Science of Computer Programming 111 (2015) 395–425

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

On bounding space usage of streams using interpretation analysis

Marco Gaboardi [a,b,1,2], Romain Péchoux [c,*,1]

[a] Harvard University, United States
[b] University of Dundee, United Kingdom
[c] Université de Lorraine - INRIA project Carte, Loria, UMR 7503, France

ARTICLE INFO

Article history:
Received 2 September 2013
Received in revised form 20 April 2015
Accepted 17 May 2015
Available online 1 June 2015

Keywords:
Stream programs
Interpretations
Program space usage
Lazy languages
Implicit computational complexity

ABSTRACT

Interpretation methods are important tools in implicit computational complexity. They have been proved particularly useful to statically analyze and to limit the complexity of programs. However, most of these studies have been so far applied in the context of term rewriting systems over finite data.

In this paper, we show how interpretations can also be used to study properties of lazy first-order functional programs over streams. In particular, we provide some interpretation criteria useful to ensure two kinds of stream properties: *space upper bounds* and *input/output upper bounds*. Our space upper bounds criteria ensure global and local upper bounds on the size of each output stream element expressed in terms of the maximal size of the input stream elements. The input/output upper bounds criteria consider instead the relations between the number of elements read from the input stream and the number of elements produced on the output stream.

This contribution can be seen as a first step in the development of a methodology aiming at using interpretation properties to ensure space safety properties of programs working on streams.

1. Introduction

The advances obtained in communication technology in the last two decades have posed new challenges to the software community. One of these challenges comes from the advancements achieved in computer networking where new software able to handle huge amount of data in an efficient way is required.

This situation has brought a renewed interest for stream-like data structures and for programs managing those data structures. Indeed, by representing discrete potentially infinite information flows, streams can be used to formalize and study situations as real-time data processing, network communication flows, audio and video signals flows, etc. Clearly, the problems that stream programs raise are different from the ones generally considered in the usual scenario where data

* Corresponding author.
E-mail addresses: m.gaboardi@dundee.ac.uk (M. Gaboardi), pechoux@loria.fr (R. Péchoux).
URLs: http://www.cs.unibo.it/~gaboardi (M. Gaboardi), http://www.loria.fr/~pechoux (R. Péchoux).

[1] Work partially supported by the projects ANR-14-CE25-0005 "ELICA", MIUR-PRIN'07 "CONCERTO" and INRIA Associated Team "CRISTAL".
[2] The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement n° 272487.

http://dx.doi.org/10.1016/j.scico.2015.05.004
0167-6423/

are assumed to be finite. For this reason, several programming languages have been proposed with the aim of modeling stream-based computations, see [39] for a survey.

The aim of the present work is to contribute to the current scenario by developing some static analysis techniques useful to ensure basic properties of lazy functional programs working on streams. This is the first step of a more general investigation of complexity and efficiency properties of programs working on streams.

Stream-like languages and properties Several formal frameworks have been designed for the manipulation of infinite objects including infinitary rewriting [23] and infinitary lambda-calculus [24]. Important properties of these models such as infinitary weak normalization and infinitary strong normalization have been deeply studied in the literature. However, little attention has been paid to space properties of such models. A different setting handling infinite data-structures is computable analysis, which provides several models of computation over real numbers [40]. In this setting a lot of work has been done to adapt the classical concept of complexity class and obtain implicit characterizations. However, even if streams can be considered as particular real numbers, the properties of interest for stream programs are usually different from the ones of interest in computable analysis.

A well-established approach to deal with infinite data, and in particular with streams, is by using *laziness* in functional programming languages [21]. In languages like Haskell, streams are expressions denoting infinite lists whose elements are evaluated on demand. In this way streams can be treated by finitary means. The practical diffusion of lazy programming languages has stimulated the development of tools and techniques in order to prove properties of programs in the presence of infinite data structures.

For example, on the side of program equivalence much attention has been paid to the study of co-induction and bisimulation techniques in languages working on streams [35,20]. A property of stream definitions that has motivated many studies is productivity [14]. A stream definition is productive if it can be effectively evaluated in a unique constructor infinite normal form. Productivity is in general undecidable, so, many restricted languages and restricted criteria have been studied to ensure it [38,12,22,15].

Besides program equivalence and productivity, other stream program properties, in particular space-related properties, have received little attention. Such properties are studied in this paper through the use of interpretations, a static analysis tool.

Interpretations Interpretation methods originate from the natural observation that, in order to reason about program properties, it is sometimes more convenient to interpret syntactic program constructions into the objects of an abstract domain and prove properties about the obtained abstract objects.

Interpretation methods have been proved useful in many situations and are nowadays well-established verification tools for proving properties of programs. Variants of interpretation have been used for example to prove the termination of term rewriting systems [31,26], to obtain sound approximations of program behaviors useful to static analysis [13] and to obtain implicit characterizations of complexity classes [7,32].

One variation of particular interest for implicit computational complexity is the notion of quasi-interpretation [7]. A quasi-interpretation maps program constructions to functions over real numbers. The mapping is chosen in such a way that the function obtained as the interpretation of a program describes an upper bound on the size of the computed values with respect to the size of input values. Thanks to this, quasi-interpretations are particularly adapted to study program complexity in an elegant way.

Another important property of quasi-interpretation is that the problem of finding a quasi-interpretation of a given program for some restricted class of polynomials is decidable [2,9]. This suggests that quasi-interpretations can be used as a concrete tool to analyze the complexity of functional programs.

An important new issue is whether interpretations can be used in order to infer such properties on programs computing over infinite data. Here we approach this problem by considering lazy programs over stream data.

Contribution In this paper, we consider a simple first-order lazy language and we start a systematic study of *space properties* of programs working on streams by means of interpretation methods.

In many stream applications one is interested in processing data in a fast and memory-safe way. In order to do this, one can think to improve space-efficiency by using some buffering operations to memorize only the part of the stream involved in the actual computation. Following this intuition, it becomes natural to study space properties of programs working on streams in a more abstract way. We study two classes of space properties:

- *Stream upper bounds*: these are properties about the size of each stream element produced by a program. They correspond to properties about the elements memorized in the buffer.
- *Bounded input/output properties*: these are properties about the number of stream elements produced by a program. They correspond to properties about the number of elements produced on the output wrt to the number of elements read on the input.