A Modal Type Theory of Expected Cost in Higher-Order Probabilistic Programs

VINEET RAJANI, University of Kent, United Kingdom
GILLES BARTHE, MPI-SP, Germany and IMDEA Software Institute, Spain
DEEPAK GARG, Max Planck Institute for Software Systems, Germany

The design of online learning algorithms typically aims to optimise the incurred loss or *cost*, e.g., the number of classification mistakes made by the algorithm. The goal of this paper is to build a type-theoretic framework to *prove* that a certain algorithm achieves its stated bound on the cost.

Online learning algorithms often rely on randomness, their loss functions are often defined as expectations, precise bounds are often non-polynomial (e.g., logarithmic) and proofs of optimality often rely on potential-based arguments. Accordingly, we present pλ-amor, a type-theoretic graded modal framework for analysing (expected) costs of higher-order *probabilistic* programs with recursion. pλ-amor is an effect-based framework which uses graded modal types to represent potentials, cost and probability at the type level. It extends prior work (λ-amor) on cost analysis for *deterministic* programs. We prove pλ-amor sound relative to a Kripke step-indexed model which relates potentials with probabilistic coupling. We use pλ-amor to prove cost bounds of several examples from the online machine learning literature. Finally, we describe an extension of pλ-amor with a graded comonad and describe the relationship between the different modalities.

CCS Concepts: • **Theory of computation** → **Type theory**; **Probabilistic computation**; • **Computing methodologies** → **Online learning settings**.

Additional Key Words and Phrases: graded modal types, expected cost, higher-order programs, potentials, probabilistic coupling

ACM Reference Format:
Vineet Rajani, Gilles Barthe, and Deepak Garg. 2024. A Modal Type Theory of Expected Cost in Higher-Order Probabilistic Programs. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 285 (October 2024), 26 pages. https://doi.org/10.1145/3689725

1 Introduction

An important task in online machine learning is to both learn and make optimal decisions incrementally (as data becomes available) over a series of rounds. Algorithms for online learning often rely on randomness to prevent overfitting to specific data that has been seen in the past. The efficacy of such an algorithm is typically measured by a *cost* function, and the goal of algorithm design in this space is to find online algorithms that minimise the *worst-case cost* where the worst-case is over all possible sequences of input data. The cost function is typically an expectation over the algorithm's internal randomness, e.g., the expected number of mistakes made by the algorithm over N rounds (also known as incurred loss), or the incurred loss relative to the minimum possible loss in hindsight (also known as the accumulated regret).

Authors' Contact Information: Vineet Rajani, University of Kent, Canterbury, United Kingdom, V.Rajani@kent.ac.uk; Gilles Barthe, MPI-SP, Bochum, Germany and IMDEA Software Institute, Madrid, Spain, gilles.barthe@mpi-sp.org; Deepak Garg, Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany, dg@mpi-sws.org.

ACM 2475-1421/2024/10-ART285
https://doi.org/10.1145/3689725

For a given online learning algorithm, one often wishes to prove a worst-case upper-bound on the cost function. These bounds can be involved functions (e.g., logarithmic) of the input length, and are often proved using *potential*-based arguments [Arora et al. 2012; Bansal and Gupta 2019; Cesa-Bianchi and Lugosi 2006]. To understand this, consider a probabilistic program that starts in some initial state s, samples from a distribution μ, depending on the outcome a of the sampling (where a is in the support of μ) moves to a state s'_a, incurs a cost $cost(s, s'_a)$ in doing so, and then recurses. To obtain an upper-bound on the cost of such a program, it suffices to show the existence of a potential function $\phi(s)$ s.t. $\phi(s) \geq \mathbb{E}_{a \leftarrow \mu}[cost(s, s'_a) + \phi(s'_a)]$, meaning that the expectation over the cost of a single iteration ($cost(s, s'_a)$) and the remaining potential ($\phi(s'_a)$) is upper-bounded by the initial potential. By repeating this argument for every iteration of the probabilistic program with the remaining potential from the previous round, and using a telescopic sum along with the linearity of expectations, one can show that the starting potential is an upper-bound on the total expected cost.

The goal of this work is to internalise the above potential-based reasoning into a type theory to compositionally prove upper-bounds on the worst-case expected costs of probabilistic recursive programs, such as many online learning algorithms. We note that the potential-based reasoning above is similar to standard potential-based arguments for amortised cost analysis of operations on data structures [Okasaki 1996; Tarjan 1985]. Recent work, λ-amor [Rajani et al. 2021], has shown how a combination of graded modal types, refinement types, and affineness (in the sense of affine logic) can be used to build an expressive type theory for amortised cost analysis for *deterministic* higher-order programs. In this paper, we use λ-amor as a starting point and extend its modal framework to reason about *probabilistic computation and expected cost*. We call our framework pλ-amor (short for probabilistic λ-amor).

In the following, we provide an overview of our work, some of the challenges in the design of pλ-amor, and our contributions.

Type-Level Cost and Probabilistic Effects. In a probabilistic higher-order setting, precise reasoning about expected cost requires quantitative reasoning with symbolic probability distributions at the type level [Avanzini et al. 2019]. Our key insight is to use a *joint probability and expected cost monad*, $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ (\tau(a))$, which represents computations that internally sample a value a from the symbolic distribution μ, eventually outputting something of type $\tau(a)$, and whose *expected cost* (averaged over μ) is upper-bounded by κ. Our syntactic development focuses on the typing rules for this new monad. We also have a graded modality for potentials, $[p]\ \tau$, which represents values of type τ along with an expected potential of p units.

Relating Type-Level and Runtime Probabilistic Effects. Prior type-theoretic work on expected cost analysis either works with finite-support discrete distributions over first-order data [Wang et al. 2020] or assumes a bijection between the sample spaces of the static distribution (in the types) and the runtime distribution (in the semantics) [Avanzini et al. 2019].

In pλ-amor, we overcome both of these limitations. We allow finite-support discrete distributions over higher-order data, and also do not assume a bijection between the static and the dynamic distributions. The key idea behind overcoming these limitations is the development of a logical relation model that uses probabilistic coupling [Villani 2008] to relate the static and dynamic distributions. We believe this insight could be useful even outside of the cost setting. For instance, it could be interesting to investigate if continuous runtime distributions can be analysed using discrete approximations at the level of types, when there is a coupling that relates the two appropriately.

Semantic Model. A natural question that arises when building a modal type theory is about the semantics of the modalities. In this work, we build a Kripke step-indexed logical relation model [Ahmed 2004; Mitchell 1996] for the types of pλ-amor. It not only uses probabilistic coupling to relate the static and dynamic distributions (as mentioned above), but also uses coupling to relate potentials to the expected cost. We believe this is the first paper which explores logical relation models with potential functions, expected cost and probabilistic couplings.

Multiple Modalities and the Interaction between Them. As mentioned above, pλ-amor has two graded modalities: a joint probability and expected cost monad (simply called the *cost monad* in the rest of the paper) and a potential modality. Building on Bounded Linear Logic (BLL) [Girard et al. 1992], many graded modal type theories [Avanzini et al. 2019; Dal Lago and Gaboardi 2011; Dal Lago and Petit 2012; Orchard et al. 2019; Rajani et al. 2021] include a graded comonad for fine-grained resource tracking, such as the number of times each program variable is used. We present an extension of pλ-amor (called pλ-amor$^{\mathsf{C}}$) with such a BLL-style graded comonad and study the interaction among the three modalities. We relate the cost monad and the potential modality via coercion functions forming an isomorphism, and relate the graded comonad and the potential modality via two distributive laws that we internalise as subtyping rules in the extended theory.

Summary of Contributions. To summarise, we make the following technical contributions:

- A modal type theory, pλ-amor, for reasoning about the expected cost of recursive higher-order probabilistic programs using potentials.
- A Kripke, step-indexed model of types that uses probabilistic couplings to model our cost monad and potentials, and a proof that pλ-amor is sound relative to this model.
- Verification of expected cost bounds (expected loss and regret) for several examples from the online machine learning literature, which we believe cannot be verified using existing formal approaches.
- An extension of pλ-amor with a BLL-style graded comonad, and a study of the interaction between the three modalities (the monad, the comonad, and potentials).

Organisation. We begin with relevant background material in section 2. We cover online machine learning (section 2.1), basics of λ-amor (section 2.2) that is the prior work we build upon, and necessary concepts from probability theory (section 2.3). Section 3 describes the language (statics and dynamics) and the type system of pλ-amor. It also describes the semantic model of types, which we use to prove the soundness of pλ-amor. Section 4 describes applications of pλ-amor by verifying expected costs of several problems from the online learning literature. In section 5, we describe an extension of pλ-amor with a graded comonad. This extension is named. We describe the key changes needed in the type theory and the model to support the graded comonad. We also describe how the graded comonad relates to the potential modality through distributive laws. Section 6 describes related work and section 7 concludes the paper.

A technical appendix [Rajani et al. 2024] contains proofs of theorems and additional details that are omitted from this paper due to space restrictions.

Limitations and Scope. Our focus is on developing the theoretical foundations of our type theory. The implementation and mechanisation of this theory is out of the scope of this paper, but would be an interesting direction of future work.

2 Background

We start with background material on online machine learning, a high-level overview of λ-amor, and relevant concepts from probability theory.

2.1 Online Machine Learning with an Illustrative Example

Online machine learning deals with the problem of sequential decision making under uncertainty. Unlike the batch learning approach, where the model is trained a priori on the entire dataset before the model is used for decision making, in an online setting both the learning and decisions are made *as data becomes available* over a sequence of trails or rounds.

We illustrate the key characteristics of online learning algorithms using a well-known algorithm, Randomised Weighted Majority (RWM) [Arora 2013; Arora et al. 2012], which solves the problem of prediction with experts' advice [Arora et al. 2012]. Specifically, the goal is to make a sequence of binary predictions (one per round), e.g., whether a stock's price will go up or down each day. The prediction can use the advice of experts (formally, a finite list of functions passed as inputs), but the algorithm does not know a priori which expert(s) give the correct advice in any given round. In fact, the experts could be adversarial and might make wrong predictions deliberately. This is countered using randomisation. At the end of each round, the ground truth is revealed, which the online algorithm can use to improve its decisions in the future, e.g., by assigning appropriate weights to each of those experts. Naturally, a good algorithm is one that incurs the minimum loss (number of mistakes) in expectation, typically expressed as a bound on the expected loss of the algorithm.

We describe an encoding of the RWM algorithm in a Haskell-like functional language as a higher-order program (Fig. 1). The function rwm takes four inputs: the number of rounds (of type nat), a list of experts (each expert is a function from nat to bool), a learning rate eta (a real number) and an initial list of weights, one for each expert (also real numbers). It produces a distribution over the list of predictions made in each round (of the type $\mathbb{P}\,(L\,\mathbf{B})$, where $\mathbb{P}$ is the Giry monad of probability distributions, L is the list type constructor and $\mathbf{B}$ is the boolean type).[1] If the number of remaining rounds is zero then the algorithm just returns a point distribution over an empty list. Otherwise, it follows the steps of the RWM algorithm outlined above (lines 6–12). The $getAdvice$ function simply gets the advice of all the experts for that round. The $makePred$ function randomly samples an expert, weighing each expert in proportion to its current weight, and returns the advice of that expert. This sampled advice, pr, is the algorithm's outcome for the round. The $getAnswer$ function (not shown in the figure) is an oracle that provides the ground truth for that round. This ground truth is later used to adjust the weights of the experts using the $chgWts$ function, which reduces the weight of all the experts who answered incorrectly by a factor $1 - eta$. This procedure is repeated for the remaining rounds, and finally, a distribution over the list of predictions, one per round, is returned as the output.

A natural question about this algorithm is the following: can we get an upper-bound on the expected number of mistakes (often referred to as the expected loss) made by the algorithm over a certain number of rounds? The answer to this question is yes. In particular, the following theorem shows such a bound on the expected loss of rwm.

Theorem 1 (Expected loss of rwm)**.** Let (1) T be the number of rounds for which we want to make predictions with rwm (i.e., T is the initial value of $round$), (2) t be a counter over rounds starting from $t = 0$ (so, $t \triangleq T - round$ and round $t = T$ is an additional trivial round at the end that terminates immediately in the Z case of the algorithm), (3) n be the number of experts, (4) η denote the learning rate, (5) $w_i(t)$ be the weight of the i^{th} expert at the beginning of round t, such that $w_i(t) \geq 1/(1-\eta)^T$, (6) $loss(t)$ denote the loss of the algorithm in round t, and (7) $\phi(t) \triangleq \sum_{i=0}^{n-1} w_i(t)$. Then,

$$\mathbb{E}\left[\sum_{t=0}^{T-1} loss(t)\right] \leq \frac{\log(\phi(0))}{\eta} - \frac{\log(\phi(T))}{\eta}$$

[1] We later refine $\mathbb{P}\,\tau$ with probabilities and expected cost to get the monad $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau(a)$ from section 1.

```
rwm : N  → L(N → B)  → R  → L  R
      → ℙ (L  B)
fix rwm.λ round experts eta wts.
matchN round
,Z ↦ (return nil)
,S rnd ↦
  let advs = getAdvice round experts in
  bind pr = makePred advs wts in
    let ans = getAnswer round in
    let nw = chgWts wts advs ans eta in
    bind rec = rwm rnd experts eta nw in
      return (pr :: rec)
```

```
getAdvice : N → L(N → B) → L  B
getAdvice ≜
  λ r exp. map (λ e. e r) exp
```

```
makePred : L  B → L  R → ℙ (L  B)
makePred ≜ λ advs wts.
let phi = sum wts in
let prob = map (λx. (x/phi)) wts in
bind x = toDist prob in
  let pred = lookup x advs in
  return pred
```

```
chgWts : L  R → L  B → B → R
         → L  R
chgWts ≜  λ weights advs ans eta.
map (λp. let ⟨an, wt⟩ = p in
            if an == ans
              then wt
              else wt ∗ (1 − eta)
        ) (zip advs weights)
```

Fig. 1. The Randomised Weighted Majority (RWM) algorithm for prediction with experts' advice

The proof of theorem 1 relies on a potential-based argument. In particular, a potential of $\frac{\log(\phi(t))}{\eta}$ units is sufficient to account for the expected cost of the t^{th} and subsequent rounds. The constraint on the weights (in condition 5 of the theorem) is required to ensure that the potential always remains non-negative. The method of potentials [Tarjan 1985] is sound only under this positivity assumption. In the following, we give the intuition behind the proof. Later, we formalise the proof in our type theory (Section 4.2).

