS* 4501, Springer, pp. 340–354.

[4] B. Lucien Kaminski, J.-P. Katoen, C. Matheja & F. Olmedo (2016): *Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs*. In: *Proc. of 25th ESOP*, *LNCS* 9632, Springer, pp. 364–389.

[5] R. Motwani & P. Raghavan (1995): *Randomized algorithms*. Cambridge university press.

[6] N. C. Ngo, Q. Carbonneaux & J. Hoffmann (2018): *Bounded expectations: resource analysis for probabilistic programs*. In: *Proc. of 39th PLDI*, pp. 496–512.

[7] F. Olmedo, B. Lucien Kaminski, J.-P. Katoen & C. Matheja (2016): *Reasoning about Recursive Probabilistic Programs*. In: *Proc. of 16th LICS*, pp. 672–681.