Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 276 (2011) 247–261

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Ramified Corecurrence and Logspace

Ramyaa Ramyaa[a] and Daniel Leivant[a,b]

[a] *Indiana University*

[b] *LORIA Nancy*

Abstract

Ramified recurrence over free algebras has been used over the last two decades to provide machine-independent characterizations of major complexity classes. We consider here ramification for the dual setting, referring to coinductive data and corecurrence rather than inductive data and recurrence.
Whereas ramified recurrence is related basically to feasible time (PTime) complexity, we show here that ramified corecurrence is related fundamentally to feasible space. Indeed, the 2-tier ramified corecursive functions are precisely the functions over streams computable in logarithmic space. Here we define the complexity of computing over streams in terms of the output rather than the input, i.e. the complexity of computing the n-th entry of the output as a function of n. The class of stream functions computable in logspace seems to be of independent interest, both theoretical and practical.
We show that a stream function is definable by ramified corecurrence in two tiers iff it is computable by a transducer on streams that operates in space logarithmic in the position of the output symbol being computed. A consequence is that the two-tier ramified corecursive functions over *finite* streams are precisely the logspace functions, in the usual sense.

Keywords: Coinductive data, stream automata, corecurrence, lazy corecurrence, ramification, logarithmic space, implicit computational complexity

1 Introduction

1.1 Overview

Implicit computational complexity (ICC) deals with intrinsic properties of complexity classes, properties that do not refer directly to machine-based resources, such as computation time or space. That is, one matches complexity measures defined in terms of machine models resources, such as time and space, with declarative paradigms that are restricted along functionality, linearity, repetitions, flow control, or similar parameters. The benefits and potential applications of this research are well known (see e.g. [3,14]). Of particular practical interest is the characterization of computational complexity classes by restricted but natural declarative programming languages, since such languages guarantee complexity bounds automatically.

[1] {ramyaa,leivant}@indiana.edu

doi:10.1016/j.entcs.2011.09.025

A well known approach along these lines is *data ramification*, also known as *tiering.* One thinks of data as coming in varying computational strengths. For example, very large data may be thought of as a database to be queried, but too large to be used as a template to drive another recurrence; that is, a function's recurrence argument should be at a higher tier than its output. Data tiering has been used to characterize PTime [3,14], PSpace [15], as well as other feasible complexity classes.

Here we explore the ramification approach for coinductive, rather than inductive data, in particular streams rather than words. We identify a natural notion of logspace Turing machines over streams, and show that it computes those functions over streams that are definable by ramified corecurrence (using two tiers). More precisely, our data-type of choice is the set $S(\Sigma)$ of infinite as well as finite streams over a fixed finite alphabet Σ, i.e. the set generated coinductively (in a sense to be made precise below) from the nullary constructors ε and σ for each $\sigma \in \Sigma$, and the constructor *cons*, with the proviso that the first argument of *cons* is a symbol $\sigma \in \Sigma$. When ε is absent as constructor, we get the infinite streams only. As usual we write $\sigma : S$ for the stream $cons(\sigma, S)$. Evidently, our streams are merely a notational variant of words, infinite as well as finite, over the alphabet Σ.

The connections between coinduction and automata have been studied extensively in a category theoretic setting (see e.g. [21]). Corecurrence (without ramification) were similarly studied in [26,25].

Relations between corecursion and implicit computational complexity have also been explored. A program scheme based on safe recursion over streams was given in [5], but its computational complexity was not fully discussed. Feree et als. [9] use first order functional programs over streams to characterize complexity classes of functionals using second order polynomial interpretations. Semantic characterizations have been given for stream programs' termination and complexity in [10]. These characterizations are for streams over words or natural numbers rather than the digit streams we consider here. Finally, we have studied in [19] the relation between ramified corecurrence over words and the basic feasible functionals of Cook and Urquhart [8]. In the latter one mixes inductive and coinductive data, and ramification applies to both.

2 Machine transducers over streams

2.1 Finite transducers

We consider machine models for computing functions from streams to streams. Our basic machine model is the finite transducer (FT) over streams (often defined over ω-words instead), which allows for one-way reading of the input, and one-way writing of the output, with no requirement to either read or write during a computation step. [2] Note the differences between finite transducers, where acceptance conditions play no role, and familiar finite acceptors, such as Büchi automata. As for FTs over words, FTs over streams can differ in the number of cursors, in their allowed

[2] An equivalent formulation has the machine read a finite word and write a finite word at each step, where each of the two words may be empty [20].