Higher-Order Order-Sorted Algebras

Zhenyu Qian

FB3 Dept. of Computer Science, University of Bremen
D-2800 Bremen 33, Fed. Rep. of Germany
E-mail: qian@informatik.uni-Bremen.de

Abstract: The aim of this paper is to present a new semantics of higher-order order-sorted types for functional programming, data type specification and program transformation. Our type discipline unifies higher-order functions, overloading and subtype polymorphism in a very simple way. The new approach can be considered as an extension of order-sorted algebra with higher-order functions. We show the existence of initial algebras and give a sound and complete equational deduction system.

1. Introduction

Specifications by many-sorted algebras have become one of the main lines for data type specifications since this approach was proposed by Guttag [75] and Goguen, Thatcher and Wagner [78] (cf. also [Ehrig and Mahr 85]). To model subtype polymorphism in data type specifications, Goguen [78], Gogolla [84], Smolka, Nutt, Goguen and Meseguer [87], Goguen and Meseguer [89] have extended many-sorted algebra to order-sorted algebra. Two significant new features are added in the extension: a partial ordering over the sorts, which is interpreted as the subset relation, and multiple function declarations, which are interpreted as the functions consistently yielding the same values for the same arguments. The interpretation of the partial ordering on sorts has been recently generalized to conversion functions by Kreowski and Qian [89], following the original idea of Reynolds [80].

In this paper, we extend order-sorted algebra with higher-order functions.

The introduction of higher-order functions has been a major consideration in the realm of functional programming. There, functions are first considered to be values, which can be passed as arguments to other functions and returned as results, and in particular, whose equivalence should be clearly defined. Second, functions can be applied as operations building new functions from existing ones. The definition of the equivalence of functions is basic for transforming programs and solving equations whose unknowns are functions (cf. [Krieg-Brückner 89]).

Introduced by Cardelli [84], functional programming over subtypes has been the subject of research for a few years. However, most of the existing work has only captured the application aspect of functions. More precisely, let types s be interpreted as sets [s] of values. One has only required that all function values $\phi \in [d \rightarrow r]$ in the semantic set of a function type $d \rightarrow r$ should satisfy $\text{dom}(\phi) \supseteq [d]$ and $\forall a \in [d], \phi(a) \in [r]$. This is enough to guarantee the applicability of functions of $d \rightarrow r$ to values of $d$, but too restrictive to capture the equivalence of functions of $d \rightarrow r$.

For example, let $\text{int}$ be the type of integers and $\text{nat}$ the type of natural numbers. Let $id$ be the identity function on $[\text{int}]$, and $\text{abs}$ the function on $[\text{int}]$ yielding the absolute values of integers. Then $id, abs \in [\text{nat} \rightarrow \text{int}]$. Although $id(n) = abs(n)$ for each $n \in [\text{nat}]$, they are two distinct functions. In general, the question of whether two functions of the type $d \rightarrow r$ are equal may not be answered by considering their behaviors on $d$.

1 The research has been partially supported by the Commission of the European Communities under the ESPRIT Programme in the PROSPECTRA Project, ref #390, and the ESPRIT Basic Research Action COMPASS, ref #3264.
We want to change the semantics of function types so that the equivalence of functions of \( d \rightarrow r \) can be clearly defined by their behaviors on \( d \). The new semantics should capture both value and application aspects of functions, and therefore provide a common foundation for functional programming, data type specification, and especially program synthesis and transformation.

Different from [Cardelli 84] and [Cardelli and Wegner 85] where \( \lambda \)-calculus-based models are used, we choose to use the algebras described by (conditional) equations on the terms of functions. The reason is that this framework is simple and rich enough for our purpose.

The extensions of many-sorted algebra with higher-order functions have been studied, among others, by Maibaum and Lucena [80], Parsaye-Ghomi [81], Poigne [86] in the context of category theory, and by Möller [87] within the algebraic framework. Especially, two important aspects have been considered by the last two authors which are of great importance for program transformation. The first is the property of extensionality, which guarantees the semantic equality of two programs whenever they behave the same; the second is a sound and complete calculus, which guarantees the derivability of the semantic equality of functions.

We will formulate our semantics in an elementary way and do not assume familiarity with category theory. This semantics unifies higher-order functions, ad-hoc and subtype polymorphism (cf. [Cardelli and Wegner 85] or [Goguen and Meseguer 89] for classifications of polymorphism). Partial (higher-order) functions are supported, since total functions on subtypes can be considered to be partial on their supertypes. Furthermore, we give a sound and complete equational deduction.

This paper is a compact version of a part of [Qian 90]. We organize this paper as follows: Section 2 introduces a function restriction operation. A new semantics for higher-order subtypes is motivated in Section 3. Section 4 tries to formulate an natural semantics of higher-order subtypes by a notion of extensional higher-order order-sorted algebras. Section 5 introduces higher-order order-sorted algebras. Well-typed terms of a corresponding logic are defined in Section 6. A sound and complete calculus is given in Section 7. Section 8 contains some illustrative examples. Conclusions are stated in Section 9.

2. Functions and function restrictions

Let \( U \) be a set. For \( n \geq 0 \), the set of all \( n \)-tuples of elements of \( U \) is denoted by \( U^n \) or by \( U \times \ldots \times U \) with \( n \) times \( U \). For \( a \in U^n \), the \( i \)-th component, \( 1 \leq i \leq n \), of \( a \) is denoted as \( a_i \). In the case \( n=0 \) we have \( U^0 = \{ () \} \), where () denotes the 0-tuple. For \( A_1, \ldots, A_n \subseteq U \), \( n \geq 0 \), \( A_1 \times \ldots \times A_n \) denotes the set \( \{ a \in U^n | a_i \in A_i \text{ for } 1 \leq i \leq n \} \).

A \( (n \text{-ary}) \) function \( \phi \) is a subset of \( U^{n+1} \), \( n \geq 0 \), that does not contain two tuples having the same first \( n \) components. The domain of \( \phi \), denoted as \( \text{dom}\phi \), is the set \( \{ (a_1, \ldots, a_n) | a \in \phi \} \). The range of \( \phi \), denoted as \( \text{ran}\phi \), is the set \( \{ a^{n+1} | a \in \phi \} \). Furthermore, \( \text{dom}^i \phi \) denotes \( \{ a^i | a \in \phi \} \), the set of the first \( i \) components of the domain.

In the case \( n=0 \) the above definition merely states that \( \phi \) is an one-element set \( \{ (a) \} \) consisting of an 1-tuple \( (a) \) with \( a \in U \). This kind of 0-ary functions \( \{ (a) \} \) may also be called constants and identified with the value \( a \).

If some \( D_i = \emptyset \) then \( \phi = \emptyset \), which is called empty function.

Let \( D_1, \ldots, D_n \subseteq U \), \( n \geq 0 \). A function \( \phi \) may be denoted as \( \phi : D_1 \times \ldots \times D_n \rightarrow R \) if for each \( b \in D_1 \times \ldots \times D_n \) there is exactly one tuple \( a \in \phi \) such that \( a_i = b_i \), \( 1 \leq i \leq n \). In this case, \( \text{dom}\phi = D_1 \times \ldots \times D_n \) and \( \text{ran}\phi \subseteq R \). Note that according to the above definition, \( \phi : D_1 \times \ldots \times D_n \rightarrow R \) for any \( R \) with \( R \supseteq R \) denotes the same function as \( \phi : D_1 \times \ldots \times D_n \rightarrow R \).