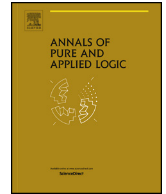
Annals of Pure and Applied Logic 168 (2017) 470–500

Contents lists available at ScienceDirect

Annals of Pure and Applied Logic

www.elsevier.com/locate/apal



Combining control effects and their models: Game semantics for a hierarchy of static, dynamic and delimited control effects

J. Laird

Department of Computer Science, University of Bath, UK

A R T I C L E I N F O

Article history:
Available online 13 October 2016

MSC:
68Q55
68N15
18C20
18C50

Keywords:
Game semantics
Control operators
Computational monads
Exceptions
Continuations
Delimited control

A B S T R A C T

Computational effects which provide access to the flow of control (such as first-class continuations, exceptions and delimited continuations) are important features of higher-order programming languages. There are fundamental differences between them in terms of operational behaviour, expressiveness and implementation, so that understanding how they combine and relate to each other is a challenging objective, with a key role for semantics in making this precise.

This paper develops operational and denotational semantics for a hierarchy of programming languages which include combinations of locally declared control prompts to which a program can escape, with first-class continuations which may either capture their enclosing prompts, or be delimited by them. We describe two different hierarchies of models, both based on categories of games and strategies with a computational monad, but obtained using different methodologies. By *relaxing* combinations of behavioural constraints on strategies with control flow represented by annotation with *control pointers* we are able to give direct and explicit characterizations of control operators and their effects, including examples characterizing their macro-expressiveness. By constructing a parallel hierarchy of models by applying sequences of *monad transformers*, and relating these to the direct interpretation of control effects, we obtain games interpretations of higher-level abstractions such as continuations and exceptions, which can be used as the basis for equational reasoning about programs.

© 2016 The Author. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction and related work

Control effects are key features of higher-order programming languages. They may be used to mark, reify and return to control points in a variety of ways (e.g. with static or dynamic binding, local or global variables, delimited or undelimited continuations). Combining control effects can highlight and amplify these differences, which may have significant impacts, and lead to complicated control flow. Therefore, principles for reasoning about combinations of control effects are important in producing safe and expressive pro-

E-mail address: jiml@cs.bath.ac.uk.

http://dx.doi.org/10.1016/j.apal.2016.10.011
0168-0072/© 2016 The Author. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

grams. Denotational semantics provides a basis for such principles with (broadly speaking) two approaches to combining effects. Constructions such as computational *monads* [19,18] and *continuation-passing-style* interpretation [6], and *algebraic theories* [8] are valuable tools for reasoning about programs, although they typically impose additional layers of definition and interpretation through which this must be filtered, particularly in the presence of multiple effects or properties such as locality. By contrast, *game semantics* provides a setting in which to model combinations of effects more directly by the relaxation of constraints on strategies representing functional programs. This approach has been used successfully to give *fully abstract* interpretations of many features, including an account of locality for features such as state [2,1]. However, the combinatorial nature of games models means that reasoning about denotations — even proving basic soundness results — can be difficult in the absence of structuring principles. Thus it can be useful to relate the direct (games) and indirect (monads) approaches to effects, to gain the advantages of both representations. This paper will do so for control effects which include statically bound, first-class continuations and locally declared, dynamically bound prompts. Determining the interaction between these features presents us with a basic choice: does call-with-current-continuation capture its enclosing prompts, or do they act as *delimiters* for continuations? Allowing either, both or neither of these options leads us to a simple hierarchy of programming languages and their semantics.

1.1. A hierarchy of monads for control

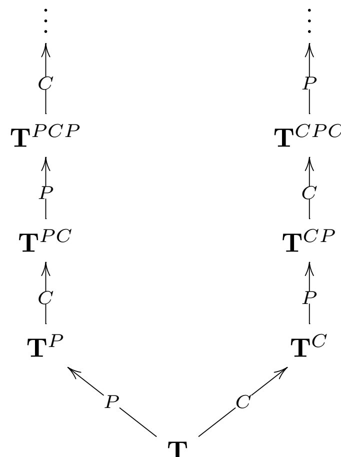
Suppose we have a model of the computational λ-calculus (a λ_C-model) [18] — i.e. a pair $(\mathcal{C}, \mathbf{T})$ consisting of:

- a category $\mathcal{C}$ with finite products and
- a strong monad $(\mathbf{T}, \eta, (_)^*, t)$ on $\mathcal{C}$, and *exponentials* $A \Rightarrow \mathbf{T}B$ for each pair of objects A, B in $\mathcal{C}$.

Assuming that $\mathcal{C}$ also has (distributive) coproducts (and thus an initial object 0 and terminal object 1), we may define further λ_C-models via the following *monad transformers* [24]:

- The *continuations monad transformer*, which sends $\mathbf{T}$ to the strong monad $\mathbf{T}^C = (_ \Rightarrow \mathbf{T}0) \Rightarrow \mathbf{T}0$.
- The *maybe transformer*, which sends $\mathbf{T}$ to the strong monad $\mathbf{T}^P = \mathbf{T}(_ + 1)$.

The latter is often called the *exceptions monad* — we will also use it to interpret continuation-delimiting *prompts*. Note that $(\mathbf{T}^C)^C$ is equivalent to $\mathbf{T}^C$, and $(\mathbf{T}^P)^P$ to the maybe monad $\mathbf{T}(_ + (1+1))$. However, by alternating the continuations and maybe transformers we may obtain a hierarchy of different λ_C-models:



Download English Version:

https://daneshyari.com/en/article/5778168

Download Persian Version:

https://daneshyari.com/article/5778168

Daneshyari.com