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:
Christopher Schwaab, Edwin Brady, and Kevin Hammond. 2017. Extended Abstract: Type-Directed Reasoning for Probabilistic, Non-Compositional Resources. In *Proceedings of Type-Driven Development, Oxford, United Kingdom, Sun. 3 Sep. 2017 (TyDE),* 3 pages.

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: $y = x * 5$. 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 shape is then given as the normalized, flattened tree of term shapes. Individual expressions are costed by an opaque heuristic, $h^\star$, 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:

$$\textbf{for} \begin{array}{c} s = 1 \\ i = n \end{array} \textbf{ to } 0.\ s \leftarrow n * s$$

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 is 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 $h^\star$, 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

$$\text{CLoop}\frac{C_0 \sim h^\star(\phi(\textbf{for})) \quad [\![n]\!] = u \quad \sigma \vdash s :^{h^\star} C}{\sigma \vdash \textbf{for} \begin{array}{c} w = w_0 \\ i = n \end{array} \textbf{ to } 0.\ w \leftarrow s :^{h^\star} u(C_0 + C)}$$

TyDE, Sun. 3 Sep. 2017, Oxford, United Kingdom

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00

Here, the environment σ is a mapping from free names, such as functions, to program terms. The heuristic $h^\star$ gives an estimate of the running time consumed by the loop's control flow. More generally this function costs any expression e by $h^\star(\phi(e))$—where the function ϕ decomposes e into its normalized, flattened shape tree. Where does this estimating function come from? Assuming that the true cost of an expression is drawn from some underlying distribution, $h^\star$ can be calculated by observing a large number of representative training samples. However, because $h^\star$ is only an estimate, a means of reasoning about its error is required. This is accomplished by extending the costing rules on C with *accuracy*, ϵ, and *confidence*, ρ, written $s : C \uplus \epsilon, \rho$. The CLoop rule is then readily extended:

$$\text{HLoop}\ \frac{\begin{array}{c} C_0 \sim h^\star(\phi(\textbf{for})) \\ [\![n]\!] = u - 1 \quad \sigma \vdash s : C \uplus \frac{\epsilon}{u},\ \sqrt[u]{\rho} \end{array}}{\sigma \vdash \textbf{for}\ \begin{array}{c} w = w_0 \\ i = n \end{array}\ \textbf{to}\ 0.\ w \leftarrow s : u(C_0 + C) \uplus \epsilon, \rho}$$

Intuitively the **for** statement has some finite amount of top-level confidence and accuracy which are split evenly across the executions of its body s and the overhead of the conditional branch.

3 Soundness

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 $\sigma \vdash s : C \uplus \epsilon, \rho$ 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, $\mathbb{E}[T]$

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 $- \rightsquigarrow -$. The step relation takes a triple of current time T, environment σ, and program m to some *approximate* future time $T + K$, an updated environment σ', and a program continuation m'. Programs in S are transformed to machine code by a standard CPS compiler with value domain $\mathcal{V}$

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

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

Theorem 3.1 (Type soundness)**.**

$$\begin{array}{l} \forall(\sigma : \mathit{Env})(s : S_\tau)(\epsilon, \rho : \mathbb{R}).\sigma \vdash s : C \uplus \epsilon, \rho \wedge \\ \forall(P : M_\tau \to \star)(k : \mathcal{V} \to M_\tau)(\forall(v : \mathcal{V}).P\ (k\ v)) \Rightarrow \\ \exists(m : M_\tau).\mathit{compile}\ s\ k = m \wedge \\ \quad (\exists(v : \mathcal{V})\ T.0, \sigma, m \rightsquigarrow^\star T, \sigma', \mathrm{ret}\ v) \wedge \\ \quad P(|C - \mathbb{E}[T]| \geq \epsilon) \leq \rho) \end{array} \tag{1}$$

This states that given a program s with cost C, accuracy ϵ, and confidence ρ, and supposing that s compiles to m, then our system guarantees, with ρ confidence, that the estimated cost, C, will never be greater than ϵ 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 $n = 2$.

$$\frac{\dfrac{C \sim h^\star(\phi(2 * s) = \{\mathrm{MUL}\})}{\sigma \vdash 2 * s : C \uplus \frac{1}{3}\epsilon,\ \sqrt[3]{\rho}} \quad C_0 \sim h^\star(\phi(\textbf{for}))}{\sigma \vdash \textbf{for}\ \begin{array}{c} s = 1 \\ i = 2 \end{array}\ \textbf{to}\ 0.\ s \leftarrow 2 * s : u(C_0 + C) \uplus \epsilon, \rho}$$

The leaves of the typing derivation tell us what accuracy and confidence are required of $h^\star$. To solve for $h^\star$ programs are generated and their expressions costs observed in a variety of contexts. Finally, Hoeffding's inequality can guarantee the accuracy and confidence of $h^\star$ given sufficient training samples.

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 $h^\star$ 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

[1] Kevin Hammond. 2000. Hume: a Concurrent Language with Bounded Time and Space Behaviour. In *Proc. 7th IEEE International Conference on Electronic Control Systems (ICECS 2K), Lebanon*. 407–411.

[2] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. 2010. Static Determination of Quantitative Resource Usage for Higher-order Programs. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '10)*. ACM, New York, NY, USA, 223–236. https://doi.org/10.1145/1706299.1706327

[3] Steffen Jost, Pedro Vasconcelos, Mário Florido, and Kevin Hammond. 2017. Type-Based Cost Analysis for Lazy Functional Languages. *J. Autom. Reason.* 59, 1 (June 2017), 87–120. https://doi.org/10.1007/s10817-016-9398-9

[4] Jens Knoop, Laura Kovács, and Jakob Zwirchmayr. 2012. *Symbolic Loop Bound Computation for WCET Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 227–242. https://doi.org/10.1007/978-3-642-29709-0_20

[5] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The Worst-case Execution-time Problem&Mdash;Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages. https://doi.org/10.1145/1347375.1347389