Henkin Models of the Partial \(\lambda\)-Calculus

Lutz Schröder

BISS, Department of Computer Science, University of Bremen

Abstract. We define (set-theoretic) notions of intensional Henkin model and syntactic \(\lambda\)-algebra for Moggi’s partial \(\lambda\)-calculus. These models are shown to be equivalent to the originally described categorical models via the global element construction; the proof makes use of a previously introduced construction of classifying categories. The set-theoretic semantics thus obtained is the foundation of the higher order algebraic specification language HasCASL, which combines specification and functional programming.

Introduction

The partial \(\lambda\)-calculus has been introduced in \cite{14,15,17} as a natural generalization of the simply typed \(\lambda\)-calculus that encompasses abstraction for partial functions. Partial functions generally serve to model both non-termination and irregular termination of programs. Consequently, they feature in several specification languages such as RSL \cite{7}, SPECTRUM \cite{4}, and CASL \cite{2}. The language HasCASL \cite{20}, which offers a setting for both specification and implementation of higher order functional programs, is based on the partial \(\lambda\)-calculus. The work presented here forms the centerpiece of the semantical underpinnings of HasCASL.

The semantics of the partial \(\lambda\)-calculus as presented in \cite{14} is given in terms of models in partial cartesian closed categories (pcccs). HasCASL is designed as an extension of the first order algebraic specification language CASL, which comes with a set-theoretic semantics; this necessitates the development of a set-theoretic semantics for the partial \(\lambda\)-calculus, in the spirit of Henkin’s semantics for classical higher order logic \cite{8}. The notion of model chosen for HasCASL is that of intensional Henkin models, where partial function types are interpreted by sets that need not contain all set-theoretic partial functions and, moreover, may contain several elements representing the same set-theoretic function. This choice avoids problems such as incompleteness of deduction or non-existence of initial models, and moreover allows a more detailed analysis of the function space \cite{20}.

The central result presented here states that pccc models and intensional Henkin models are in a suitable sense equivalent, in accordance with promises made in \cite{20}. This result generalizes (and elucidates) a corresponding statement made in \cite{3} for the total \(\lambda\)-calculus. One direction of the equivalence works by

* Research supported by the DFG project HasCASL (KR 1191/7-1)
passing from a pccc model to the Henkin model determined by its global elements (i.e. the elements of a type $T$ in the Henkin model are the morphisms $1 \to T$ in the pccc model); it is non-trivial to prove that this process can indeed be reversed.

The required background w.r.t. the partial $\lambda$-calculus and pcccs is recalled in Sections 1 and 2. Section 2 also introduces the novel notion of pre-pccc. In Section 3 results on classifying categories proved in [19,18] are summarized and adapted to deal with pre-pcccs. The main novel results are presented in Sections 4 and 5. In Section 4 Henkin models are defined as first-order set-valued functors and shown to be equivalent to a notion of syntactic $\lambda$-algebra that generalizes the corresponding definition for the total case given in [3]. Moreover, we define internal languages of Henkin models, which play a crucial role in the proof of the equivalence with pccc-models carried out in Section 5. As general references for categorical concepts, we recommend [1,12].

1 The Partial $\lambda$-Calculus

The natural generalization of the simply typed $\lambda$-calculus to the setting of partial functions is the partial $\lambda$-calculus as introduced in [14,15,17]. The basic idea is that function types are replaced by partial function types, and $\lambda$-abstractions denote partial functions instead of total ones. Due to the fact that terms need not denote, equational reasoning requires some care. We will focus on existential equations here, to be read ‘both sides are defined and equal’; this will be discussed further below (Remark 5).

A signature consists of a set of sorts and a set of partial operators with given profiles (or arities) written $f : \bar{s} \to t$, where $t$ is a type and $\bar{s}$ is a multi-type, i.e. a (possibly empty) list of types. A type is either a sort or a partial function type $\bar{s} \to \to t$, with $\bar{s}$ and $t$ as above (one cannot resort to currying for multi-argument partial functions [14]). Following [14], we assume application operators in the signature, so that application does not require extra typing or deduction rules. For $\bar{t} = (t_1, \ldots, t_m)$, $\bar{s} \to \bar{t}$ denotes the multi-type $(\bar{s} \to \to t_1, \ldots, \bar{s} \to \to t_m)$, not to be confused with the (non-existent) ‘type’ $\bar{s} \to \to t_1 \times \cdots \times t_m$. A morphism between two signatures is a pair of maps between the corresponding sets of sorts and operators, respectively, that is compatible with operator profiles.

A signature gives rise to a notion of typed terms in context according to the typing rules given in Figure 1 where a context $\Gamma$ is a list $(x_1 : s_1, \ldots, x_n : s_n)$, shortly $(\bar{x} : \bar{s})$, of type assignments for distinct variables. More precisely, we speak simultaneously about terms and multi-terms, i.e. lists of terms. The judgement $\Gamma \vdash \alpha : t$ reads ‘(multi-)term $\alpha$ has (multi-)type $t$ in context $\Gamma$’. The empty multi-term $(\bar{x} : \bar{s})$, also denoted $\ast$, doubles as a term of ‘type’ $(\bar{x} : \bar{s})$, also denoted 1. When convenient, we use a context to denote the associated multi-type, as e.g. in $\Gamma \to \to t$; moreover, we write $\lambda$-abstraction in the form $\lambda \Gamma. \alpha$ where suitable.