A General Equational Framework for Static Profiling of Parametric Resource Usage ∗ †

P. LOPEZ-GARCIA[1,2] M. KLEMEN[1] U. LIQAT[1] M.V. HERMENEGILDO[1,3]

[1] *IMDEA Software Institute*
(*e-mail:* {pedro.lopez,maximiliano.klemen,umer.liqat,manuel.hermenegildo}@imdea.org)
[2] *Spanish Council for Scientific Research (CSIC)*
[3] *Technical University of Madrid (UPM)*

Extended Abstract

Resources are numerical properties about the execution of a program, such as number of resolution steps, execution time, energy consumption, number of calls to a particular predicate, number of network accesses, number of transactions in a database, etc. The goal of automatic static cost analysis is estimating the resource usage of the execution of a program without running it, as a function of input data sizes and possibly other (environmental) parameters. The significant body of work on static analysis for logic programs has actually also been applied to the analysis of other programming paradigms, including imperative programs. This is achieved via a transformation of the program into *Horn Clauses* (Méndez-Lojo et al. 2007). In this work we concentrate on the analysis of Horn Clause programs, independently of whether they are the result of a translation or the actual program source.

Given a program $\mathcal{P}$ and a predicate $\mathtt{p} \in \mathcal{P}$ of arity k and a set Π of k-tuples of actual arguments to $\mathtt{p}$, we refer to the *standard cost* of a call $\mathtt{p}(\bar{\mathtt{e}})$ (i.e., a call to $\mathtt{p}$ with actual data $\bar{e} \in \Pi$), as the resource usage (under a given cost metric) of the complete execution of $\mathtt{p}(\bar{\mathtt{e}})$. Thus, the *standard cost* is a per-call cost formalized as a function $\mathcal{C}_{\mathtt{p}} : \Pi \rightarrow \mathcal{R}_{\infty}$, where $\mathcal{R}_{\infty}$ is the set of real numbers augmented with the special symbol ∞ (which is used to represent non-termination). *Standard cost*, and, in general, resource usage information, is very useful for a number of applications, such as automatic program optimization, verification of resource-related specifications, detection of performance bugs, or helping developers make resource-related design decisions. In the latter case, the analysis has to show which parts of the program are the most resource-consuming, i.e., which predicates would bring the highest overall improvement if they were optimized, so that programming efforts can be focused more productively. The standard cost information only partially meets these objectives. For example, often predicates with the highest (standard) cost are not the ones whose optimization is most profitable, since predicates which have lower costs but which are called more often may be responsible for a larger part of the overall resource usage. The input data sizes to such calls are also relevant. Thus, rather than the global costs provided by standard cost analyses, what is really needed in many such

† This document is an extended abstract of the article (Lopez-Garcia et al. 2016).

∗ This research has been partially funded by EU FP7 agreement no 318337, *ENTRA*, Spanish MINECO TIN2015-67522-C3-1-R *TRACES* project, and the Madrid M141047003 *N-GREENS* program. Special thanks are due to John Gallagher for many fruitful and inspiring discussions.

applications is the results of a *static profiling* of the program that helps identify the parts of a program responsible for highest fractions of the cost, or, more generally, how the total resource usage of the execution of a program is *distributed* over selected parts of it. By *static profiling* we mean the static inference of the kinds of information that are usually obtained at run-time by profilers.

For this reason, herein we are more interested in what we refer to as *accumulated cost*. To give an intuition of this concept, we first explain our notion of *cost centers*, which is similar to the one we use in (Haemmerlé et al. 2016), and was inspired from (Sansom and Jones 1995; Morgan and Jarvis 1998): they are user-defined program points (predicates, in our case) to which execution costs are assigned during the execution of a program. Data about computational events is accumulated by the cost center each time the corresponding program point is reached by the program execution control flow. Assume for example that predicate $\mathtt{p}$ calls another predicate $\mathtt{q}$ (either directly or indirectly), and that we declare that both predicates are cost centers. In this case, the cost of a (single) call $\mathtt{p}(\bar{\mathtt{e}})$ *accumulated in* cost center $\mathtt{q}$, denoted $\mathcal{C}^{\mathtt{q}}_{\mathtt{p}}(\bar{\mathtt{e}})$, expresses how much of the standard cost of $\mathtt{p}(\bar{\mathtt{e}})$ is attributed to $\mathtt{q}$, and is the sum of the costs of the computations performed "under the scope" of all the calls to $\mathtt{q}$ generated during the complete execution of $\mathtt{p}(\bar{\mathtt{e}})$. We say that a computation is "under the scope" of a call to cost center $\mathtt{q}$ if the closest ancestor of such computation in the call stack that is a cost center is $\mathtt{q}$. The *accumulated cost* is formalized as a function $\mathcal{C}^{\mathtt{q}}_{\mathtt{p}} : \Pi \rightarrow \mathcal{R}_\infty$. We refer the reader to (Haemmerlé et al. 2016) for a formal definition of accumulated cost.[1]