From the algorithm, the probability of selecting the i^{th} expert in round t is $\frac{w_i(t)}{\phi(t)}$. Define the loss of the i^{th} expert in the t^{th} round, $loss_i(t)$, as 0 if the i^{th} expert predicts correctly in the t^{th} round, and 1 otherwise. Then, the expected loss of the algorithm in the t^{th} round is $\mathbb{E}[loss(t)] = \frac{1}{\phi(t)} \sum_{i<n} w_i(t) \cdot loss_i(t)$, where n is the number of experts. From the algorithm, we also know that the weight of the i^{th} expert in the $(t+1)^{th}$ round is $w_i(t) \cdot (1 - \eta \cdot loss_i(t))$. Using this, we calculate:

$$\begin{array}{lcl}
\phi(t+1) & = & \sum_{i<n} w_i(t) \cdot (1 - \eta \cdot loss_i(t)) \\
 & = & \sum_{i<n} w_i(t) - \eta \sum_{i<n} w_i(t) \cdot loss_i(t) \\
 & = & \phi(t)(1 - \eta \cdot \mathbb{E}[loss(t)]) \\
 & \leq & \phi(t) \cdot e^{-\eta \cdot \mathbb{E}[loss(t)]} \\
\mathbb{E}[loss(t)] & \leq & \frac{\log(\phi(t))}{\eta} - \frac{\log(\phi(t+1))}{\eta}
\end{array} \qquad (1)$$

Equation (1) gives a bound on the expected loss of rwm in the t^{th} round. If we define the input potential for the t^{th} round to be $\frac{\log(\phi(t))}{\eta}$, then the expected loss in the t^{th} round is bounded by the change in potential when going from round t to round $t+1$, i.e., $\frac{\log(\phi(t))}{\eta} - \frac{\log(\phi(t+1))}{\eta}$. Repeating this argument over all the rounds from $t = 0$ to $t = T$ one obtains a telescopic sum, resulting in the bound the sum of expected loss over all the rounds by $\frac{\log(\phi(0))}{\eta} - \frac{\log(\phi(T))}{\eta}$. Finally, using linearity of expectation we can conclude the proof of Theorem 1.

$$\frac{\Theta \vdash \kappa : \mathbb{R}^{+}}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \uparrow^{\kappa} : \mathbb{M}\,\kappa\,\mathbf{1}}\ \text{T-tick} \qquad \frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{return}\, e : \mathbb{M}\,0\,\tau}\ \text{T-ret}$$

$$\frac{\begin{array}{c}\Psi; \Theta; \Delta; \Omega; \Gamma_1 \vdash e_1 : \mathbb{M}\,\kappa_1\,\tau_1 \\ \Psi; \Theta; \Delta; \Omega; \Gamma_2, x : \tau_1 \vdash e_2 : \mathbb{M}\,\kappa_2\,\tau_2 \\ \Theta \vdash \kappa_1 : \mathbb{R}^{+} \qquad \Theta \vdash \kappa_2 : \mathbb{R}^{+}\end{array}}{\Psi; \Theta; \Delta; \Omega; \Gamma_1 + \Gamma_2 \vdash \mathsf{bind}\ \ e_1 = x\ \mathsf{in}\ e_2 : \mathbb{M}(\kappa_1 + \kappa_2)\,\tau_2}\ \text{T-bind}$$

$$\frac{\begin{array}{c}\Psi; \Theta; \Delta; \Omega; \Gamma_1 \vdash e_1 : [\kappa_1]\,\tau_1 \qquad \Psi; \Theta; \Delta; \Omega; \Gamma_2, x : \tau_1 \vdash e_2 : \mathbb{M}(\kappa_1 + \kappa_2)\,\tau_2 \\ \Theta \vdash \kappa_1 : \mathbb{R}^{+} \qquad \Theta \vdash \kappa_2 : \mathbb{R}^{+}\end{array}}{\Psi; \Theta; \Delta; \Omega; \Gamma_1 + \Gamma_2 \vdash x = \mathsf{release}\, e_1;\ e_2 : \mathbb{M}\,\kappa_2\,\tau_2}\ \text{T-release}$$

$$\frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau \qquad \Theta \vdash \kappa : \mathbb{R}^{+}}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{store}\, e : \mathbb{M}\,\kappa\,([\kappa]\,\tau)}\ \text{T-store} \qquad \frac{\Psi; \Theta, a; \Delta, a < I; \Omega; . \vdash e : \tau}{\Psi; \Theta; \Delta; \sum_{a<I} \Omega; . \vdash !e :!_{a<I}\tau}\ \text{T-subExpI}$$

$$\frac{\Psi; \Theta; \Delta; \Omega_1; \Gamma_1 \vdash e : (!_{a<I}\tau) \qquad \Psi; \Theta; \Delta; \Omega_2, x :_{a<I} \tau; \Gamma_2 \vdash e' : \tau'}{\Psi; \Theta; \Delta; \Omega_1 + \Omega_2; \Gamma_1 + \Gamma_2 \vdash \mathsf{let}\,!\,x = e\ \mathsf{in}\ e' : \tau'}\ \text{T-subExpE}$$

Fig. 2. A subset of the typing rules of λ-amor [Rajani et al. 2021]

To summarise, our illustrative example, RWM, highlights several key characteristics of online learning algorithms [Arora et al. 2012; Bansal and Gupta 2019; Cesa-Bianchi and Lugosi 2006]: it is higher-order and recursive, it uses randomisation, and it has a potential-based proof.

2.2 λ-amor

Potential-based reasoning is not limited to the analysis of probabilistic programs like RWM. For instance, the domain of amortised complexity analysis [Tarjan 1985] makes crucial use of potential functions even in a deterministic setting. λ-amor [Rajani et al. 2021] is a graded modal type theory for the amortised cost analysis of *deterministic* higher-order functional programs. Our work builds on λ-amor and generalises it to a probabilistic setting, which is why we call our framework pλ-amor. In this subsection, we recapitulate the salient points of λ-amor.

The typing judgment of λ-amor, $\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau$, says that an expression e has type τ under the assumptions specified by five contexts: Ψ is a map from type variables to kinds, Θ is a map from index variables to sorts, Δ is a set of constraints on indices, Ω is a map from non-affine variables to their types and multiplicities, and Γ is a map from affine variables to their types. Here, we give an overview of the typing rules for the three modalities used by λ-amor (listed in Fig. 2).

Cost Monad. λ-amor uses a cost monad to track *cost*: $\mathbb{M}\,\kappa\,\tau$ is the type – technically, a graded monad – of expressions with underlying type τ and a cost of at most κ units when forced. T-ret is the typing rule for the return of the monad, which simply injects a pure term of type τ into the monad with 0 cost. T-bind is the rule for sequencing monadic terms; it sums up the costs of the terms being sequenced. Finally, λ-amor has a construct 'tick', denoted $\uparrow^{\kappa}$, which incurs cost κ and returns a unit value (rule T-tick). This construct is the only way to obtain a non-zero cost in a λ-amor program.

Potential Modality. Potentials are the key to amortised cost analysis [Tarjan 1985] and λ-amor uses a separate modality to handle potentials at the type level. The type $[p]\,\tau$ specifies the type of a term that has potential p stored with it. There are two ways to manipulate potential. First, potential can be attached to a type using the 'store' construct. Second, potential can be detached from a type using the 'release' construct. Storing potential (T-store) with a type is a costful operation and incurs the same cost as the potential stored. Dually, releasing potential (T-release) reduces the cost of a subsequent computation by an amount equal to the potential released.

Multiplicity-Graded Comonad. Finally, λ-amor inherits the standard graded sub-exponential, a graded comonad, ($!_{i<n}\tau$) from Bounded Linear Logic [Girard et al. 1992]). This comonad tracks the number of times a term may be used. Informally, $!_{i<n}\tau$ is the type of a term that has n variant copies of types $\tau[0/i], \ldots, \tau[(n-1)/i]$. Morally, $!_{i<n}\tau$ is the affine iterated tensor product $\tau[0/i] \otimes \tau[1/i] \ldots \otimes \tau[(n-1)/i]$. T-subExpI is the rule for introducing a $!_{i<n}\tau$. It says that, under a non-affine context Ω and an empty affine context (denoted by $\cdot$), if a term e can be assigned a type τ (with a free, s.t. $a < I$), then under I copies of the non-affine context (denoted by $\sum_{a<I}\Omega$), $!e$ can be assigned a type $!_{a<I}\tau$. T-subExpE is the dual rule, which says that eliminating a term of type $!_{a<I}\tau$ yields I copies of that term (bound to x) in its continuation (denoted by $x :_{a<I} \tau$).

2.3 Probability Theory Preliminaries

Next, we summarise relevant concepts of probability theory.

Definition 2 (Discrete probability distribution)**.** A (total) discrete probability distribution, μ, consists of a carrier set S and a measure M on that set, i.e., $M : S \longrightarrow [0,1]$ s.t. $\sum_{a\in S}(M\,a) = 1$. μ is a *sub-distribution* when $\sum_{a\in S} M\,a \leq 1$.

Definition 3 (Marginals)**.** For $S = S_1 \times S_2$, the first marginal of a product distribution (S, M) is defined as $\mathbb{M}_1(S, M) \triangleq (S_1, M_1)$ where $M_1(x) \triangleq \sum_{y\in S_2} M(x, y)$. The second marginal $\mathbb{M}_2(S, M)$ is defined symmetrically.

A probabilistic coupling (or simply coupling) is a fundamental concept used in the analysis of many probabilistic processes [Villani 2008].

Definition 4 (Coupling)**.** Let $\mu_1 = (S_1, M_1)$ and $\mu_2 = (S_2, M_2)$ be two distributions. A distribution μ over the product space $S_1 \times S_2$ is a coupling of μ_1 and μ_2, denoted $\mu : \mu_1 \leftrightarrow \mu_2$, iff $\mathbb{M}_1(\mu) = \mu_1$ and $\mathbb{M}_2(\mu) = \mu_2$.

Finally, the convolution is a specific way of creating a product distribution (see below).

Definition 5 (Convolution)**.** Let $\mu_1 = (S_1, M_1)$ be a distribution and let $\mu_2(a) = (S_2(a), M_2(a))$ for $i \in S_1$ be an S_1-indexed family of distributions. We define the convolution $(\mu_1 \otimes a.\mu_2) \triangleq (S, M)$, where $S \triangleq \{(i, j) \mid i \in S_1 \text{ and } j \in S_2(i)\}$ and $M(i, j) \triangleq M_1(i) \cdot M_2(i)(j)$.

Note that if μ_1 and μ_2 are total distributions and μ_2 does not depend on μ_1 (i.e., $\mu_2(i)$ is independent of i), then their convolution is a (trivial) coupling between them.

3 Pλ-amor

This section describes our language and type theory, which we refer to as pλ-amor. pλ-amor is an *affine* type theory with *refinement types* and *two graded modalities* for the purpose of cost analysis: a monad graded with probability and expected cost and a graded modal type for potentials.

3.1 Statics

Indices and Constraints. An *index term* (*pl.* index terms or static indices or indices) represents grades like costs and potentials, and standard type refinements like list lengths. Indices are sorted.

$$
\begin{array}{llcl}
\text{Types} & \tau & ::= & \mathbf{1} \mid \mathbf{B}(I) \mid \mathbf{N}(I) \mid \mathbf{R}(I) \mid L_{i<I}\ \tau \mid \tau_1 \multimap \tau_2 \mid \tau_1 \otimes \tau_2 \mid \tau_1 \,\&\, \tau_2 \mid \tau_1 + \tau_2 \mid \\
 & & & !\,\tau \mid \exists x{:}S.\tau \mid \forall x{:}S.\tau \mid \forall \alpha.\tau \mid \alpha \mid C\&\tau \mid C \Rightarrow \tau \mid\ [I]\,\tau \mid\ \mathbb{PC}_{(a \leftarrow \mu)}\ I\ \tau \\
\text{Index term } I, \kappa, p, n & & ::= & i \mid B \mid N \mid R \mid I + I \mid \lambda_s i : S.I \mid I\ I \mid \text{if } I\ I\ I \mid I == I \mid log\ I \mid exp\ I \mid \mu \\
\text{Static Distribution} & \mu & ::= & (C_s, M_s) \\
 & & & C_s = \text{Finite set of indices and } M_s : C_s \longrightarrow \mathbb{R}^+[0,1] \\
\text{Sort} & S & ::= & \mathbb{B} \mid \mathbb{N} \mid \mathbb{R} \mid (S,S) \mid\ S \rightarrow S\ \mid \mathbb{D}_S \mid \mathbb{S}_S \\
\text{Constraints} & c & ::= & I = I \mid I < I \mid c \Rightarrow c \mid c \wedge c \mid (1 - I) \leq exp\ (-I) \mid \\
 & & & I > 0 \Rightarrow exp\ (-I) \leq 1 - I + I^2
\end{array}
$$

Fig. 3. Types of pλ-amor

The basic sorts are: booleans ($\mathbb{B}$), natural numbers ($\mathbb{N}$) and real numbers ($\mathbb{R}$). Standard sort-specific operations like conjunction/disjunction (on $\mathbb{B}$) and addition/subtraction (on $\mathbb{N}$, $\mathbb{R}$) are supported, as are functions like exponential and logarithm (on $\mathbb{R}$). We also support pairs of indices, index-level functions and their applications. Finally, a discrete probability distribution over a finite set of indices can be represented as an index term. Such an index term, denoted μ, is a pair (C_s, M_s) of a carrier set C_s of indices and a measure $M_S : C_S \longrightarrow [0, 1]$.

