Cost Analysis Using Automatic Size and Time Inference

Álvaro J. Rebón Portillo[1], Kevin Hammond[1],
Hans-Wolfgang Loidl[2], and Pedro Vasconcelos[1]

[1] School of Computer Science, University of St Andrews
St Andrews, KY16 9SS, UK
{alvaro,kh,pv}@dcs.st-and.ac.uk
[2] Ludwig-Maximilians-Universität München
Institut für Informatik, D-80538 München, Germany
hwloidl@informatik.uni-muenchen.de

Abstract. Cost information can be exploited in a variety of contexts, including parallelizing compilers, autonomic GRIDs and real-time systems. In this paper, we introduce a novel type and effect system – the *sized time system* that is capable of determining upper bounds for both time and space costs, and which we initially intend to apply to determining good granularity for parallel tasks. The analysis is defined for a simple, strict, higher-order and polymorphic functional language, $\mathcal{L}$, incorporating arbitrarily-sized list data structures. The inference algorithm implementing this analysis constructs cost- and size-terms for $\mathcal{L}$-expressions, plus constraints over free size and cost variables in those terms that can be solved to produce information for higher-order functions. The paper presents both the analysis and the inference algorithm, providing examples that illustrate the primary features of the analysis.

1 Introduction

Good cost information is useful or even vital to a large number of application areas. Examples range from small-scale real-time embedded systems through databases and parallel systems to large-scale *autonomic* GRID computations. We are especially concerned with the issue of determining appropriate task granularity, which is highly important to the efficient execution of parallel programs [1]: excessively fine granularity introduces high overheads; conversely, excessively coarse granularity can lead to poor load balance and starvation. This paper introduces a novel static cost analysis for automatically determining upper bound cost information in the presence of higher-order, polymorphic but non-recursive functions.

Our static analysis is defined as a *type and effect system* [2], a modern approach that uses standard type inference mechanisms to perform static analysis. A type system defines upper bound costs and sizes for expressions in a simple, strict, higher-order and polymorphic functional language, $\mathcal{L}$. Types include size- and cost-annotations, which are related by a subtyping relation [3]. The

R. Peña and T. Arts (Eds.): IFL 2002, LNCS 2670, pp. 232–247, 2003.

corresponding inference algorithm yields cost and size terms for $\mathcal{L}$-expressions, plus constraints over cost and size variables. These constraints are resolved in a separate constraint solver and combined with the cost and size terms to yield closed forms of those terms. These closed forms can be used to solve the costs of function applications.

While our focus is on cost rather than size information, we also infer a restricted form of size information primarily in order to obtain cost information in common cases where cost depends on input sizes. Our analysis is defined for both scalar and compound data structures. We have illustrated the approach with reference to recursive lists, the primary data structure used in functional languages. It should be straightforward to extend the analysis to other recursive data structures such as binary trees, or to arbitrary non-recursive data structures such as vectors or tuples.

The design of our analysis is guided by the intended use of the information it can provide. Although it is desirable to produce quality cost information, precise cost information is not absolutely essential for scheduling parallel tasks: it is sufficient to be able to identify tasks that are *potentially* large enough to be worth executing in parallel. We have structured our system so as to generate strict upper bounds on size and cost information.

2 A Sized Time System for $\mathcal{L}$

$\mathcal{L}$ is a very simple functional language, intended solely as a vehicle to explore static analysis for cost determination. $\mathcal{L}$ is strict, polymorphic, and higher-order; with lists as its only compound data type. The abstract syntax of $\mathcal{L}$ is given below. For simplicity, variables, $v \in \mathbf{Var}$, and constants, $k \in \mathbf{Const}$, are required to be disjoint and all names are unique. Boolean values, $b \in \{\mathsf{true}, \mathsf{false}\}$, and natural numbers, $n \in \mathbb{N}$, are both in $\mathbf{Const}$.

$$e \;:=\; v \;\mid\; k \;\mid\; \lambda v.e \;\mid\; e_1\, e_2 \;\mid\; \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \;\mid\; \mathsf{let}\ v = e_1\ \mathsf{in}\ e_2\,.$$

Local bindings (let) are non-recursive. An $\mathcal{L}$ program is defined to be an $\mathcal{L}$ expression.

2.1 The Type Language

$\mathcal{L}$ uses *sized types* [4], a small extension to standard Hindley-Milner polymorphic types: each type, other than function and boolean types, has a superscript specifying an upper bound for its size. For function types, a *latent cost* [5] is attached to the function arrow. The latent cost of a function type is an upper bound for the cost of evaluating the function body. In the following syntax of type expressions, α represents a type variable:

$$\tau \;:=\; \alpha \;\mid\; \mathsf{Bool} \;\mid\; \mathsf{Nat}^z \;\mid\; \mathsf{List}^z\tau \;\mid\; \tau_1 \xrightarrow{z} \tau_2\,.$$

Size and latent cost expressions are specified by z-expressions:

$$z \;:=\; l \;\mid\; n \;\mid\; z_1 + z_2 \;\mid\; z_1 - z_2 \;\mid\; z_1 \times z_2 \;\mid\; \max(z_1, z_2) \;\mid\; \omega\,.$$