Journal of Applied Logic 11 (2013) 1–29

Contents lists available at SciVerse ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Denotation of contextual modal type theory (CMTT): Syntax and meta-programming

Murdoch J. Gabbay [a,*], Aleksandar Nanevski [b]

[a] School of Mathematical and Computer Sciences, Heriot–Watt University, Riccarton Edinburgh, EH14 4AS, United Kingdom
[b] IMDEA Software, Facultad de Informática (UPM), Campus Montegancedo, 28660 Boadilla del Monte, Madrid, Spain

ARTICLE INFO

Article history:
Received 3 January 2012
Accepted 27 June 2012
Available online 16 July 2012

Keywords:
S4 modal logic
Curry–Howard correspondence
Contextual modal type theory
Meta-programming
Higher-order logic
Syntax

ABSTRACT

The modal logic S4 can be used via a Curry–Howard style correspondence to obtain a λ-calculus. Modal (boxed) types are intuitively interpreted as 'closed syntax of the calculus'. This λ-calculus is called modal type theory—this is the basic case of a more general *contextual* modal type theory, or CMTT.

CMTT has never been given a denotational semantics in which modal types *are* given denotation as closed syntax. We show how this can indeed be done, with a twist. We also use the denotation to prove some properties of the system.

1. Introduction

The box modality □ from modal logic has proven its usefulness in logic. It admits various logical and semantic interpretations in the spirit of 'we know that' or 'we can prove that' or 'in the future it will be the case that'. A nice historical overview of modal logic, which also considers the specific impact of computer science, is in [6, Section 1.7].

CMTT (contextual modal type theory) is a typed λ-calculus based via the Curry–Howard correspondence on the modal logic S4. The box modality becomes a type-former, and box types are intuitively interpreted as 'closed syntax of'.

So CMTT has types for programs that generate CMTT syntax.

Because of this, CMTT has been applied to meta-programming, but it has independent interest as a language, designed according to rigorous mathematical principles and in harmony with modal logic, which interprets □ in a programming rather than a logical context. Box types are types of the syntax of terms.

Until now this has not been backed up by a denotational semantics in which box types really *are* populated by the syntax of terms. In this paper, we do that: our intuitions are realised in the denotational semantics in a direct and natural, and also unexpected, manner.

The denotation is interesting from the point of view of the interface between logic and programming. Furthermore, we exploit the denotation to prove properties of the language, showing how denotations are not only illuminating but can also serve for new proof-methods.

* Corresponding author.
URLs: http://www.gabbay.org.uk (M.J. Gabbay), http://software.imdea.org/~aleks/ (A. Nanevski).

1570-8683/$ – see front matter
http://dx.doi.org/10.1016/j.jal.2012.07.002

1.1. Keeping it simple

This paper considers two related systems:

- The purely modal system, based on box types like $\Box A$.
- The contextual modal system, based on 'boxes containing types' like $[A_1, A_2]B$—the reader might like to think of the contextual system as a multimodal logic [15, Section 1.4] (whose modalities are themselves indexed over propositions).

Broadly speaking, the purely modal system is nicer to study but a little too simple. The contextual modal system generalises the purely modal system and gives it slightly more expressive power, but it can be a little complicated; not obscure, just long to write out.

Therefore, we open this paper with the modal system, make the main point of our denotation in the simplest and clearest possible manner—the reader who wants to jump right in and work backwards could do worse than start with the example denotations in Section 3.3.2 onwards—and then we consider the contextual system as the maths becomes more advanced. Section 2 presents syntax and typing of the modal system and Section 5 does the same for the contextual modal system; Section 3 gives modal denotations and Section 6 gives contextual modal denotations.

The developments are parallel, but not identical. Where proofs are not very different between the modal and contextual systems, we omit routine repetition. We consider reduction of the modal system in Section 4 but not reduction of the contextual system.[1] Also, we develop the important notion of *shapeliness* only for the contextual system in Section 7; it is obvious how the modal case would be a special case.

1.2. Key ideas

Our main technical results are Theorems 3.14 and 6.10, and Corollary 7.8.

However, the key technical ideas that make these results work, and indeed contribute to making them interesting, occur beforehand. It might be useful to briefly enumerate some of them here, as an aid to navigating the mathematics. Exposition is in the body of the paper:

- *Inflation* in the case of $[\![\Box A]\!]$ in Fig. 3, and the 'tail of' semantics of $X_@$ in Fig. 4. This is discussed in Remark 3.5.
- Proposition 2.23 and the fact that it is needed for soundness of the denotation in Theorem 3.14.
- The remarkable Proposition 3.13, in which valuations get turned into substitutions and closed syntax in the denotation interacts directly with the typing system. This is a kind of dual to the interaction seen in Proposition 2.23.
- The denotation of $[\![[A_i]A]\!]$ in Fig. 8, which in the context of the rest of the paper is very natural.
- The notion of *shapeliness* in Definition 7.1 and the 'soundness result' Proposition 7.7.

1.3. On intuitions

1.3.1. 'Syntax' means syntax

One early difficulty the authors of this paper faced was in communication, because we sometimes used terms synonymously without realising that the words were so slippery.

The intuition we give to $\Box A$ is self-reflectively *closed syntax of the language itself*. This is a distinct intuition from 'computations', 'code', 'values', or 'intensions', because these are not necessarily intended self-reflectively.

It is very important not to confuse this intuition with apparently similar intuitions expressed as 'code of A', 'values of A', 'computations of A', or 'intension of A'. These are not quite the same thing. It may be useful to briefly survey them here:

- 'Code of A' is an ambiguous term; this is often understood as precompiled code or bytecode, rather than syntax of the original language. See [28] for a system based on that intuition.
- 'Values of A' is a dangerous intuition and there probably should be a law against it: depending on whom one is speaking with, this could be synonymous in their mind with 'normal forms of A' (a syntactic notion) or 'denotations of A' (a non-syntactic notion).
  Matters become even worse if one's interlocutor assumes that denotations may be silently added to syntax as constants (fine for mathematicians; not so fine for programmers). More than one conversation has been corrupted by the associated misunderstandings.
- For a discussion of 'computation of A' see the Related work in the Conclusions, where we discuss how this intuition can lead to a notion of Moggi-style *monad*.
- 'Intension of A' is similar to 'syntax of A', but significantly more general: there is no requirement that the intension be syntactic, or if it is syntactic, that it be the same calculus. One could argue that 'intension of' should also satisfy that the

[1] We choose not to do this, not because it is hard, but because it is easy (apologies to John F. Kennedy).