Constraints are predicates over indices, like $<$ and $>$. A particular constraint that we need for analysing the RWM example in section 4.2 is $1 - x \leq e^{-x}$ (this constraint holds for all $x : \mathbb{R}$.) Similarly, a constraint that we need for analysing the multi-armed bandit problem in section 4.3 is $e^{-x} \leq 1-x+x^2$ (this constraint holds for all $x : \mathbb{R} > 0$). Other constraints can be added axiomatically for verifying programs, as is standard in refinement type systems.

Types. pλ-amor inherits standard types of intuitionistic affine logic including the affine function space ($\multimap$), multiplicative pairs ($\otimes$), additive pairs ($\&$), the affine sum type ($+$) and the exponential ($!$). pλ-amor also has quantification over indices – both existential ($\exists i{:}S.\tau$) and universal ($\forall i{:}S.\tau$), type quantification ($\forall \alpha.\tau$), and constraint types ($C \Rightarrow \tau$ and $C\&\tau$).

We also include length-refined list type $L_{i<n}\ \tau(i)$, which ascribes lists of length n whose ith element has type $\tau(i)$, and singleton types over booleans, natural numbers and reals. Standard bool, natural number and real number types can be coded using existential quantification over indices. For example, if n is of the sort $\mathbb{N}$, then $\mathbf{N}(n)$ is the singleton type of terms that represent the number n. The usual type $\mathbf{N}$ can be defined as $\exists n.\mathbf{N}(n)$. All types are assorted into *kinds*, but we elide the standard details of kinds here.

Cost in pλ-amor is tracked using a graded monad, $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau(a)$, which is doubly graded with a static distribution $\mu = (C_s, M_s)$ and an expected cost κ. The index a may appear free in the type τ but not in κ. To a first approximation, this monadic type ascribes computations that, for every $a \in C_s$, produces a result of type $\tau(a)$ with probability $M_s(a)$, and whose *expected* cost (over μ) is upper-bounded by κ. The precise definition of the monadic type, shown in Figure 6, stipulates the existence of a coupling between the static distribution μ over static indices and the runtime distribution over language terms produced by the ascribed computation. This is unlike prior work [Avanzini et al. 2019], which requires that these two distributions be the same. Our generalisation is necessary to give a semantic interpretation to the coupling-based subtyping introduced in Section 3.2, which prior work did not examine.

Finally, pλ-amor includes a modality $[p]\ \tau$ for representing potentials at the type level. Like the similar-looking modality of λ-amor, our modality ascribes values of type τ paired with p units of potential. However, unlike λ-amor, potentials in pλ-amor are used to offset *expected* cost, not deterministic cost, as explained in Sections 3.2 and 3.5.

Expressions and Values. The syntax of pλ-amor terms is described in Fig. 4. We only describe key terms pertaining to the cost monad and the potential modality here. The monadic type has a return (return e) and bind (bind $x = e_1$ in e_2) to create and compose distributions over values of the underlying type of the monad. As in λ-amor, the tick construct ($\uparrow^{\kappa}$) adds a cost of κ units to the computation. It can be placed appropriately within a program to model the program's cost [Danielsson 2008]. The term toDist e treats a list of real numbers, e, as weights and converts them into a probability distribution over a set of size length(e). Finally, dynamic (discrete) distributions ν are pairs (V_d, M_d), which mirror the structure of static distributions, except that, now, V_d is a carrier set of language terms (not static indices), and M_d is a function from that set to $[0, 1]$.

The potential type $[p]\ \tau$ has the same values as the type τ. This is because potentials are *ghost* – they are used only in proofs and do not appear at runtime. Two primary operations on potentials include releasing and storing them. They are required, for instance, to formalise the informal proof of Theorem 1 from section 2.1. The intuitive idea is as follows. In the proof of the *rwm* example, the input potential of $\log(\phi(t))/\eta$ units needs to be released before it can be used to pay for the cost of the t^{th} round. Similarly, the remaining potential of $\log(\phi(t+1))/\eta$ units needs to be stored back before we can continue reasoning about the $(t+1)^{th}$ round. These steps are formalised in pλ-amor using the constructs release and store, which mirror similar constructs in λ-amor but are generalised to our probabilistic setting. The term release : $[\kappa]\ \tau \multimap \mathbb{PC}_{(a \leftarrow \mu)}\ (\kappa + \kappa')\ \tau' \multimap \mathbb{PC}_{(a \leftarrow \mu')}\ \kappa'\ \tau'$, releases the input potential of κ units from the first argument (of type $[\kappa]\ \tau$) to offset κ units of cost from its second argument. The term store : $\tau[0/a] \multimap \mathbb{PC}_{(a \leftarrow \delta)}\ \kappa\ ([\kappa]\ \tau)$ attaches potential of κ units to its argument (which is of type $\tau[0/a]$).

Additionally, pλ-amor introduces two new constructs for working with potentials. The term Swap : $([p]\ \tau_1 \otimes \tau_2) \multimap (\tau_1 \otimes [p]\ \tau_2)$ transfers potential from the first component of a pair to the second component whereas the term Split : $[p]\ \mathbf{1} \multimap ([p_1]\ \mathbf{1} \otimes [p_2]\ \mathbf{1})$ divides the input potential p into two positive parts p_1 and p_2 (s.t. $p \geq p_1 + p_2$). Inside the cost monad, the functionality of Swap and Split can be simulated using the store and release constructs. Swap and Split add this functionality outside the monad, which is convenient for typing some programs.

3.2 Type System

The typing judgment of pλ-amor, $\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau$, is similar to that of λ-amor syntactically. The difference is that, in pλ-amor, we do not track multiplicities, so Ω is a context mapping non-affine (!-ed) variables to their types only. We add multiplicities later in an extension of pλ-amor (Section 5).

Typing Rules. Selected typing rules of pλ-amor are shown in Fig. 4. **T-ret** says that if e is of type τ, then return e is a point distribution over type τ with 0 expected cost. **T-bind** composes probabilities and expected cost sequentially. The static distribution of bind $x = e_1$ in e_2 (in the conclusion) is the convolution μ of the static distribution μ_1 of e_1 and the static distribution family $\mu_2(a)$ of e_2 for every a in the support of μ_1. The static cost of bind $x = e_1$ in e_2 is the sum of the static cost of e_1 and the expected cost of e_2 (with the expectation taken over μ_1). The free index variables a and b in τ_2 are substituted by the projections of samples drawn from μ. In contrast, the return and bind of the prior work λ-amor do not handle probabilities and work only in a deterministic setup.

T-store types store e. It stores κ units of potential with e, incurring a cost of κ units. Intuitively, T-store ensures that potentials cannot be obtained for free: storing κ units of potential costs κ units. Like T-ret, the distribution resulting from store e is a point distribution.

T-release is the typing rule for release $x = e_1$ in e_2. It offsets κ units of cost from the cost of e_2 (which is $\kappa + \kappa'$ units), by releasing the κ units of potential stored with e_1. The rule is sound due to the *linearity of expectations.* The distribution of the result is the same as the distribution of

$$\begin{array}{llll}
\text{Expressions} & e & ::= & v \mid x \mid S\ e \mid e_1\ e_2 \mid e :: e \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \mathsf{if}\ e\ e_1\ e_2 \mid () \mid \\
 & & & \langle\!\langle e_1, e_2 \rangle\!\rangle \mid \mathsf{let}\langle\!\langle x, y \rangle\!\rangle = e_1\ \mathsf{in}\ e_2 \mid \langle e, e \rangle \mid \mathsf{fst}(e) \mid \mathsf{snd}(e) \mid \mathsf{inl}(e) \mid \\
 & & & \mathsf{inr}(e) \mid \mathsf{case}\ e\ \mathsf{of}\ x.e;\ y.e \mid \mathsf{let}\,!\,x = e_1\ \mathsf{in}\ e_2 \mid \mathsf{Split}\ \ e \mid \mathsf{Swap}\ \ e \\
\text{Values} & v & ::= & () \mid \mathsf{tt} \mid \mathsf{ff} \mid Z \mid r \mid \lambda x.e \mid \mathsf{fix}\ x.e \mid \langle\!\langle v_1, v_2 \rangle\!\rangle \mid \langle v, v \rangle \mid \mathsf{inl}(v) \mid \mathsf{inr}(v) \mid \\
 & & & !\,e \mid nil \mid v_1 :: v_2 \mid \mathsf{return}\,e \mid \mathsf{bind}\ \ e_1 = x\ \mathsf{in}\ e_2 \mid \uparrow^I \mid \mathit{toDist}\ e \mid \\
 & & & \nu \mid \mathsf{store}\,e \mid \mathsf{release}\ \ x = e_1\ \ \mathsf{in}\ \ e_2 \\
\text{Dynamic Distribution} & \nu & ::= & (V_d, M_d) \\
 & & & V_d = \text{Finite set of values and } M_d : V_d \longrightarrow [0,1]
\end{array}$$

$$\frac{\Theta; \Delta \vdash \kappa : \mathbb{R}^+ \qquad \delta_0 = (\{0\}, \{0 \mapsto 1\})}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \uparrow^\kappa : \mathbb{PC}_{(a \leftarrow \delta_0)}\ \kappa\ \mathbf{1}}\ \text{T-tick}$$

$$\frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau[0/a] \qquad \delta_0 = (\{0\}, \{0 \mapsto 1\})}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{return}\,e : \mathbb{PC}_{(a \leftarrow \delta_0)}\ 0\ \tau}\ \text{T-ret}$$

$$\frac{\begin{array}{c}
\Psi; \Theta; \Delta; \Omega; \Gamma_1 \vdash e_1 : \mathbb{PC}_{(a \leftarrow \mu_1)}\ \kappa_1\ \tau_1 \\
\Psi; \Theta, a; \Delta; \Omega; \Gamma_2, x : \tau_1 \vdash e_2 : \mathbb{PC}_{(b \leftarrow \mu_2)}\ \kappa_2\ \tau_2 \qquad \kappa \geq \kappa_1 + \displaystyle\sum_{a \in \pi_1(\mu_1)} \pi_2(\mu_1)\ a\ \cdot\ \kappa_2\ a \\
\mu = \mu_1 \otimes a.\mu_2 \qquad \Psi; \Theta, c; \Delta, c \in \pi_1(\mu) \vdash \tau_2[\pi_1(c)/a][\pi_2(c)/b] <: \tau \\
\{a, b\} \notin \mu, \kappa, \tau \qquad \Theta; \Delta \vdash \kappa_1 : \mathbb{R}^+ \qquad \Theta; \Delta \vdash \kappa_2 : \mathbb{R}^+
\end{array}}{\Psi; \Theta; \Delta; \Omega; \Gamma_1 \oplus \Gamma_2 \vdash \mathsf{bind}\ \ x = e_1\ \mathsf{in}\ e_2 : \mathbb{PC}_{(c \leftarrow \mu)}\ \kappa\ \tau}\ \text{T-bind}$$

$$\frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau[0/a] \qquad \Theta; \Delta \vdash \kappa : \mathbb{R}^+ \qquad \delta_0 = (\{0\}, \{0 \mapsto 1\})}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{store}\,e : \mathbb{PC}_{(a \leftarrow \delta_0)}\ \kappa\ ([\kappa]\ \tau)}\ \text{T-store}$$

$$\frac{\begin{array}{c}
\Psi; \Theta; \Delta; \Omega; \Gamma_1 \vdash e_1 : [\kappa]\ \tau \\
\Psi; \Theta; \Delta; \Omega; \Gamma_2, x : \tau \vdash e_2 : \mathbb{PC}_{(a \leftarrow \mu)}\ (\kappa + \kappa')\ \tau' \qquad \Theta; \Delta \vdash \kappa : \mathbb{R}^+ \qquad \Theta; \Delta \vdash \kappa' : \mathbb{R}^+
\end{array}}{\Psi; \Theta; \Delta; \Omega; \Gamma_1 \oplus \Gamma_2 \vdash \mathsf{release}\ \ x = e_1\ \ \mathsf{in}\ \ e_2 : \mathbb{PC}_{(a \leftarrow \mu)}\ \kappa'\ \tau'}\ \text{T-release}$$

$$\frac{\begin{array}{c}
\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : L_{i<n}\,\mathbf{R}(p(i)) \qquad \Theta; \Delta \vdash p : \mathbb{N} \to \mathbb{R} \qquad \Theta; \Delta \models n > 0 \\
\mu = (\{0 \ldots n-1\}, \{i \mapsto p(i) \mid i < n\}) \qquad \displaystyle\sum_{i<n} p(i) = 1
\end{array}}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathit{toDist}\ e : \mathbb{PC}_{(i \leftarrow \mu)}\ 0\ \mathbf{N}(i)}\ \text{T-toDist}$$

$$\frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : [p]\ \mathbf{1} \qquad \Theta; \Delta \vdash p \geq p_1 + p_2 \qquad \Theta; \Delta \vdash p_1 : \mathbb{R}^+ \qquad \Theta; \Delta \vdash p_2 : \mathbb{R}^+}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{Split}\ \ e : ([p_1]\ \mathbf{1} \otimes [p_2]\ \mathbf{1})}\ \text{T-split}$$

$$\frac{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : [p]\ \tau_1 \otimes \tau_2}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{Swap}\ \ e : \tau_1 \otimes [p]\ \tau_2}\ \text{T-swap} \qquad \frac{\Psi; \Theta; \Delta \vdash \tau <: \tau' \qquad \Psi; \Theta; \Delta \vdash p' \leq p}{\Psi; \Theta; \Delta \vdash [p]\ \tau <: [p']\ \tau'}\ \text{sub-potential}$$

$$\frac{\Psi; \Theta; \Delta \vdash \exists \alpha : \mu \leftrightarrow \mu' \qquad \Psi; \Theta, a, a'; \Delta, \alpha(a, a') > 0 \vdash \tau <: \tau' \qquad \Psi; \Theta; \Delta \vdash \kappa \leq \kappa'}{\Psi; \Theta; \Delta \vdash \mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau <: \mathbb{PC}_{(a' \leftarrow \mu')}\ \kappa'\ \tau'}\ \text{sub-coupling}$$

Fig. 4. Language syntax, selected typing and subtyping rules of pλ-amor

$$\frac{}{\uparrow^{\kappa} \Downarrow^{\kappa} (\{()\}, \{() \mapsto 1\})}\ \text{E-tick} \qquad \frac{e \downarrow v}{\mathsf{return}\, e \Downarrow^{0} (\{v\}, \{v \mapsto 1\})}\ \text{E-ret}$$

$$\frac{\begin{array}{c} e_1 \downarrow v_1 \qquad v_1 \Downarrow^{\kappa_1} (V_1, M_1) \\ \forall a \in V_1.\ e_2[a/x] \downarrow v_{2,a} \wedge v_{2,a} \Downarrow^{\kappa_{2,a}} \mu_{2,a} \qquad \kappa_2 = \sum_{a \in V_1} (M_1\ a) \cdot \kappa_{2,a} \qquad \mu = \mu_1 \otimes a.\mu_{2,a} \end{array}}{\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_2 \Downarrow^{\kappa_1 + \kappa_2} \mathbb{M}_2(\mu)}\ \text{E-bind}$$

$$\frac{e \downarrow v}{\mathsf{store}\, e \Downarrow^{0} (\{v\}, \{v \mapsto 1\})}\ \text{E-store} \qquad \frac{e_1 \downarrow v_1 \qquad e_2[v_1/x] \downarrow v_2 \qquad v_2 \Downarrow^{\kappa} (V, M)}{\mathsf{release}\ x = e_1\ \mathsf{in}\ e_2 \Downarrow^{\kappa} (V, M)}\ \text{E-release}$$

Fig. 5. Forcing evaluation of monadic terms in pλ-amor (selected rules)

e_2. T-store and T-release are direct generalisations of the corresponding typing rules from λ-amor (section 2.2) to the probabilistic setting.