The goal of static analysis is to infer approximations (i.e., abstractions) of the concrete functions $\mathcal{C}^{\mathtt{q}}_{\mathtt{p}}$ and $\mathcal{C}_{\mathtt{p}}$ (or, more precisely, of the extensions of such functions to the powerset of Π) that represent the *accumulated* and *standard* cost respectively. In this work we propose a novel, general, and flexible framework for setting up cost equations/relations which can be instantiated for performing a wide range of static resource usage analyses, including both *accumulated cost* and standard cost. Our starting point is the well-developed technique of setting up recurrence relations representing resource usage functions parameterized by input data sizes (Wegbreit 1975; Rosendahl 1989; Debray et al. 1990; Debray and Lin 1993; Debray et al. 1997; Navas et al. 2007; Albert et al. 2011; Serrano et al. 2014), which are then solved to obtain (exact or safely approximated) closed-forms of such functions (i.e., functions that provide upper or lower bounds on resource usage in general). In addition, recently many other approaches have been proposed for resource analysis (Vasconcelos and Hammond 2003; Hoffmann et al. 2012; Grobauer 2001; Igarashi and Kobayashi 2002; Nielson et al. 2002; Giesl et al. 2012; Albert et al. 2011; Gulwani et al. 2009). While based on different techniques, all these analyses are aimed at inferring the *standard* notion of cost. Please see (Haemmerlé et al. 2016) for a further discussion of related work. Our proposal extends and generalizes the standard resource analysis techniques that are based on setting up recurrence relations. This is mainly achieved by introducing into the derived relations extra Boolean control variables whose value is 0 or 1. A particular resource profile can be analyzed by assigning values to the control variables, effectively switching on or off different terms in the relations. The standard resource analysis is obtained by assigning 1 to all variables. We also define

[1] In (Haemmerlé et al. 2016) we use the notation $\mathtt{C}^{\mathtt{p}}_{\mathtt{q}}(\bar{\mathtt{e}})$ instead of $\mathcal{C}^{\mathtt{q}}_{\mathtt{p}}(\bar{\mathtt{e}})$.

a concrete Boolean variable assignment that instantiates our framework so that it performs *static profiling of accumulated cost*, similarly to (Haemmerlé et al. 2016), where the results are also parameterized by input data sizes. However, the approach we present in this work is quite different from our previous approach (Haemmerlé et al. 2016), which was based on a program transformation. The main contributions of this work and the differences and advantages over that work can be summarized as follows:

- We propose a novel, general, and flexible framework for setting up cost relations which can be instantiated for performing a wide range of resource usage analyses. Is more general than (Haemmerlé et al. 2016), which is limited to accumulated cost analysis.
- Our new approach can deal with non-deterministic/multiple-solution predicates, unlike (Haemmerlé et al. 2016). This is obviously a requirement for analyzing logic programs and is also useful for dealing with certain aspects of imperative programs, such as multiple dispatch; see (Méndez-Lojo et al. 2007). While our previous approach could conceivably be extended to deal with such programs, it would certainly result in a more complicated and indirect solution.
- Our new approach and its implementation are based on a direct application of abstract interpretation and integration into the Ciao preprocessor, CiaoPP (Hermenegildo et al. 2005), rather than on a program transformation. As a result, many useful CiaoPP features are inherited for free, such as *multivariance* (being able to infer separate cost functions for different abstract call patterns for the same predicate), communication with the other required analyses, integrated treatment of special control features (such as, e.g., the cut), assertion-based verification and user interaction, efficient fixpoint, etc. Also, for this integration we define a novel abstract domain for resource analysis that keeps track of the *environment*.
- The integration also inherits the capability of CiaoPP's analyzers of *analyzing for several resources* at the same time. While it might be possible to define a new transformation capable of keeping track of several resources, this would further complicate the transformed program, and in any case requires additional work.
- Finally, as our experimental results show, our new approach is more efficient than the transformation-based approach. This is not only due to its implementation as a direct abstract interpretation, but also to the inclusion and use of reachability information, performed automatically by the abstract interpretation framework.

