Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 137 (2005) 47–68

www.elsevier.com/locate/entcs

Dealing Denotationally With Stream-based Communication

Mercedes Hidalgo-Herrero[1],[2]

Dept. Didáctica de las Matemáticas
Facultad de Educación, Universidad Complutense de Madrid.
Madrid, Spain

Yolanda Ortega-Mallén[1],[3]

Dept. Sistemas Informáticos y Programación
Facultad de CC. Matemáticas, Universidad Complutense de Madrid.
Madrid, Spain

Abstract

We define a denotational semantics for a kernel-calculus of the parallel functional language Eden. We choose continuations to deal with side-effects (process creation and communication) in a lazy context. The calculus includes streams for communication, and their modelization by a denotational semantics is not direct because a stream may be infinite.

Keywords: Denotational semantics, continuation semantics, laziness, parallel programming, functional programming, Eden.

1 Introduction

Assuming that parallelism and distribution are efficiency improvements in programming, the main goal for designing Eden [7] was to profit from both of them in a functional paradigm. Eden extends the functional language Haskell

[1] Work partially supported by the Spanish project TIC2003-01000.

[2] E-mail: mhidalgo@edu.ucm.es

[3] E-mail: yolanda@sip.ucm.es

doi:10.1016/j.entcs.2005.01.039

[10] with constructs for defining explicit processes, so that the Eden programmer controls —from a higher level of abstraction— the process granularity, the data distribution, and the process topology. This circumstance is endorsed by the fact that the programmer has not to worry about synchronization tasks.

The language Eden comprises two layers: the functional level, or computational model, and the processes level, or coordination model [1]. The computational model is the lazy functional language Haskell, while the coordination level includes the following features:

Process abstractions: expressions that define the general behaviour of a process in a purely functional way.

Process creations: the application of some process abstraction to a particular group of expressions produces the creation of a new process to compute the result of that application.

Interprocess communications: these are asynchronous and implicit, since the programmer does not need to specify the message passing. Communications in Eden are not restricted to the transmission of a single value, processes can communicate values in a stream-like manner.

Eden also includes some constructs to model reactive systems:

Dynamic creation of channels: without this possibility communications are only hierarchical, i.e. from parent to child and viceversa. Dynamic channels facilitate the creation of more complex communication topologies [9].

Non-determinism: in order to model communications from many-to-one Eden introduces a predefined process, `merge`, whose inputs are several streams while its output is just one stream; the latter is the non-deterministic merge of the elements of the former.

The introduction of parallelism leads to a certain loss of laziness:

- Processes are eagerly created even if the output of the new process has not still been demanded.
- Communication is achieved even without demand; whenever a process is created, it is initiated the evaluation of the expressions which will yield the values to be communicated through its channels.

In general, the evaluation of an expression comes to an end when a weak head normal form (whnf) is reached. However, when this value has to be communicated and it is not a λ-abstraction, it will be evaluated to normal form. On the one hand, the head of a stream is strict, so that it is evaluated until a communicable value is obtained. On the other hand, the whole stream evaluation is lazy, allowing in this way the existence of potentially infinite