Finally, **T-toDist** takes as input a list of real numbers that sum to 1, and converts the list into a static distribution with indices ranging over the length of that list. The rule is useful for the verification of algorithms like RWM (section 4.2), which explicitly manipulate finite distributions.

Subtyping. pλ-amor supports subtyping, formalised as the judgment $\Psi; \Theta; \Delta \vdash \tau <: \tau'$ (τ is a subtype of τ'). Fig. 4 shows the two most important rules: **sub-potential** and **sub-coupling**. The sub-potential rule allows reducing stored potential. The sub-coupling rule subtypes a monadic type $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau$ to another monadic type that has a higher κ and a different μ' that is related to μ by some coupling. This subtyping rule allows the simplification of distributions annotating monadic types, which is useful for the verification of complicated examples, as illustrated in section 4. A similar rule occurs in the prior work [Avanzini et al. 2019], but that rule is only proved sound in a setup where static and dynamic distributions coincide, which is not the case in pλ-amor.

3.3 Dynamics

pλ-amor uses a call-by-name (CBN) semantics with two evaluation relations: *pure* and *forcing*. The pure evaluation, $e \downarrow v$, is a relation between an expression e and the value v that it evaluates to *without evaluating any monadic subterms*. This relation is standard, so we defer its details to the technical appendix.

The forcing evaluation, $e \Downarrow^{\kappa} \nu$, relates a monadic term of type $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa'\ \tau$ to a dynamic *distribution* ν over values of the type family $\tau(a)$, and a number κ, which is the expected runtime cost over that dynamic distribution. Selected rules of forcing evaluation are described in Fig. 5. Rule **E-ret** says that if e reduces to v in the pure evaluation then $\mathsf{return}\, e$ reduces to a point distribution over v in the forcing evaluation with 0 expected cost. **E-store** is similar due to the ghost nature of potentials. Note that, in both rules, the output distribution is defined over a set of runtime values, $\{v\}$, unlike the typing-rules T-ret and T-store where the distributions are defined over a set of static indices, $\{0\}$.

E-bind says that if e_1 evaluates (with forcing) to a dynamic distribution $\nu_1 = (V_1, M_1)$ with an expected cost κ_1, and for $a \in V_1$, $e_2[a/x]$ evaluates with an expected cost $\kappa_{2,a}$ to a distribution $\nu_{2,a}$, then the total expected cost of $\mathsf{bind}\ x = e_1\ \mathsf{in}\ e_2$ is $\kappa_1 + \sum_{a \in V_1} M_1(a) \cdot \kappa_{2,a}$ (the sum of κ_1 and the expectation of cost $\kappa_{2,a}$ taken over ν_1), and the resulting distribution is $\mathbb{M}_2(\nu_1 \otimes a.\nu_{2,a})$ (the second marginal of the convolution of ν_1 and $\nu_{2,a}$).

The expression $\text{release } x = e_1 \text{ in } e_2$ evaluates like a standard *let* expression with the exception that e_2 evaluates to a distribution (rule **E-release**). Finally, **E-tick** defines the semantics of the expression $\uparrow^{\kappa}$; the rule says that this expression adds κ units to the runtime cost and returns the point distribution over the unit type.

3.4 Relation between the Modalities and Soundness

There are two ways of describing costful probabilistic computations in pλ-amor. On one hand, we can use the monadic type, $\mathbb{PC}_{(a\leftarrow\mu)}\ \kappa\ \tau$, to describe a computation that yields a distribution coupled to μ, with expected cost no more than κ. On the other hand, we can use a function type, $([\kappa]\,\mathbf{1} \multimap \mathbb{PC}_{(a\leftarrow\mu)}\ 0\ \tau)$, to the same effect. A function of this type takes a potential of κ units as an argument and consumes it fully to produce a distribution coupled to μ. Conceptually, both these types describe the computation of distributions coupled to μ with expected cost no more than κ, so we might expect coercions between these types. In fact, we can prove a stronger result: These two types are *isomorphic*, i.e., $\mathbb{PC}_{(a\leftarrow\mu)}\ \kappa\ \tau \cong ([\kappa]\,\mathbf{1} \multimap \mathbb{PC}_{(a\leftarrow\mu)}\ 0\ \tau)$. The functions $(\lambda e.\lambda p.\text{release } y = p \text{ in } e)$ and $(\lambda f.\,\text{bind } x = \text{store}() \text{ in } f\ x)$ are coercions from $\mathbb{PC}_{(a\leftarrow\mu)}\ \kappa\ \tau$ to $([\kappa]\,\mathbf{1} \multimap \mathbb{PC}_{(a\leftarrow\mu)}\ 0\ \tau)$ and back, respectively. The two roundtrip compositions of these functions are identity functions. A proof of this fact can be found in the technical appendix.

Our end-to-end soundness theorem (Theorem 6) for expected costs is that certainly terminating computations (i.e., computations that terminate in all probabilistic branches, even those with 0 probability) of either of the two types above actually run with expected cost no more than κ. Note that our type system does not enforce termination: If a program does not certainly terminate, Theorem 6 holds vacuously. In particular, our type theory cannot be used to reason about programs that terminate in an almost sure sense [Bournez and Garnier 2005; McIver and Morgan 2005], so programs like the coupon collector and rejection samplers cannot be analysed using pλ-amor. Extending our type theory to handle variants of almost sure termination is an interesting direction of future work.

We prove Theorem 6 using a model of types that is described in section 3.5. Using the fundamental theorem of the model (Theorem 7), we can also prove that a computation of either type above outputs a distribution coupled to μ, but we elide this aspect here.

Theorem 6 (Soundness for expected costs)**.**
(1) If $\vdash e : \mathbb{PC}_{(a\leftarrow\mu)}\ \kappa\ \tau$ and $e \Downarrow_t^{\kappa'} v$ then $\kappa' \leq \kappa$.
(2) If $\vdash e : [\kappa]\,\mathbf{1} \multimap \mathbb{PC}_{(a\leftarrow\mu)}\ 0\ \tau$ and $e() \Downarrow_t^{\kappa'} v$ then $\kappa' \leq \kappa$.

3.5 Model of Types

To prove Theorem 6, we define a Kripke step-indexed logical relation model for pλ-amor's types. Our model extends the model of λ-amor [Rajani et al. 2021] by adding support for probabilistic computation. The key insights in the design of our model are the use of probabilistic coupling to relate static and dynamic distributions and our handling of potentials.

Step-indices [Ahmed 2004; Mitchell 1996], denoted T, are integral to our model but they are purely an artefact of the meta-theory. They avoid circularity arising from the impredicative type quantifier. We note that some prior work uses step-indexed logical relations to reason about equi-termination or may/must termination of probabilistic programs [Aguirre and Birkedal 2023; Wand et al. 2018]. In those settings, step-indices play a significant role in the model. In contrast, because we do not enforce termination and only give guarantees to programs that certainly terminate, our use of step-indices is fairly standard. Consequently, we do not discuss step indices in detail here.

We augment the evaluation relations, both pure and forcing, with an additional natural number T, which is the number of rules used in the derivation of the relation. The augmented judgments are written $e \downarrow_T v$ and $e \Downarrow_T^{\kappa} v$, respectively.

Fig. 6 shows selected cases of the definition of the semantic model, which consists of four relations. Next, we describe these relations.

Value Relation. The value interpretation of a type τ, denoted by $\mathcal{V}[\![\tau]\!]$, is a set of triples (p, T, v) containing a potential p, a step index T and a value v. The intuitive meaning of the triple is that v is (semantically) of type τ, and the expected potential stored with v and its subterms is no more than p. We describe a few cases of the value relation here, starting with some standard affine types constructors.

The interpretation of the tensor ($\otimes$) pair says that the available potential must be at least the sum of the potentials required for interpreting the two components. This is because both the components of a tensor pair can be used simultaneously by the context. The interpretation of the with (&) pair is very different. Since only one, but not both, of its components can be used by the context, we only need potential to cover each of the components separately.

The interpretation of the function type $\tau_1 \multimap \tau_2$ says that potential p is sufficient for $\lambda x.e$ at the type if for *any* substitution e' of the input type τ_1, which comes with its own potential p', the total potential $p + p'$ is sufficient to interpret the body $e[e'/x]$ at the result type τ_2.

Next, we describe the interpretation of $!\tau$. Recall that the affine type $!\tau$ means "any finite number of copies of τ". Hence, the potential needed to interpret $!e$ in $!\tau$ is ∞ if e at type τ needs any non-zero potential. However, if the potential needed for e at type τ is 0, then we can interpret $!e$ at type $!\tau$ with any potential.

The interpretation of the potential modality ($[n]\ \tau$) says that the potential p must be at least n *plus* the potential needed to interpret v at the type τ. The ghost nature of potential becomes explicit in this definition, as the same value v must occurs in the interpretations of $[n]\ \tau$ and τ.

In the interpretation of the monadic type $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau$, we have to account for the probability and cost effects simultaneously. In particular, we have to stipulate a relation between the dynamic distribution (over values) obtained at runtime and the static distribution (over indices) that occurs in the type. We achieve this by requiring that there be a *coupling* ρ relating the static and the dynamic distributions, such that: (a) for every pair of index term i and value v'_j in the support of the coupling, v'_j semantically types at $\tau[i/a]$ with some potential p_j, and (b) the sum of the expected runtime cost κ' and the expected value (over the coupling) of the potentials p_j is no more than the sum of the available potential p and the static cost κ in the monadic type. The sum $p + \kappa$ on the right hand side of the inequality in clause (b) justifies why it is sound to offset cost using available potential, as in the rule T-release.

Expression Relation. The expression interpretation, $\mathcal{E}[\![\tau]\!]$, is standard. A triple of the form (p, T, e) is in the expression relation at type τ if the value v obtained from the pure evaluation of e is in the value relation at the same type with the same potential (pure evaluation has no cost, so it does not consume potential).

Substitution Relations. The interpretations of the affine (Γ) and the non-affine context (Ω) define semantically-typed variable substitutions. The relation $\mathcal{G}[\![\Gamma]\!]$ for the affine context says that a substitution is semantically well-typed with a potential if the potential is enough to interpret each expression in the range of the substitution at the respective type provided by Γ. For the non-affine context, the interpretation $\mathcal{G}[\![\Omega]\!]$ is similar, except that the required potential can be finite only if all substituted expressions require a zero potential, else it must be ∞.