Finally, we argue that our approach is quite general, and it can be easily applied to other paradigms, including imperative programs, functional programs, CHR, etc., using the strategy based on compilation to Horn Clauses, as in our previous work with Java or XC.

References

Albert, E., Arenas, P., Genaim, S., and Puebla, G. 2011. Closed-Form Upper Bounds in Static Cost Analysis. *Journal of Automated Reasoning 46,* 2, 161–203.

Albert, E., Genaim, S., and Masud, A. N. 2011. More Precise yet Widely Applicable Cost Analysis. In *Proc. of VMCAI'11.* LNCS, vol. 6538. Springer, 38–53.

Debray, S. K. and Lin, N. W. 1993. Cost analysis of logic programs. *ACM TOPLAS 15,* 5 (November), 826–875.

Debray, S. K., Lin, N.-W., and Hermenegildo, M. V. 1990. Task Granularity Analysis in Logic Programs. In *Proc. PLDI'90.* ACM, 174–188.

Debray, S. K., López-García, P., Hermenegildo, M. V., and Lin, N.-W. 1997. Lower Bound Cost Estimation for Logic Programs. In *ILPS'97*. MIT Press, 291–305.

Giesl, J., Ströder, T., Schneider-Kamp, P., Emmes, F., and Fuhs, C. 2012. Symbolic evaluation graphs and term rewriting: a general methodology for analyzing logic programs. In *Proceedings of PPDP'12*. ACM, 1–12.

Grobauer, B. 2001. Cost recurrences for DML programs. In *Proceedings of ICFP '01*. ACM, New York, NY, USA, 253–264.

Gulwani, S., Mehra, K. K., and Chilimbi, T. M. 2009. SPEED: Precise and Efficient Static Estimation of Program Computational Complexity. In *The 36th Symposium on Principles of Programming Languages (POPL'09)*. ACM, 127–139.

Haemmerlé, R., Lopez-Garcia, P., Liqat, U., Klemen, M., Gallagher, J. P., and Hermenegildo, M. V. 2016. A Transformational Approach to Parametric Accumulated-cost Static Profiling. In *FLOPS'16*. LNCS, vol. 9613. Springer, 163–180.

Hermenegildo, M., Puebla, G., Bueno, F., and García, P. L. 2005. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming 58,* 1–2, 115–140.

Hoffmann, J., Aehlig, K., and Hofmann, M. 2012. Multivariate amortized resource analysis. *ACM TOPLAS 34,* 3, 14:1–14:62.

Igarashi, A. and Kobayashi, N. 2002. Resource usage analysis. In *Symposium on Principles of Programming Languages*. ACM, 331–342.

Lopez-Garcia, P., Klemen, M., Liqat, U., and Hermenegildo, M. V. 2016. A General Framework for Static Profiling of Parametric Resource Usage. *Theory and Practice of Logic Programming, 32nd Int'l. Conference on Logic Programming (ICLP'16) Special Issue 16,* 5-6 (October), 849–865.

Méndez-Lojo, M., Navas, J., and Hermenegildo, M. 2007. A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs. In *LOPSTR*. LNCS, vol. 4915. Springer-Verlag, 154–168.

Morgan, R. G. and Jarvis, S. A. 1998. Profiling Large-Scale Lazy Functional Programs. *Journal of Functional Programing 8,* 3, 201–237.

Navas, J., Mera, E., López-García, P., and Hermenegildo, M. 2007. User-Definable Resource Bounds Analysis for Logic Programs. In *Proc. of ICLP'07*. LNCS, vol. 4670. Springer, 348–363.

Nielson, F., Nielson, H., and Seidl, H. 2002. Automatic complexity analysis. In *Programming Languages and Systems*. LNCS. Springer, 243–261.

Rosendahl, M. 1989. Automatic Complexity Analysis. In *Proc. of FPCA'89*. ACM Press, 144–156.

Sansom, P. M. and Jones, S. L. P. 1995. Time and Space Profiling for Non-Strict, Higher-Order Functional Languages. In *Proc. of POPL'95*. ACM, New York, NY, USA, 355–366.

Serrano, A., Lopez-Garcia, P., and Hermenegildo, M. V. 2014. Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types. *TPLP, ICLP'14 Special Issue 14,* 4-5, 739–754.

Vasconcelos, P. and Hammond, K. 2003. Inferring Cost Equations for Recursive, Polymorphic and Higher-Order Functional Programs. In *IFL'03*. LNCS, vol. 3145. Springer, 86–101.

Wegbreit, B. 1975. Mechanical Program Analysis. *Comm. of the ACM 18,* 9, 528–539.