On the Essence of Oberon

David A. Naumann
Southwestern University, Georgetown, TX 78626 U.S.A.

Abstract. Reynolds described the "essence of Algol" as the simple imperative language combined with the typed lambda calculus. We provide a similar description of Wirth's language Oberon as the simple imperative language combined with procedure types and record extension. Whereas the semantics of Algol has been given in terms of a (domain theoretic) model using an explicit representation of storage, our semantics uses predicate transformers; this is possible thanks to recent advances in the theory of predicate transformers. Predicate transformer semantics connects one of the most successful methods of rigorous program development with one of the most successful pragmatically-designed programming languages.

1 Introduction

In a seminal paper [18], Reynolds argued that the essence of Algol-60 is: (a) the simple imperative language as the basis for (b) call-by-name typed lambda calculus. Of Reynolds' other principles, the only one relevant here is: (c) storable values are distinguished from commands (and meanings of other "phrase types"). This analysis has been substantiated by elegant denotational models [18, 20, 15] and by the recent result [21] that computation in such a language can be separated into two phases: elimination of procedures (using the copy rule) and then execution of the procedureless program. It seems that the call-by-name reduction strategy is essential to both denotational and operational models, so they are not easily adapted to two essential features of Oberon [22, 16]: call-by-value (and result) parameter passing, and stored procedures (i.e. procedure variables). Moreover, implementations of languages like Oberon do not separate computations into two phases; the copy rule provides the correctness criterion but not the method for implementation. This paper is a preliminary report on an analysis of "theoretical Oberon" which justifies the design of Oberon from the point of view of systematic software development methods.

Our claim is that the essence of Oberon is characterized by the following.

1. The simple imperative language —assignments, local variables, and control constructs— is given a rich type system with procedure types and limited subtyping. In contrast with (c), procedures are a data type and can be stored as well as passed as parameters.
2. The call-by-name lambda calculus —the copy rule— accounts for the binding of names to constant values of all kinds, both "constant declarations" and procedure definitions.
3. Parameters are passed by value and result, not by name. Parameter passing is derived from the other imperative features, rather than from the lambda calculus.

These claims are substantiated by our semantics of a language we call "theoretical Oberon". The semantics interprets programs as predicate transformers; not only does this justify our claims, it also shows the close link between Oberon and standard methods of program development [4, 8]. In contrast, call-by-name is not well suited to these methods (without significant modifications, as for example in Reynolds' Specification Logic [17, 19]). The book Programming in Oberon [16] uses Hoare triples to present the formal semantics of assignment and control constructs; our purpose is to show that the entire language fits well with programming calculus. We do so by embedding Oberon in a language unencumbered by implementation-oriented restrictions. (Those restrictions are judiciously chosen, but not relevant to our analysis.)

Recent progress in the theory of predicate transformers is the technical foundation for our claims, which involve type structure including higher types [12, 9, 13]. Predicate transformers are well suited to modeling imperative languages, because the notion of state is explicated without recourse to an explicit operational model of the store as is used in the Algol models of Reynolds, et al [18, 20]. Using an explicit store, it is difficult to avoid making spurious distinctions between programs; abstract semantics for Algol-like languages remains an open problem [15].

The rest of the paper is organized as follows: Section 2 contrasts Wirth's language Oberon with theoretical Oberon (unqualified, "Oberon" always means the former). Section 3 discusses the specification and development of Oberon programs. Section 4 develops the predicate transformer semantics of theoretical Oberon. Section 5 explores the most significant new problem that arises in the predicate transformer semantics, namely program types.

2 Theoretical Oberon

Oberon differs from theoretical Oberon in several ways:

- the data types include pointer types
- expressions can have side effects
- it includes a notion of module for fine control on the scope of names, to encapsulate implementations of abstract objects
- it includes function procedures
- it restricts the use of some features

Pointers. We consider pointers to be a low-level feature which is usually used in very disciplined ways, e.g. to represent recursively defined types and unique identities of objects. We are not aware of programming methods with substantial general treatments of pointers; they are usually reduced to arrays, as they are in standard denotational semantics. Since we want to view Oberon as a subset