$$
\begin{array}{lcl}
\mathcal{V}\llbracket \tau_1 \otimes \tau_2 \rrbracket & \triangleq & \{(p, T, \langle\!\langle v_1, v_2 \rangle\!\rangle) \mid \exists p_1, p_2 . p_1 + p_2 \leq p \wedge (p_1, T, v_1) \in \mathcal{V}\llbracket \tau_1 \rrbracket \\
 & & \qquad\qquad\qquad\qquad\qquad \wedge\ (p_2, T, v_2) \in \mathcal{V}\llbracket \tau_2 \rrbracket\} \\
\mathcal{V}\llbracket \tau_1 \,\&\, \tau_2 \rrbracket & \triangleq & \{(p, T, \langle v_1, v_2 \rangle) \mid (p, T, v_1) \in \mathcal{V}\llbracket \tau_1 \rrbracket \wedge (p, T, v_2) \in \mathcal{V}\llbracket \tau_2 \rrbracket\} \\
\mathcal{V}\llbracket \tau_1 \multimap \tau_2 \rrbracket & \triangleq & \{(p, T, \lambda x.e) \mid \forall p', e', T' {<} T\ .(p', T', e') \in \mathcal{E}\llbracket \tau_1 \rrbracket \implies \\
 & & \qquad\qquad\qquad\qquad\qquad (p + p', T', e[e'/x]) \in \mathcal{E}\llbracket \tau_2 \rrbracket\} \\
\mathcal{V}\llbracket !\tau \rrbracket & \triangleq & \{(p, T, !e) \mid (0, T, e) \in \mathcal{E}\llbracket \tau \rrbracket\} \cup \\
 & & \{(\infty, T, !e) \mid \exists p' . p' > 0 \wedge (p', T, e) \in \mathcal{E}\llbracket \tau \rrbracket\} \\
\mathcal{V}\llbracket [n]\, \tau \rrbracket & \triangleq & \{(p, T, v) \mid \exists p' . p' + n \leq p \wedge (p', T, v) \in \mathcal{V}\llbracket \tau \rrbracket\}\} \\
\mathcal{V}\llbracket \mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau \rrbracket & \triangleq & \{(p, T, v) \mid \mu = (C_s, M_s) \wedge \forall \kappa', v', T' {<}\, T. \\
 & & \qquad v \Downarrow^{\kappa'}_{T'} (V_d, M_d) \wedge V_d = \{v'_1, \ldots, v'_{|V_d|-1}\} \implies \\
 & & \qquad \exists \rho : (C_s, M_s) \leftrightarrow (V_d, M_d).\ \exists p_0, \ldots, p_{|V_d|-1}. \\
 & & \qquad a)\ \forall (i, v'_j) \in C_s \times V_d .\ \rho(i, v'_j) \neq 0 \implies \\
 & & \qquad\qquad\qquad\qquad\qquad (p_j, T - T', v'_j) \in \mathcal{V}\llbracket \tau[i/a] \rrbracket \\
 & & \qquad b)\ \kappa' + \sum_{i \in C_s} \sum_{v'_j \in V_d} (p_j \cdot \rho(i, v'_j)) \leq p + \kappa\} \\
\mathcal{E}\llbracket \tau \rrbracket & \triangleq & \{(p, T, e) \mid \forall\ T' {<} T, v . e \downarrow_{T'} v \implies (p, T - T', v) \in \mathcal{V}\llbracket \tau \rrbracket\} \\
\mathcal{G}\llbracket \Gamma \rrbracket & \triangleq & \{(p, T, \rho_l) \mid \exists f : \mathcal{V}ars \to \mathbb{R}^+ .\ (\forall x \in \mathsf{dom}(\Gamma).\ (f(x), T, \rho_l(x)) \in \mathcal{E}\llbracket \Gamma(x) \rrbracket) \\
 & & \qquad\qquad\qquad\qquad\qquad \wedge\ (\sum_{x \in \mathsf{dom}(\Gamma)} f(x) \leq p)\} \\
\mathcal{G}\llbracket \Omega \rrbracket & \triangleq & \{(p, T, \rho_m) \mid \exists f : \mathcal{V}ars \to \mathbb{R}^+ .\ \forall x \in \mathsf{dom}(\Omega).\ (f(x), T, \rho_m(x)) \in \mathcal{E}\llbracket \Omega(x) \rrbracket \\
 & & \qquad\qquad\qquad\qquad\qquad \wedge\ (\exists x \in \mathsf{dom}(\Omega).\ f(x) > 0) \implies p = \infty\}
\end{array}
$$

Fig. 6. Model of pλ-amor types (selected cases)

Fundamental Theorem. Next, we state the standard fundamental theorem for our logical relation. The theorem says that syntactically well-typed open terms are in the semantic interpretations of their respective types, provided that the closing substitutions are semantically well-typed. The proof of the fundamental theorem proceeds by induction on the typing derivation. In the proof for the monadic type, we show the existence of a coupling between the static and runtime distributions in the cases, as required by the definition of the logical relation. To show the soundness of the sub-coupling rule, we rely on the fact that the composition of two couplings is also a coupling. The full proof of the fundamental theorem is in our technical appendix. The soundness theorem (Theorem 6) is a direct corollary of the fundamental theorem.

Theorem 7 (Fundamental theorem). $\Psi; \Theta; \Delta; \Omega; \Gamma \vdash e : \tau \wedge (p_l, T, \rho_l) \in \mathcal{G}\llbracket \Gamma\ \theta\ \iota \rrbracket \wedge (p_m, T, \rho_m) \in \mathcal{G}\llbracket \Omega\ \theta\ \iota \rrbracket \wedge . \models \Delta\ \iota \implies (p_l + p_m, T, e\ \rho_l\ \rho_m) \in \mathcal{E}\llbracket \tau\ \theta\ \iota \rrbracket$.

4 Examples

In this section, we present three example programs and establish cost bounds for them in pλ-amor. We begin with a simple example, which we call randomised response. This example is not based on online learning and it is meant to further illustrate the workings of pλ-amor. As our second example, we revisit the Randomised Weighted Majority algorithm from section 2.1. Our third example is the EXP3 algorithm for the multi-armed bandit problem [Auer et al. 2002] for which we derive a closed-form bound on the expected loss.

Our technical appendix includes an additional example – a bound on the expected regret of the stochastic gradient descent algorithm [Bansal and Gupta 2019; Robbins and Monro 1951].

```
1   $RR : \forall v : \mathbb{N}, v' : \mathbb{N}, k : \mathbb{N}\,.$
2             $!\mathbf{N}(v) \multimap\ !\mathbf{N}(v') \multimap\ !\mathbf{N}(k) \multimap \left[(2 - \frac{1}{2^{k-1}})\right] \mathbf{1} \multimap \mathbb{PC}_{(a \leftarrow \delta_0)}\ 0\ \mathbf{N}$
3                        where $\mathbf{N} \triangleq (\exists v'' : \mathbb{N}.\ \mathbf{N}(v''))$ and $\delta_0 \triangleq (\{0\}, (0 \mapsto 1))$.
4   fix $RR.\Lambda.\Lambda.\Lambda.\lambda\ vl\ vl'\ n\ p.$
5    let! $vl_u = vl$ in let! $vl'_u = vl'$ in let! $n_u = n$ in
6    matchN $n_u$
7       ,Z $\mapsto$ return (Ex $vl'_u$)
8       ,S $n'$ $\mapsto$
9          release _ = $p$ in bind _ = $\uparrow^1$ in
10         bind $x$ = Unif 1 in
11         matchN $x$
12            ,Z $\mapsto$ bind $p'$ = Store () in $RR$ $!vl_u$ $!vl'_u$ $!n'$ $p'$
13            ,S $x'$ $\mapsto$ return (Ex $vl_u$)
```

Fig. 7. Cost analysis of randomised response in pλ-amor

4.1 Randomised Response

Our first example, randomised response or *RR*, is shown in Fig. 7. *RR* is a recursive function which runs for at most k rounds. In each round, it flips a fair coin (line 11). If the outcome is tails (represented as 1; heads is 0) the function returns the value vl, otherwise it recurses. If k rounds are exhausted without any tails, then the function returns the other value vl'. Each recursive call incurs a unit cost, which is modelled with the $\uparrow^1$ operation on line 10. Using pλ-amor, we show that the expected cost of *RR* is bounded by $(2 - \frac{1}{2^{k-1}})$ (Theorem 8).

Theorem 8. The expected cost of *RR* (Fig. 7) is upper-bounded by $(2 - \frac{1}{2^{k-1}})$.

We start with a description of *RR*'s type. *RR* takes four arguments: the two possible result values vl and vl'; a natural number n of the singleton type $\mathbf{N}(k)$ (so, n represents the static index k); and a potential p of type $\left[(2 - \frac{1}{2^{k-1}})\right] \mathbf{1}$. The underlying type $\mathbf{1}$ in p's type means that p does carry a useful runtime value – p's only purpose is to provide potential. Upon completion, *RR* returns one of the two input values, so a value of type $\mathbf{N} \triangleq \exists v'' : \mathbb{N}.\ \mathbf{N}(v'')$. The overall output type of *RR* is the monadic type $\mathbb{PC}_{(a \leftarrow \delta_0)}\ 0\ (\exists v'' : \mathbb{N}.\ \mathbf{N}(v''))$. This type has 0 residual expected cost as the input potential p is sufficient to account for the expected runtime cost of *RR*. The output type also says that the final distribution is a point distribution over $\mathbf{N}$. This distribution is an *approximation* of *RR*'s actual output distribution, which is vl with probability $1 - 1/2^{k-1}$ and vl' with probability $1/2^{k-1}$. This type-level approximation is established using the sub-coupling subtyping rule.

When $n = 0$ (i.e., $k = 0$), the code returns immediately with 0 cost (line 7) and establishing the output type is trivial. When $n \neq 0$ (i.e., $k \neq 0$), *RR* uses the release construct on line 9 to release p's potential of $2 - \frac{1}{2^{k-1}}$ units. Of these, 1 unit is consumed immediately by $\uparrow^1$ later on the same line.

Hence, we have to show that the expected cost of lines 10–13 is at most $1 - \frac{1}{2^{k-1}}$. For this, we examine lines 10–13. Here, the code tosses a fair coin (line 10) and depending on the outcome, either returns immediately with 0 cost (line 13) or recurses with third parameter $n' : \mathbf{N}(k-1)$ instead of $n : \mathbf{N}(k)$ (line 12). The recursive call needs potential $2 - \frac{1}{2^{k-2}}$, which is formally passed via the parameter p'. Now, potential is stored in p' using the store construct, whose cost is exactly the

D0:

$$\dfrac{\dfrac{}{.;\Theta;\Delta;\Omega;\Gamma \vdash p : T_p} \qquad D1}{.;\Theta;\Delta;\Omega;\Gamma \vdash \mathsf{release}\ \_ = p\ \mathsf{in}\ E_1 : T_r}$$

D1:

$$\dfrac{\dfrac{\dfrac{}{.;\Theta;\Delta;\Omega;. \vdash \uparrow^1 : \mathbb{PC}_{(a\leftarrow\delta_0)}\ 1\ \mathbf{1}} \qquad D2}{.;\Theta;\Delta;\Omega;. \vdash \mathsf{bind}\ \_ = \uparrow^1\ \mathsf{in}\ E_2 : T_{r1}}}{.;\Theta;\Delta;\Omega;. \vdash E_1 : T_{r1}}$$

D2:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{.;\Theta;\Delta;\Omega;. \vdash \mathsf{Unif}\ 1 : \mathbb{PC}_{(a'\leftarrow\mu)}\ 0\ \mathbf{N}(a)} \qquad \vdots}{.;\Theta;\Delta;\Omega;. \vdash \mathsf{bind}\ x = \mathsf{Unif}\ 1\ \mathsf{in}\ \mathsf{matchN}\ldots : T_{r2'}}}{.;\Theta;\Delta;\Omega;. \vdash \mathsf{bind}\ x = \mathsf{Unif}\ 1\ \mathsf{in}\ \mathsf{matchN}\ldots : T_{r2}}\ \text{sub-coupling}}{.;\Theta;\Delta;\Omega;. \vdash E_2 : T_{r2}}$$

$$\begin{aligned}
\Theta &\triangleq v : \mathbb{N}, v' : \mathbb{N}, k : \mathbb{N}, k' : \mathbb{N}\\
\Delta &\triangleq k' + 1 = k\\
\Gamma &\triangleq p : T_p\\
\Omega &\triangleq RR : T_0, vl_u : \mathbf{N}(v),\\
&\quad\ vl'_u : \mathbf{N}(v'), n_u : \mathbf{N}(k)\\
&\quad\ n' : \mathbf{N}(k')\\
T_r &\triangleq \mathbb{PC}_{(a\leftarrow\delta_0)}\ 0\ \mathbf{N}\\
T_{r1} &\triangleq \mathbb{PC}_{(b\leftarrow\delta_0)}\ (2 - \tfrac{1}{2^{k-1}})\ \mathbf{N}\\
T_{r2} &\triangleq \mathbb{PC}_{(c\leftarrow\delta_0)}\ (1 - \tfrac{1}{2^{k-1}})\ \mathbf{N}\\
T_{r2'} &\triangleq \mathbb{PC}_{(c\leftarrow\mu')}\ (1 - \tfrac{1}{2^{k-1}})\ \mathbf{N}\\
T_{r3} &\triangleq \mathbb{PC}_{(d\leftarrow\delta_0)}\ \left(2 - \tfrac{1}{2^{k-2}}\right)\ \mathbf{N}\\
T_{r3'} &\triangleq \mathbb{PC}_{(d\leftarrow\delta_0\otimes\delta_0)}\ \left(2 - \tfrac{1}{2^{k-2}}\right)\ \mathbf{N}\\
\mu &\triangleq \left(\{0,1\}, (0 \mapsto \tfrac{1}{2}, 1 \mapsto \tfrac{1}{2})\right)\\
\mu' &\triangleq \mu \otimes \delta_0
\end{aligned}$$

D3:

$$\dfrac{\dfrac{D4 \qquad \vdots}{.;\Theta;\Delta,\ldots;\Omega;. \vdash \mathsf{bind}\ p' = \mathsf{Store}\ ()\ \mathsf{in}\ RR\ !vl_u\ !vl'_u\ !n'\ p' : T_{r3'}}}{.;\Theta;\Delta,\ldots;\Omega;. \vdash \mathsf{bind}\ p' = \mathsf{Store}\ ()\ \mathsf{in}\ RR\ !vl_u\ !vl'_u\ !n'\ p' : T_{r3}}\ \text{sub-coupling}$$

D4:

$$\dfrac{}{.;\Theta;\Delta,\ldots;\Omega;. \vdash \mathsf{Store}\ () : \mathbb{PC}_{(\_\leftarrow\delta_0)}\left(2 - \dfrac{1}{2^{k-2}}\right)\left(\left[\!\left[\left(2 - \dfrac{1}{2^{k-2}}\right)\right]\!\right]\mathbf{1}\right)}$$

Fig. 8. Derivation snippet

potential stored, i.e., $1 - \frac{1}{2^{k-2}}$. Hence, the expected cost of lines 10–13 is $\frac{1}{2}\cdot 0 + \frac{1}{2}\cdot\left(2 - \frac{1}{2^{k-2}}\right) = 1 - \frac{1}{2^{k-1}}$, as needed.

We show the formal typing derivations of some of the key steps of this reasoning in Fig. 8. Derivation D0 of the release construct from line 9 gives 0 costs to the release construct (in type T_r) after checking (via derivation D1) that the potential in p, i.e., $(2 - \frac{1}{2^{k-1}})$ units, equals the cost of the continuation, which is denoted E_1. $E_1 \triangleq \mathsf{bind}\ \_ = \uparrow^1\ \mathsf{in}\ E_2$.

