Correct OO Systems in Computational Logic

Kung-Kiu Lau\textsuperscript{1} and Mario Ornaghi\textsuperscript{2}

\textsuperscript{1} Department of Computer Science, University of Manchester
Manchester M13 9PL, UK
kung-kiu@cs.man.ac.uk

\textsuperscript{2} Dipartimento di Scienze dell’Informazione, Universita’ degli studi di Milano
Via Comelico 39/41, 20135 Milano, Italy
ornaghi@dsi.unimi.it

Abstract. Object Oriented Design Frameworks (OOD frameworks) are groups of interacting objects. We have formalised them in computational logic as open systems of interacting objects. Our formalisation is based on steadfast logic programs in the context of open specification frameworks. However, we have considered only the static aspects, namely the specification of constraints and the correctness of queries (programs that do not update the current state). In this paper we extend this static model, by introducing actions that update the current state.

1 Introduction

In Object Oriented Analysis and Design (OOA&D), the traditional unit of design is of course the class. However, the observation that in the real world, design artefacts are rarely about single classes, but about groups of objects and the way they interact, has led to the introduction of \textit{OOD frameworks}, which represent such groups of interacting objects. It is widely recognised that OOD frameworks are better design (and reuse) units than single classes.

Different characterisations of OOD frameworks as components have been proposed, using UML (Unified Modelling Language) [19], e.g. frameworks in Catalysis [7] and \textit{Komponents} in KobrA [2]. These approaches model OOD frameworks as UML subsystems. However, they do not define OOD frameworks (or components) precisely or formally.

A natural candidate for formalising OOD frameworks in UML would be the Object Constraint Language (OCL) [22]. OCL aims to become a standard for the formal specification of OO systems. It is being formalised in its various aspects [17] and there are Java tools (e.g., [6]), but its syntax and semantics are not yet fixed in a definitive and stable way.

Independently, we have been working on the formalisation of OOD frameworks in computational logic [12, 14]. In [14] we started with a more logically oriented formalisation of OO systems, because we believe that computational logic can offer a solid base for a formalisation of OO systems (and hence OOD frameworks). Moreover, we believe that OCL would benefit from a separate and more logically oriented approach. It would inherit a lot of knowledge, like Abstract Data Types, temporal logic, proof theoretical tools, and so on. We are
Looking for a situation like the one depicted in Fig. 1, where we can design specification formalisms for OO systems, on top of a meta-theory, i.e., a sound and general underlying logic formalisation.

To formalise OOD frameworks as open systems of interacting objects, we started from our previous work on steadfast programs in the context of open specification frameworks [15]. A specification framework is an open first-order theory that formalises a class of intended models. Specification frameworks can be instantiated (or reused) by theory morphisms, like in algebraic ADT’s [23], and steadfast programs are open programs that correctly instantiate in every framework instance. Thus steadfast programs have the properties desired for the methods of correctly reusable units, and specification frameworks specifying and containing steadfast programs become good candidates for the formalisation of OOD frameworks.

In OO systems, instances are systems of interacting objects, that we call object-level instances. To give a precise definition of object-level instances, while preserving the features of open steadfast programs, we designed a three-tier approach, with an object, a specification and a meta level. The object level gives the semantics for our object-level instances. The specification level allows us to specify OO systems. The meta-level allows us to reason about OO systems, in particular, to reason about steadfast programs.

Although our aim was not the formalisation of UML, our three-level structure turns out to be very close to the multi-level approach of UML. The UML object and model levels correspond to our object and specification levels, while our meta-level has the role of the meta-model and meta-meta-model levels of UML. As a consequence, the approach developed in [14] allows us to formalise class and object UML diagrams in the context of computational logic. However, in that paper, we could only reason about the open correctness of queries, that is, open logic programs that do not update the current system state. In this paper we extend our approach by adding the ability to specify programs that update the current state. Moreover, we generalise our three-tier approach. We consider the meta level and the object level as general meta-theory of Fig. 1, and we allow many different specification formalisms, to be built on top of it.

This paper is organised as follows. First we briefly review our past work in Section 2, and we discuss the general features of our approach. In Section 3, we introduce the object level, and we discuss the object-level semantics. In Section 4, we introduce the meta-level, we introduce projections (suitable semantic maps between object and meta levels). All this is used in Section 5, to show how intermediate specification formalisms, tailored to different aspects, can be defined, together with their object- and meta-level semantics. The paper is centred