Extended Abstract: Type-Directed Reasoning for Probabilistic, Non-Compositional Resources

Christopher Schwaab  
University St Andrews  
cjs26@st-andrews.ac.uk

Edwin Brady  
University St Andrews  
ecb10@st-andrews.ac.uk

Kevin Hammond  
University St Andrews  
kh8@st-andrews.ac.uk

ACM Reference Format:

1 Introduction

Our overall research goal is to develop ways to systematically relate the non-functional properties of a program, such as its time and energy usage, directly to its source code. This will enable source-level reasoning about important software properties, enabling them as first-class citizens. Unfortunately, as processors and compiler optimizations continue to increase in complexity, so modelling such properties is becoming increasingly difficult. Even simple high-level language constructs can be difficult to understand and may require a detailed understanding of the compilation process, ISA and micro-architectural details such as cache, pipeline, memory layout and memory buffers. To exemplify the problem, consider a simple multiplication in C:

\[ y = x \times 5. \]

With no optimization on an x86 machine, the compiler might produce a simple `imul` instruction; however, an optimised version might be `lea rcx, [rax+rax*4]`. This may change which hardware components are used, and so have an impact on execution time, energy usage and other properties.

In this paper, we propose to use a type-directed, statistical approach. We aim to mitigate error by developing a mechanism for mechanically reasoning about both the accuracy and the confidence of a probabilistic cost (time, energy etc.). In our approach, individual program expressions are assigned a probabilistic cost. These are composed in a way that follows the underlying program shape. Intuitively, a term’s shape can be taken by dropping its arguments; the shape of an entire expression is then given as the normalized, flattened tree of term shapes. Individual expressions are costed by an opaque heuristic, \( h^* \), that is determined by observing many expressions of the same shape in different execution contexts and measuring their effects. The resource consumption of a term in each shape is assumed to be sub-gaussian following some underlying distribution. This assumption is key to the possibility of reasoning about potential error, admitting the application of well established bounds. The main contributions of the full paper will be to develop:

1. a type-directed, probabilistic approach to resource analysis (for time, energy etc.), relying on a single, calculated distribution; and
2. mechanisms for mechanically reasoning about error.

2 Cost typed expressions

We use a simple language of expressions with conditionals and down to loops. So, e.g., the factorial function is:

\[
\text{for } i = n \text{ to } 0, s \leftarrow n \times s
\]

To understand the time/energy usage of factorial \( n \), we ideally need to know:

1. the cost of testing to exit the loop when \( n = 0 \);
2. the cost of the multiplication; and
3. the overhead of performing the loop.

In the style of Hume [1], our language is broken into two layers: expressions whose terms are costed atomically by a heuristic \( h^* \), and a coordination layer of statements which are costed semi-structurally. Statement costs are not entirely structural because the system must account for the extra cost of machine level control flow. As an example, the rule for the cost of a loop is the cost of the loop body and the conditional jump, times the number of iterations—which is assumed to be statically known
We would like to derive a general type soundness theorem to guarantee that the cost is “reasonably close” to some expected value. Intuitively, the compiled machine code representation $m$ of a well-typed statement $\sigma \vdash s : C \cup \epsilon, \rho$ should generally have running time close to $C$. Given that the underlying execution time follows a distribution $D$, we can interpret “generally” to mean the expected running time of $m$, $E[T].$

As a machine model $M$ we use a simple SSA language with infinite registers where each instruction has probabilistic cost. Program execution is modeled with a small-step operational semantics $\leadsto$. The step relation takes a triple of current time $T$, environment $\sigma$, and program $m$ to some approximate future time $T + K$, an updated environment $\sigma'$, and a program continuation $m'$. Programs in $S$ are transformed to machine code by a standard CPS compiler with value domain $\mathcal{V}$

$$\text{compile} : S_{\tau} \rightarrow (\mathcal{V}_{\tau} \rightarrow M_\alpha) \rightarrow M_\alpha.$$ 

Given a model of a machine, the soundness theorem ensures that the cost of a program, $p$, is within $\epsilon$ of the actual cost of the underlying machine code, $m$.

### Theorem 3.1 (Type soundness).

\[
\forall (\sigma : \text{Env})(s : S_\tau)(\epsilon, \rho : \mathbb{R}). \sigma \vdash s : C \cup \epsilon, \rho \land \\
\forall (P : M_{\tau} \rightarrow *) (k : \mathcal{V} \rightarrow M_\tau)(\forall (v : \mathcal{V}). P(k \cdot v)) \Rightarrow \\
\exists (m : M_{\tau}). \text{compile } s = k \land \\
(\exists(v : \mathcal{V}) T.0, \sigma, m \leadsto^* T, \sigma', \text{ret } v) \land \\
P(|C - E[T]| \geq \epsilon) \leq \rho)
\]

This states that given a program $s$ with cost $C$, accuracy $\epsilon$, and confidence $\rho$, and supposing that $s$ compiles to $m$, then our system guarantees, with $\rho$ confidence, that the estimated cost, $C$, will never be greater than $\epsilon$ steps away from the expected cost of the compiled program.

### 4 Example: Costs for factorial

Given the soundness theorem, it is easy to analyse the accuracy of the costs for the factorial of $n = 2$.

\[
\sigma \vdash 2 \times s : C \cup \frac{\epsilon}{2}, \sqrt{\rho} \\
\sigma \vdash \text{for } s = 1 \text{ to } 0, \ s \leftarrow 2 \times s : u(C_0 + C) \cup \epsilon, \rho
\]

The leaves of the typing derivation tell us what accuracy and confidence are required of $h^*$.

### 5 Conclusion

We have sketched how to determine probabilistic program execution costs, in terms of time, energy etc. from a source level program, using a type-based approach derived from base term costs, and building on known results from machine learning. We have stated the key soundness result that is required. We have illustrated the use of our approach using a simple factorial function. Clearly, it is necessary to complete this proof and to determine the best way to calculate $h^*$ for more complex examples. There are then several avenues that would repay further investigation. These include: Loop Bounds. Currently, we only handle loops with a fixed iteration count. There is a large body of work in e.g. the worst-case execution time community on determining more complex loop bounds [4, 5]. Effects. To cost basic effects we simply need to know the cost and the error. However, we would need e.g. a dependent type system if the overall cost depended on the value of the effect (e.g. if a loop bound depended on the value of some variable). Higher order functions and Lazy Evaluation. We can exploit similar approaches to those taken by Jost et al. [2, 3] for amortised analysis, to embed costs within our types and track these across composed expressions.
References