E_1's type derivation is $D1$. The overall cost of E_1, reflected in E_1's type, T_{r1}, is $(2 - \frac{1}{2^{k-1}})$ units as explained above. Of this, cost 1 is incurred by the $\uparrow^1$ construct as shown in the derivation D1, and the rest $(1 - \frac{1}{2^{k-1}})$ is incurred by the continuation of $\uparrow^1$, which is denoted E_2. $E_2 \triangleq \mathsf{bind}\ x = \mathsf{Unif}\ 1\ \mathsf{in}\ \mathsf{matchN}\ldots$.[2]

E_2's cost, $1 - \frac{1}{2^{k-1}}$, is reflected in its type T_{r2}. This type is established in derivation D2. E_2 tosses a coin and branches on the outcome x using the bind construct. Following the rule T-bind, D2 establishes that the expected cost of the two branches equals $1 - \frac{1}{2^{k-1}}$. This reasoning proceeds as

[2] Unif 1 samples uniformly from $\{0, 1\}$. It is defined using the toDist construct.

explained earlier: The $x = 0$ case has cost $2 - \frac{1}{2^{k-2}}$ as established in derivation D3, while the case $x = 1$ has cost 0 (derivation not shown here). Their average is $1 - \frac{1}{2^{k-1}}$, as required.

Finally, we explain the use of the subtyping rule sub-coupling in this example. This rule is used to type the two bind expressions on lines 9 and 12. We start with the bind on line 12, which is typed in derivation D3. Here, using the T-bind rule, we know that the distribution of the bind expression will have the static distribution $\delta_0 \otimes \_.\delta_0$, as in the type $T_{r3'}$. Using the fact that there is a trivial coupling between $\delta_0 \otimes \_.\delta_0$ and δ_0, we can give the bind expression the simpler type T_{r2}, which has the static distribution δ_0. A similar use of sub-coupling occurs in derivation D2 in typing the bind on line 9. In this case, we replace the distribution $\mu \otimes \_.\delta_0$ with δ_0, relying on the fact that every distribution is coupled to δ_0. We find such simplifications of the static distributions useful in typing later examples as well.

4.2 Randomised Weighted Majority from Section 2.1, Revisited

We revisit our example from section 2.1. We describe how pλ-amor can be used to formally establish the key step in the proof of Theorem 1, namely, that the expected cost of an iteration of *rwm* is upper bounded by the change in potential across that iteration. Fig. 9 shows a re-encoding of *rwm*, this time in pλ-amor's syntax. We describe the changes relative to the earlier encoding of *rwm* in Fig. 1, and focus on cost-related changes as the other changes due to affineness and refinements are routine.

First, we describe type-level changes. The revised type of *rwm* has three key aspects: 1) The type carries a condition on the weights of the experts, namely, that each weight $\geq 1/(1-\eta)^r$. As mentioned earlier, this constraint ensures that the potential (described next) remains non-negative; 2) The *rwm* function has an additional argument – the potential – whose type depends on the weight of the experts. Specifically, *rwm* requires an input potential of $(log(\sum_{i \leq n} w\ i))/\eta$ units as we saw in section 2.1; and 3) The output type includes the remaining potential $(log\ (\sum_{i \leq n} w'\ i))/\eta$, where $w'\ i$ is the revised weight of the i^{th} expert after the round. Note that, the output type is monadic with 0 cost as the cost of *rwm* is covered entirely by its potential. The output type's distribution is a point distribution, which is obtained via the sub-coupling rule.

The type for *makePred* follows the same idea and its revised type also has potentials. On the other hand, *chgWts* is a pure function so its revised type does not have potentials. However, the revised type encodes the invariant that our condition on weights, point 1 above, holds.

At the code level, the key changes in the *rwm* function pertain to potentials. If the number of remaining rounds is zero, then the input potential is attached to the nil using the swap function (line 10), wrapped in an existential and returned. Otherwise, the input potential, which equals $(log(\phi(t)))/\eta$, is the split into two parts (line 13): (i) *here*, which is passed to the *makePred* function on line 14 and consumed in the current round; and (ii) *later*, which is used recursively on line 17. The potential *here* equals the expected loss of the current round ($\mathbb{E}[loss(t)]$, as required by Theorem 1) whereas *later* equals the potential for the next round ($(log(\phi(t+1)))/\eta$). As shown in equation (1) in section 2.1, the sum of *here* and *later* is no more than the input potential.

The *makePred* function works as follows. First, on line 12, it releases the input potential that was passed by *rwm*. Second, on line 13, it samples an expert. Next, it looks up the advice of the sampled expert on line 14 and inserts a $\uparrow$ to incur a cost equal to the expected loss of the current round (line 15). This cost is 1 if the chosen expert's advice is different from the actual answer and 0 otherwise. Finally, line 16 returns the chosen prediction wrapped in an existential to match the boolean return type $\mathbf{B} \triangleq \exists b.\mathbf{B}(b)$.

```
$rwm : \forall r : \mathbb{N}, n : \mathbb{N}, adv : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{B}, \eta : \mathbb{R}^{+}, ans : \mathbb{N} \rightarrow \mathbb{B}\ .$
        $!\mathbf{N}(r) \multimap\ !L_{i<n}(\mathbf{N}(i) \multimap \mathbf{B}(adv\ \ r\ \ i)) \multimap\ !\mathbf{R}(\eta)$
    $\multimap \forall w : \mathbb{N} \rightarrow \mathbb{R}^{+}.\ !L_{i<n}\ \ (w\ \ i \geq 1/(1-\eta)^{r}\ \ \&\ \ \mathbf{R}(w\ \ i))$
    $\multimap \left[log\ (\sum_{i\leq n} w\ \ i)/\eta\right] \mathbf{1}$
    $\multimap \mathbb{PC}_{(a\leftarrow\delta_0)}\ 0\ \ (\exists w' : \mathbb{N} \rightarrow \mathbb{R}^{+}.\ \left[log\ (\sum_{i\leq n} w'\ \ i)/\eta\right] (L_{i<r}\mathbf{B}))$
fix $rwm.\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda$ $ro$ $exp$ $eta.\Lambda.\lambda$ $p$ $wts$.
⋮
matchN $round_u$
 ,Z ↦
  let⟪$x, y$⟫ = Swap ($p$, nil) in return (Ex $y$)
 ,S $rnd$ ↦
  let! $advs_u = getAdvice$ {} $!round_u$ $!experts_u$ in
  let ⟪$here, later$⟫ = Split $p$ in
  bind $pr = makePred$ {} {} {} {} {} {} {} $here$ $!advs_u$ $!wts_u$ in
  let! $ans_u = getAnswer$ {} {} $!round_u$ in
  letEx $nw = chgWts$ {} {} {} $!wts_u$ $!advs_u$ $!ans_u$ $!eta_u$ in
  bind $rec = rwm$ {} {} {} {} $!rnd$ $!experts_u$ $!eta_u$ {} $nw$ $later$ in
  letEx $rec' = rec$ in
  bind $r =$ (release $prs = rec'$ in Store $(pr :: prs)$) in
  return $(Ex\ \ r)$
```

```
$makePred : \forall n : \mathbb{N}, adv : \mathbb{N} \rightarrow \mathbb{B}, \eta : \mathbb{R}^{+},$
        $ans : \mathbb{B}, w : \mathbb{N} \rightarrow \mathbb{R}^{+}.$
        $\forall p : \mathbb{R}^{+}.(p \geq EL) \Rightarrow [p]\,\mathbf{1}$
        $\multimap L_{i<n}\ \ \mathbf{B}(adv\ \ i) \multimap L_{i<n}\ \ \mathbf{R}(w\ \ i)$
        $\multimap \mathbb{PC}_{(a\leftarrow\delta_0)}\ \ 0\ \ (\mathbf{B})$
$makePred \triangleq$
$\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda$ $po$ $advs$ $wts$.
let! $wts_u = wts$ in
letEx $phi = sum$ $!wts_u$ in
let! $phi_u = phi$ in
let! $prob_u = map$ $!(\lambda x.x/phi_u)$ $!wts_u$ in
release _ = $po$ in
bind $x$ = toDist $prob_u$ in
  let $pred = lookup$ $x$ $advs$ in
  bind _ = $\uparrow^{(loss\ \ (adv\ \ x')\ \ ans)}$ in
    return $(Ex\ \ pred)$
```

where,
$EL \triangleq \sum_{i<n}(prob\ i) \cdot (loss\ (adv\ i)\ (ans))$
$prob \triangleq \lambda i.(w\ i)/\phi$
$loss \triangleq \lambda fs.$if $f ==\ s$ then 0 else 1
$\phi \triangleq \sum_{i<n} w\ i$

```
$chgWts : \forall n, r, \eta, a', adv : \mathbb{N} \rightarrow \mathbb{B}, w : \mathbb{N} \rightarrow \mathbb{R}.$
        $!L_{i<n}(w\ \ i \geq 1/(1-\eta)^{r}\ \ \&\ \ \mathbf{R}(w\ \ i))$
        $\multimap !L_{i<n}\mathbf{B}(adv\ \ i) \multimap !\mathbf{B}(a') \multimap !\mathbf{R}(\eta)$
        $\multimap \exists w' : \mathbb{N} \rightarrow \mathbb{R}^{+}.$
        $L_{i<n}(w'\ i \geq 1/(1-\eta)^{r-1}\ \&\ \mathbf{R}(w'\ i))$
$chgWts \triangleq$
$\Lambda.\Lambda.\Lambda.\Lambda.\Lambda.\lambda$ $weights$ $advs$ $ans$ $eta$.
let! $wts_u = weights$ in let! $advs_u = advs$ in
let! $ans_u = ans$ in let! $eta_u = eta$ in
match $!wts_u$ with
,nil ↦ Ex Λ. nil
,$wt :: wts$ ↦
 let $wt' = wt$ in
 match $!advs_u$ with
 ,nil ↦ fix$x.x$
 ,$a :: as$ ↦
   letEx $r = chgWts$ {} {} {} {} {} {}
                $!wts$ $!as$ $!ans_u$ $!eta_u$ in
    if $a == ans_u$
    then $Ex\ \ (\Lambda.wt' :: r)$
    else $Ex\ \ (\Lambda.wt' * (1 - eta') :: r)$
```

Fig. 9. Representation of the RWM algorithm of Fig. 1 in pλ-amor

```
Initialisation:
 $n > 0, T > 0,\ \gamma \in (0,1)$
 $w_i(0) = e^{\frac{T\gamma}{n}}$ for $i = 0, \dots, (n-1)$

For each $t = 0, \dots, T-1$
1. Set for $i = 1, \dots, n$
      $p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=0}^{n-1} w_j(t)} + \frac{\gamma}{n}$
2. Sample $i_t$ randomly from the distribution $\{i \mapsto p_i(t)\}_{i=0}^{n-1}$
3. Receive reward $x_{i_t} \in [0,1]$ (Loss $L_{i_t} = 1 - x_{i_t}$)
4. For $j = 0, \dots, (n-1)$:
5.    - if $j = i_t$, then $\hat{L}_j(t) = L_j(t)$, else $\hat{L}_j(t) = 0$
6.    - $w_j(t+1) = w_j(t) \cdot e^{\left(-\frac{\gamma}{n} \cdot \hat{L}_j(t)\right)}$
```

Fig. 10. An instance of the EXP3 algorithm in pseudocode reproduced from [Auer et al. 2002]

As in the *RR* example of section 4.1, we use the sub-coupling rule to simplify probability distributions in monadic types to the point distribution eagerly. Our technical appendix shows the full typing derivation.

4.3 Multi-Armed Bandit

As our last example, we prove an upper bound on the expected loss of a variant of the EXP3 algorithm [Auer et al. 2002] for the *multi-armed bandit* problem. Multi-armed bandit [Cesa-Bianchi and Lugosi 2006] is a classic reinforcement learning problem where an agent, historically called the bandit, has to choose from a set of available decisions, historically called the arms, over a series of rounds. In each round, each arm has an associated distribution over rewards (rewards range from 0 to 1), but these distributions are not known to the agent. After each round, the agent receives a single reward value that is sampled from the distribution of the arm it chose. The agent's goal is to maximise the expected reward over a fixed number of rounds. The multi-armed bandit problem exemplifies a fundamental tension between the exploration of new arms and the exploitation of the arm that has been the most rewarding so far. The tension arises because the distributions associated with the arms may evolve over rounds in an uncertain way.

Both the *experts'* problem that *rwm* solves and the *multi-armed bandit* problem require decision-making under uncertainty. However, they differ in the amount of feedback the agent receives. In the experts' problem, the agent obtains the advice of all experts and, hence, knows the losses of all the experts at the end of each round. This is called the full-information setting. In the multi-armed bandit problem, the agent obtains the reward for only its chosen arm but not the other arms. This is called the partial-information setting.[3]

In the rest of this section, we model and analyse a variant of the EXP3 algorithm [Auer et al. 2002], a popular approach for solving the multi-armed bandit problem. The key idea behind the EXP3 algorithm is to balance the **EXP**loration and **EXP**loitation trade-off using an **EXP**onentially weighted sampling, as explained next. The algorithm, shown in pseudocode in Fig. 10, is parameterised by the number of arms n, the time horizon or the total number of rounds T, the learning rate or the discount factor γ (a positive real between 0 and 1) and weights for all the arms (positive real

[3] We prove a bound on the expected loss instead of expected rewards for the multi-armed bandit problem for consistency with our analysis of *rwm*. However, we believe that a similar analysis can be done with rewards as well.

numbers initialised to $e^{\frac{T\gamma}{n}}$). In each round t, where $t = 0 \ldots (T-1)$, the algorithm begins creates a probability distribution over the arms. The probability of each arm (line 1) has two summands. The first summand is a distribution obtained from arm weights as in the RWM example, but discounted by a factor $(1-\gamma)$. The second summand is a uniform distribution over the arms discounted by the factor γ. The two summands are the relative probabilities of choosing an arm for exploitation and exploration, respectively.

Next, the algorithm samples an arm i_t according to the probability distribution over the arms (line 2) and receives a reward x_{i_t} for the chosen arm from an unspecified oracle (line 3). The corresponding loss, $L_{i_t} \in (0,1)$, is $L_{i_t} = 1 - x_{i_t}$. On lines 4–6, the algorithm creates a loss vector, denoted by $[\hat{L}_j(t)]_{j=0}^{n-1}$, for the arms. The chosen arm $j = i_t$ has loss L_{i_t} while all other arms are assumed to have 0 loss. Finally, on line 6, the algorithm updates the weights of the arms using multiplicative factors that are exponential in the arms' respective losses. As a result, the weight of the chosen arm, i_t, reduces by a factor of $e^{\frac{-\gamma}{n}\hat{L}_{i_t}}$, while the weights of all other arms remain unchanged.

Our goal is to obtain a bound on the total expected loss of the above algorithm and prove that bound in pλ-amor. In particular, we prove the following closed-form bound on the expected loss (Theorem 9).

Theorem 9. The expected loss of the algorithm in Fig. 10 over T rounds is upper-bounded by $\frac{(1-\gamma)}{\gamma} \cdot \frac{n^2 log(n)}{(n-\gamma)} + \left(\frac{(n+\gamma)(1-\gamma)}{(n-\gamma)} + 1\right) \cdot T$.

As we did for *rwm*, we prove this bound by finding a suitable potential function, $\Psi(t)$, and showing that for any given round $t \in [0, T-1]$, $\mathbb{E}[loss(t)] \leq \Psi(t) - \Psi(t+1)$. From the linearity of expectations, we get that the expectation of total loss over the T rounds, say E, satisfies $E \leq \Psi(0) - \Psi(T)$. We then prove $\Psi(T) \geq 0$ and, hence, $E \leq \Psi(0)$. Finally, we show that $\Psi(0)$ equals the bound in Theorem 9.

The difficult step in this proof is finding a suitable potential function $\Psi(t)$ and proving the bound $\mathbb{E}[loss(t)] \leq \Psi(t) - \Psi(t+1)$. In our technical appendix, we prove that the following potential function satisfies this property. The proof relies on basic algebraic properties like $1 - x \leq e^{-x} \leq 1 - x + \frac{1}{2}x^2$ for $x \geq 0$.

$$\begin{array}{l} \Psi(t) \triangleq \frac{log\ \phi(t)}{\frac{\gamma}{n}\left(1-\frac{\gamma}{n}\right)\frac{1}{1-\gamma}} + (T-t) \cdot \left(\frac{\gamma(1-\gamma)}{(n-\gamma)} + 1\right) \\ \text{where } \phi(t) \triangleq \sum_{i=0}^{n-1} w_i(t) \end{array}$$

We note that $w_i(0) = e^{\frac{T\gamma}{n}}$ from the algorithm's initialisation step and every weight reduces by a factor of at most $e^{\frac{\gamma}{n}}$ in each round, so all weights remain ≥ 1 in the first T rounds. Hence, $log\ \phi(t) \geq 0$ for $t \leq T$. From this, it is easy to check that $\Psi(t) \geq 0$ for $t \leq T$. Finally, when $t = 0$, $\phi(t) = n \cdot e^{\frac{T\gamma}{n}}$. Substituting this in the definition of $\Psi(t)$ above, we easily check that $\Psi(0)$ equals the bound in Theorem 9.

To prove the bound on expected loss formally, we encode the algorithm in pλ-amor as shown in Fig. 11. We only describe the parts of the encoding that pertain to costs and potentials. The type of the *bandit* function is parameterised by a reward oracle $reward : \mathbb{N} \to \mathbb{N} \to \mathbb{R}^{+}$, which provides the reward for a given round and a given arm. As discussed above, *bandit* requires $\Psi(T-r)$ units of potential as input, where r is the number of rounds *remaining* (in the notation above, r would be $T-t$). Since this potential is no less than the total expected loss over r rounds, the monadic output type of *bandit* has 0 expected cost. Upon terminating, *bandit* returns a distribution over the arms chosen in each round. As in the *rwm* example, we statically approximate this distribution with a point distribution using the sub-coupling rule.

```
1   $bandit : \forall r : \mathbb{N}, n : \mathbb{N}, \gamma : \mathbb{R}^+, reward : \mathbb{N} \to \mathbb{N} \to \mathbb{R}^+.\ \ !\mathbf{N}(r)\ \ \multimap !\mathbf{N}(n)\ \ \multimap !\mathbf{R}(\gamma)$
2          $\multimap \forall w : \mathbb{N} \to \mathbb{R}^+.\ \ !L_{i<n}\ \ (w\ \ i \geq e^{\frac{r\gamma}{n}}\ \ \&\ \ \mathbf{R}(w\ \ i))$
3          $\multimap \left[ \frac{log\ \ (\sum_{i<n} w\ \ i)}{\frac{\gamma}{n}(1-\frac{\gamma}{n})\frac{1}{1-\gamma}} + r \cdot \left( \frac{\gamma(1-\gamma)}{(n-\gamma)} + 1 \right) \right] \mathbf{1}$
4          $\multimap \mathbb{PC}_{(a \leftarrow \delta_0)}\ \ 0\ \ (L_{i<r}\mathbf{N})$
5   fix  $bandit$.
6     $\Lambda.\Lambda.\Lambda.\Lambda.\lambda\ \ round\ \ arms\ \ g.\Lambda.\lambda\ \ wts\ \ p.$
7     let!  $round_u = round$  in  let!  $arms_u = arms$  in
8     let!  $g_u = g$  in  let!  $wts_u = wts$  in
9       matchN!  $!round_u$
10        , Z $\mapsto$ return  nil
11        , S  $rnd \mapsto$
12        letEx  $phi = sum\ \ !wts_u$  in  let!  $phi_u = phi$  in
13        let!  $prob_u = map\ \ !(\lambda x.$let  $y = x$  in  $(1 - g_u)/phi_u * y + g_u/arms_u)\ \ !wts_u$  in
14        release  $- = p$  in
15          bind  $cArm$ =
16            bind  $chArm$ = toDist  $!prob_u$  in  let!  $chArm_u = chArm$  in
17              bind  _ = $\uparrow^{1-(reward\ r\ ca)}$  in  return  $(Ex\ \ !chArm_u)$
18          in
19            letEx  $cArm' = cArm$  in  let!  $cArm_u = cArm'$  in
20            let!  $chReward = getReward\ \{\}\ \{\}\ \{\}\ !round_u\ !cArm_u$ in
21            letEx $lvs = prepareLossVector\ \{\}\ \{\}\ \{\}\ \{\}\ ![1..arms_u]\ !prob_u\ !cArm_u\ !chReward$ in
22            letEx $nw' = changeWeights\ \{\}\ \{\}\ \{\}\ \{\}\ \{\}\ !wts_u\ lvs\ !arms_u\ !g_u$ in
23              bind $later$ = store () in
24                bind $rec = bandit\ \{\}\ \{\}\ \{\}\ \{\}\ !rnd\ !arms_u\ !g_u\ \{\}\ nw\ later$ in
25                return $(Ex\ cArm_u :: rec)$
```

Fig. 11. Cost analysis of the EXP3 algorithm from Fig. 10 in pλ-amor

The body of *bandit* releases the input potential on line 14 and uses it to account for the expected loss of the current round and for the loss of the recursive call. The expected loss of the current round is modelled using the $\uparrow$ construct on line 17. The functions *prepareLossVector* and *changeWeights*, whose definitions we defer to our technical appendix, encode the weight update logic from lines 4–6 of the pseudo-code of Fig. 10. Finally, the store construct on line 23 stores $\Psi(r-1)$ units of potential, which is passed to the recursive call on line 24.

5 Pλ-amor$^{\mathbf{C}}$: An Extension to a Graded ! Modality

So far, we have considered two graded modalities, $\mathbb{PC}_{(a \leftarrow \mu)}\ \kappa\ \tau$ and $[p]\ \tau$, and an ungraded comonadic modality, $!\tau$. Prior proposals such as λ-amor [Rajani et al. 2021], $d\ell$PCF [Dal Lago and Gaboardi 2011] and ℓRPCF [Avanzini et al. 2019] have considered graded variants of $!\tau$ in the context of cost analysis of deterministic and probabilistic programs to ! track the number of times a variable is used, as this increases the expressiveness of the type theory. A natural question is whether pλ-amor can be extended to a graded ! modality. In this section, we answer this question in the affirmative.

To the best of our knowledge, graded ! modalities were first introduced in Bounded Linear Logic (BLL) [Girard et al. 1992]. BLL included the modality $!_{i<n}\tau$, which is morally equivalent to the iterated tensor $\tau[0/i] \otimes \tau[1/i] \ldots \otimes \tau[(n-1)/i]$ (i is bound locally in $!_{i<n}\tau$). The term n is called the multiplicity or the coffect of the modality [Petricek et al. 2013]. This grading of ! has two features: (a) It tracks the number of times, n, that the expression may be used, and (b) Each copy of the expression can have a different type; the j^{th} copy has type $\tau[j/i]$.

In this section, we extend pλ-amor to a new type theory pλ-amor$^{\mathsf{C}}$ by grading the ! modality of pλ-amor in a similar manner. The differences between pλ-amor and pλ-amor$^{\mathsf{C}}$ are limited to the type system and the model of types.

Due to space restrictions, we only present a brief sketch of pλ-amor$^{\mathsf{C}}$ here. The full development is included in our technical appendix, which also covers support for sub-distributions and a type-preserving embedding of ℓRPCF [Avanzini et al. 2019], which is a previously proposed coeffect-based system for the analysis of expected execution time. The embedding show that pλ-amor$^{\mathsf{C}}$ is at least as expressive as ℓRPCF.

Changes to the Types and Type System. To support graded comonads, we grade the ! modality of pλ-amor with a coeffect generalising that of BLL (as mentioned above). Our graded comonad is written $!_{\sum_{a \in S} R_a} \tau(a)$ and can be thought of as R_a copies of the type $\tau(a)$ for *every* $a \in S$, i.e., $\otimes_{a \in S} \ (\tau(a))^{R_a}$.

In the typing judgment, we annotate every variable in the non-affine context Ω with its multiplicity, as in $x :_{\sum_{a \in S} R_a} \tau$, where τ and R_a may contain a free. The juxtaposition of two Ωs, written $\Omega_1 \oplus \Omega_2$ is nontrivial now as we have to add multiplicities and take the union of the two Ses. We defer the formal definition of context juxtaposition to the technical appendix.

Typing Rules. The following typing rules change: the sub-exponential's introduction and elimination rules (**T-subExpI** and **T-subExpE**), the fixpoint constructor (**T-fix**), the variable rules of the non-affine context (**T-var2**) and the bind rule (**T-bind**). The revised rules are listed in Fig. 12.

T-subexpI says that if e has type τ for every $a \in S$ using the resources in the non-affine context Ω (which may depend on a) and the empty affine context, then for $\Sigma_{a \in S} R_a$ copies of e we will need $\Sigma_{a \in S} R_a$ times the resources in Ω. The elimination rule, **T-subexpE**, says that eliminating a term of type $!_{\sum_{a \in S} R_a} \tau$ introduces a variable binding, $x :_{\sum_{a \in S} R_a} \tau$, in the continuation's context.

T-bind says that if the continuation term e_2 can be typed using resources from $\Omega'(a)$ (for every a in the domain of μ_1), then the whole bind term can be type-checked using resources from $\Omega \oplus \left(\sum_{a \in \pi_1(\mu_1)} \pi_2(\mu_1)(a)\right) \cdot \Omega'$. Here, $\left(\sum_{a \in \pi_1(\mu_1)} \pi_2(\mu_1)(a)\right) \cdot \Omega'$ computes the average (over μ_1) resources required by the continuation.

In the fixpoint rule, **T-fix**, the parameter b denotes a generic node in the recursion tree, S is an index set that ranges over the children of b, C_a is the a^{th} child of b, and N_a is the number of times the b^{th} node uses the value returned by its a^{th} child (S, C_a and N_a may depend on b). The rule is the same as ℓRPCF's [Avanzini et al. 2019] fixpoint rule. However, in ℓRPCF, this rule is only proved sound with respect to a fixed cost model (one unit of cost for every elimination construct) and, hence, can only be used to bound the expected running time of programs. In pλ-amor$^{\mathsf{C}}$ we are not restricted to a fixed cost model, and allow the programmer to specify the cost model using the tick construct. ℓRPCF's reasoning principle for fixpoints carries over to our more general setting.

Finally, **T-var2** says that the J^{th} variant of a non-affine variable can be used only when J is in the index set of instances (indicated by $J \in S$) and $R_J \geq 1$.

Interaction between the Graded ! and the Potential Modalities. We describe the interaction between the graded ! modality and the potential modality using two distributive laws [Gaboardi et al. 2016],

$$\dfrac{\Theta, \Delta \models R_J \geq 1 \qquad \Theta, \Delta \models J \in S}{\Psi; \Theta; \Delta; \Omega, x :_{\sum_{a \in S} R_a} \tau; . \vdash x : \tau[J/a]}\ \text{T-var2}$$

$$\dfrac{\Psi; \Theta, a; \Delta, a \in S; \Omega; . \vdash e : \tau \qquad \Theta, a; \Delta, a \in S \vdash R_a : \mathbf{R}^+}{\Psi; \Theta; \Delta; \sum\limits_{a \in S} R_a \cdot \Omega; . \vdash\ !e : !_{\sum_{a \in S} R_a} \tau}\ \text{T-subExpI}$$

$$\dfrac{\Psi; \Theta; \Delta; \Omega_1; \Gamma_1 \vdash e : !_{\sum_{a \in S} R_a} \tau \qquad \Psi; \Theta; \Delta; \Omega_2, x :_{\sum_{a \in S} R_a} \tau; \Gamma_2 \vdash e' : \tau'}{\Psi; \Theta; \Delta; \Omega_1 \oplus \Omega_2; \Gamma_1 \oplus \Gamma_2 \vdash \mathsf{let}\, !\, x = e \;\mathsf{in}\; e' : \tau'}\ \text{T-subExpE}$$

$$\dfrac{\Psi; \Theta, b; \Delta; x :_{\sum_{a \in S} N_a} \tau[C_a/b]; . \vdash e : \tau}{\Psi; \Theta; \Delta; \Omega; \Gamma \vdash \mathsf{fix}\; x.e : \tau[J/b]}\ \text{T-fix}$$

$$\dfrac{\begin{array}{c} \Psi; \Theta; \Delta; \Omega; \Gamma_1 \vdash e_1 : \mathbb{PC}_{(a \leftarrow \mu_1)}\ \kappa_1\ \tau_1 \\ \Psi; \Theta, a; \Delta; \Omega'; \Gamma_2, x : \tau_1 \vdash e_2 : \mathbb{PC}_{(b \leftarrow \mu_2)}\ \kappa_2\ \tau_2 \qquad \kappa \geq \kappa_1 + \sum\limits_{a \in (\pi_1(\mu_1))} (\pi_2(\mu_1)(a)) \cdot \kappa_2(a) \\ \mu = \mu_1 \otimes a.\mu_2 \qquad \Psi; \Theta, c; \Delta, c \in \pi_1(\mu) \vdash \tau_2[\pi_1(c)/a][\pi_2(c)/b] <: \tau \end{array}}{\Psi; \Theta; \Delta; \Omega \oplus \left(\sum\limits_{a \in \pi_1(\mu_1)} \pi_2(\mu_1)(a) \right) \cdot \Omega'; \Gamma_1 \oplus \Gamma_2 \vdash \mathsf{bind}\ \ x = e_1 \;\mathsf{in}\; e_2 : \mathbb{PC}_{(c \leftarrow \mu)}\ \kappa\ \tau}\ \text{T-bind}$$

$$\dfrac{\Psi; \Theta, a; \Delta, a \in I \vdash \tau <: \tau'}{\Psi; \Theta; \Delta \vdash \left[\sum\limits_{a \in I} R_a \cdot P_a \right] !_{\sum_{a \in I} R_a} \tau \ \ <: !_{\sum_{a \in I} R_a} \left[P_a \right] \tau'}\ \text{sub-BP1}$$

$$\dfrac{\Psi; \Theta, a; \Delta, a \in I \vdash \tau <: \tau'}{\Psi; \Theta; \Delta \vdash !_{\sum_{a \in I} R_a} \left[P_a \right] \tau \ \ <: \ \left[\sum\limits_{a \in I} R_a \cdot P_a \right] !_{\sum_{a \in I} R_a} \tau'}\ \text{sub-BP2}$$

Fig. 12. Selected typing and subtyping rules of pλ-amor$^{\mathsf{C}}$

which are formalised as the subtyping rules **sub-BP1** and **sub-BP2** (Fig. 12). Rule **sub-BP1** pushes potential inside a nested graded ! modality, thus reversing the order of the modalities. Rule **sub-BP2** does the converse. These type coercions are admissible in our semantic model because potentials are ghost resources. The soundness of the two subtyping rules is proved as a part of the proof of the fundamental theorem in the technical appendix.

Model and Soundness. The model of types changes in the interpretation of ! and the interpretation of the non-affine context to account for grading as shown in Fig. 13. The interpretation of $!_{\sum_{a \in S} C_a} \tau$ contains the triple $(p, T, !e)$ if the potential p is sufficient to interpret the C_a copies of the a^{th} instance of e for each $a \in S$. A similar change is made to the interpretation of the non-affine context $\mathcal{G}[\![\Omega]\!]$, which also has multiplicity indices.

We re-prove the fundamental theorem (Theorem 7) for this extended model. The statement of the fundamental theorem does not change, but its proof changes wherever grades appear.

$$
\begin{array}{lcl}
\mathcal{V}[\![!_{\sum_{a\in S} C_a}\tau]\!] & \triangleq & \{(p,T,!e) \mid \exists p_0,\ldots,p_{|S|-1}.\,\left(\sum_{i<|S|} p_i \cdot C_a[S(i)/a]\right) \leq p \,\wedge \\
& & \qquad\qquad\qquad\qquad \forall i < |S|.(p_i,T,e) \in \mathcal{E}[\![\tau[S(i)/a]]\!]\} \\
\mathcal{G}[\![\Omega]\!] & \triangleq & \{(p,T,\rho_m) \mid \exists f : \mathcal{V}ars \to \mathbb{N} \to \mathbb{R}^+. \\
& & (\forall (x :_{\sum_{a\in S} C_a} \tau) \in \Omega.\, \forall i < |S|.(f\ x\ i, T, \rho_m(x)) \in \mathcal{E}[\![\tau[S(i)/a]]\!]) \,\wedge \\
& & (\sum_{x:_{\sum_{a\in S} C_a}\tau\in\Omega} \sum_{i<|S|} C_a[S(i)/a] \cdot f\ x\ i) \leq p\}
\end{array}
$$

Fig. 13. Revised clauses of the pλ-amor$^{\mathsf{C}}$ model

6 Related Work

We discuss closely related prior work and compare it to pλ-amor. We believe that, in the context of higher-order probabilistic programming, pλ-amor is the first type theory that simultaneously offers value-dependent potentials, static approximations of distributions in types, and soundness against a model of types (up to couplings, which is crucial for the simplification of static distributions that appear in our examples).

Much work on the cost analysis of a probabilistic programs is in a non higher-order setting. For example, there is work on probabilistic abstract reduction systems [Avanzini et al. 2020a] and on imperative probabilistic programs using ranking super-martingales [Avanzini et al. 2020a; Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2017]. There is also work based on pre-expectation calculi and program logics [Avanzini et al. 2020b; Batz et al. 2023; Kaminski et al. 2016; Ngo et al. 2018; Wang et al. 2021]. pλ-amor is structurally very different from all of these lines of work, as pλ-amor is a type theory for higher-order functional programs.

Work on the cost analysis of *higher-order* probabilistic programs is fairly limited. We discuss three different approaches: type-and-effect systems [Wang et al. 2020], coeffect-based systems [Avanzini et al. 2019] and program transformation-based systems [Avanzini et al. 2021]. pRAML [Wang et al. 2020] is a type-and-effect system for expected cost analysis. It models cost as an effect, which is subtly different but similar to pλ-amor's monadic model of costs. pRAML uses potentials to reason about amortised cost. The key difference between pRAML and pλ-amor is in the expressiveness: pRAML cannot handle value-dependent potentials, which are essential for our examples. Also, unlike pλ-amor, pRAML lacks type-level reasoning about probabilities (probabilities only appear in terms). However, pRAML has been implemented, whereas pλ-amor has no implementation yet.

ℓRPCF [Avanzini et al. 2019] is a coeffect-based type theory for the cost analysis of higher-order probabilistic programs. Its design is based on similar approaches for the cost analysis of deterministic programs [Dal Lago and Gaboardi 2011; Dal Lago and Petit 2012]. The cost model of ℓRPCF is fixed: a unit cost is incurred at every elimination step (β-reduction, unrolling of the fixpoint, and generating a uniform distribution). Hence, ℓRPCF is limited to the analysis of expected running times of programs. In contrast, the cost model of pλ-amor is not fixed and can be specified flexibly by the programmer using the $\uparrow^{\kappa}$ construct. This allows us to model non-standard costs like the loss or regret of an online learning algorithm.

There are also deeper technical differences between ℓRPCF and pλ-amor. ℓRPCF grades the typing derivation, not the types, with the expected cost. As a result, types lack cost information and typing derivations must be analysed to obtain cost information. In contrast, pλ-amor relies on a monadic type, $\mathbb{PC}$, to track expected costs, so costs are internalised into types. Next, pλ-amor uses the same type construct (the aforementioned monad) to track probability distributions. In contrast, ℓRPCF tracks distributions using a dedicated type construct called Dynamic Distribution Types (DDTs). So, stylistically, ℓRPCF separates the tracking of cost (in the typing derivations) from the tracking of distributions (in DDTs), while pλ-amor conflates the two into the same monadic construct. Next,

ℓRPCF assumes that the static distribution in an expression's type and the corresponding runtime distribution of the expression have the same carrier set. pλ-amor relaxes this assumption and instead relates the two distributions up to a coupling, which provides more flexibility in typing programs (but a weaker semantic guarantee). Finally, pλ-amor includes a logical relations model of types, and this model is used to prove the type theory sound. ℓRPCF lacks a model of types and its metatheory is based on a direct analysis of the operational semantics of the language.

Avanzini et al. [2021] present a program transformation-based approach for expected cost analysis of functional programs with probabilistic choice. Their transform is a continuation passing style (CPS) transformation of the given probabilistic program using an expected cost transformer [Kaminski et al. 2016]. The transformed program is not probabilistic and encodes the expected cost directly as a program term, which can be analysed using standard program verification techniques. This is very different from the approach of pλ-amor and the work described above where costs are modelled directly with effects or monads.

7 Conclusion

We have presented pλ-amor, a graded modal type-theory for proving bounds on the expected cost of higher-order probabilistic programs with recursion. pλ-amor uses graded modal types to encode potentials, costs and probability distributions at the type level. The type theory is proved sound relative to a Kripke step-indexed model that uses potentials and probabilistic couplings to give semantics to pλ-amor types. We have used pλ-amor to analyse the expected loss of several examples from the online learning theory literature. Finally, we also presented an extension of pλ-amor, called pλ-amor$^{\mathsf{C}}$, with graded exponential modalities, which enable precise tracking of variable uses.

Acknowledgments

The authors would like to thank the paper's anonymous reviewers for their comments and feedback. Vineet Rajani was supported in part by the EPSRC grant EP/Y003535/1.

Data-Access Statement

The full technical development with the proofs of all our theorems, the typing derivations of all our examples, the details of the extension of pλ-amor with graded exponentials and sub-distributions, and the type-preserving embedding of ℓRPCF in that extension can be found in a technical appendix available on Zenodo [Rajani et al. 2024].

References

Alejandro Aguirre and Lars Birkedal. 2023. Step-Indexed Logical Relations for Countable Nondeterminism and Probabilistic Choice. *Proc. ACM Program. Lang.* 7, POPL (2023), 33–60.

Amal Jamil Ahmed. 2004. *Semantics of types for mutable state.* Ph. D. Dissertation. Princeton University.

Sanjeev Arora. 2013. Lectures on Advanced Algorithm Design. https://www.cs.princeton.edu/courses/archive/fall13/cos521/.

Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing* 8, 6 (2012).

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. 2002. The Nonstochastic Multiarmed Bandit Problem. *SIAM J. Comput.* 32, 1 (2002).

Martin Avanzini, Gilles Barthe, and Ugo Dal Lago. 2021. On continuation-passing transformations and expected cost analysis. *Proc. ACM Program. Lang.* 5, ICFP (2021), 1–30.

Martin Avanzini, Ugo Dal Lago, and Alexis Ghyselen. 2019. Type-Based Complexity Analysis of Probabilistic Functional Programs. In *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS).*

Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. 2020a. On probabilistic term rewriting. *Sci. Comput. Program.* 185 (2020).

Martin Avanzini, Georg Moser, and Michael Schaper. 2020b. A modular cost analysis for probabilistic programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 172:1–172:30.

Nikhil Bansal and Anupam Gupta. 2019. Potential-Function Proofs for Gradient Methods. *Theory Comput.* 15 (2019).

Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. 2023. A Calculus for Amortized Expected Runtimes. *Proc. ACM Program. Lang.* 7, POPL (2023), 1–28.

Olivier Bournez and Florent Garnier. 2005. Proving Positive Almost-Sure Termination. In *Term Rewriting and Applications*. Springer Berlin Heidelberg, 323–337.

Nicolo Cesa-Bianchi and Gabor Lugosi. 2006. *Prediction, Learning, and Games*. Cambridge university press.

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis using Martingales. In *Computer-Aided Verification (CAV)*, Vol. 8044. 511–526.

Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. 2017. Automated Recurrence Analysis for Almost-Linear Expected-Runtime Bounds. In *International Conference on Computer Aided Verification (CAV)*, Vol. 10426. Springer, 118–139.

Ugo Dal Lago and Marco Gaboardi. 2011. Linear Dependent Types and Relative Completeness. *Logical Methods in Computer Science* 8, 4 (2011).

Ugo Dal Lago and Barbara Petit. 2012. Linear Dependent Types in a Call-by-value Scenario. *Science of Computer Programming* 84 (2012).

Nils Anders Danielsson. 2008. Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*.

Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. 2016. Combining Effects and Coeffects via Grading. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP)*.

Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. 1992. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science* 97, 1 (1992).

Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest Precondition Reasoning for Expected Run-Times of Probabilistic Programs. In *Proceedings of the European Symposium on Programming, (ESOP)*, Vol. 9632. Springer, 364–389.

Annabelle McIver and Carroll Morgan. 2005. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer New York, NY.

John C Mitchell. 1996. *Foundations for programming languages*. Vol. 1. MIT press Cambridge.

Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. 2018. Bounded Expectations: Resource Analysis for Probabilistic Programs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

Chris Okasaki. 1996. *Purely Functional Data Structures*. Ph. D. Dissertation. Carnegie Mellon University.

Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 1–30.

Tomas Petricek, Dominic A. Orchard, and Alan Mycroft. 2013. Coeffects: Unified Static Analysis of Context-Dependence. In *Automata, Languages, and Programming - International Colloquium*.

Vineet Rajani, Gilles Barthe, and Deepak Garg. 2024. A modal type-theory of expected cost in higher-order probabilistic programs (Technical appendix). https://doi.org/10.5281/zenodo.13450390

Vineet Rajani, Marco Gaboardi, Deepak Garg, and Jan Hoffmann. 2021. A unifying type-theory for higher-order (amortized) cost analysis. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–28.

Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400 – 407.

Robert E. Tarjan. 1985. Amortized computational complexity. *SIAM J. Algebraic Discrete Methods* 6, 2 (1985).

Cedric Villani. 2008. *Optimal transport: Old and New*. Springer Berlin, Heidelberg.

Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. 2018. Contextual equivalence for a probabilistic language with continuous random variables and recursion. *Proc. ACM Program. Lang.* 2, ICFP (2018), 87:1–87:30.

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Central moment analysis for cost accumulators in probabilistic programs. In *Proceedings of the ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*. 559–573.

Di Wang, David M. Kahn, and Jan Hoffmann. 2020. Raising expectations: automating expected cost analysis with types. *Proc. ACM Program. Lang.* 4, ICFP (2020).

Received 2024-04-05; accepted 2024-08-18