Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

An algebraic approach to computations with progress

Walter Guttmann

Department of Computer Science and Software Engineering, University of Canterbury, New Zealand

A R T I C L E   I N F O

Article history:
Received 27 September 2014
Received in revised form 17 November 2015
Accepted 21 November 2015
Available online xxxx

Keywords:
Approximation order
Axiomatic program semantics
Iteration algebras
Relations
Time
Traces

A B S T R A C T

The notion of progress appears in various computation models, for example, in the form of traces getting longer, passing of real time, incrementing a counter, going from termination to non-termination. We introduce a model of sequential computations that generalises and abstracts these examples. We generalise existing algebras for non-terminating executions and instantiate these with our model. Using these algebras we derive an approximation order for computations with time and for trace-based computations. We introduce a generalisation of omega algebras to express iteration in the new model.

1. Introduction

A computation model is a mathematical description of what happens when a program is run on a computer. Such models facilitate the construction of correct programs by mathematical calculation, in addition to the experimental method of testing programs. Models differ in the kinds of computation they can represent and the precision they achieve.

In this paper we consider models for sequential computations. An example of such a computation model are binary relations: the starting state of a computation is related to the possible final states. Having several possible final states is useful for specifications that leave freedom to the programmer. Moreover, when a program is run it might abort due to an error or it might fail to terminate; extended relational models take into account these phenomena [2,24,10,20,14].

Relations describe the input–output behaviour of a computation. This provides only a simplistic model of progress: a computation will be in its starting state before it will be in its final state. How long a computation takes to go from one state to another, or which intermediate states it goes through, is not represented by the input–output relation.

Models which include a notion of time or traces take care of these phenomena [23,20,21,11]. Time can be modelled by adding an extra variable to the state space; in this case, progress means that the value of this variable increases. A trace can describe the history of a computation, that is, the sequence of states from its start to the current state; in this case, progress means that the trace gets longer. Different domains of time can be used for modelling real time or an abstraction such as the number of execution steps. Traces can be combined with timing information to show when a computation passes through a state.

E-mail address: walter.guttmann@canterbury.ac.nz.

http://dx.doi.org/10.1016/j.jlamp.2015.11.009
2352-2208/© 2015 Elsevier Inc. All rights reserved.

All of these computation models support a notion of progress. The aim of the present paper is to distil this concept. To this end we construct a general computation model that captures progress and instantiates to the various models mentioned above. Algebras and algebraic methods that have previously been developed and refined for several relational computation models provide the technical means for this work [14–18]. The general motivation for this work is to understand how different computation models that support a notion of progress are related and to obtain a common theory. The reason for using algebras is that they facilitate such a unification [24,12,14,15,18], are well supported by theorem proving technology [26,27] and yet powerful enough to yield complex results which have been applied in the development and verification of programs [1,5].

The models discussed in this paper support non-deterministic computations, which are useful for specification purposes. Accordingly, a computation is made up of a set of executions, each of which describes one possible behaviour of the computation. In previous works we have distinguished finite executions, which terminate successfully, aborting executions, which fail due to an error, and infinite executions, which do not terminate. In the present paper we refine the latter kind of executions into two classes, which we call (potentially) incomplete executions and (actually) infinite executions. While incomplete executions arise as approximations to the semantics of recursion and as non-terminating but unproductive executions, infinite executions yield actually infinite traces or unbounded progress in time.

Our computation model thus distinguishes four kinds of execution: finite, aborting, incomplete and infinite. These kinds of execution are represented by relations over a state space, which carries the values of program variables and additional information such as time or traces. Progress is modelled by a preorder on the state space; because this is also a relation, it integrates nicely with the various kinds of execution. The relations are collected in matrices, which generalise representations used in previous works [16,21,11,17,18]. The matrices facilitate the use of well-known constructions for calculating various operations that serve as the basis for program constructs [31,19].

We prove that these operations satisfy the axioms of algebras which have previously been used to describe choice, conjunction, sequential composition and various forms of iteration [28,5,14,15]. Moreover, we introduce algebras that describe the incomplete and infinite executions; they generalise algebras which have previously been used to describe the states from which such executions exist [14,18]. By instantiating these algebras we automatically inherit hundreds of properties that have previously been derived using interactive and automated theorem provers. We also obtain an approximation order for our computations, which is the key ingredient for the semantics of recursive programs; in particular, this covers while-loops. We express the necessary fixpoints in this approximation order in terms of fixpoints in the less complex refinement order.

The contributions of the present paper are as follows:

- A new computation model that describes progress. It generalises previous models which feature progress in the form of time or traces or by distinguishing terminating and non-terminating executions.
- A distinction between potentially incomplete and actually infinite executions. The latter are represented by heterogeneous relations.
- A new algebra for incomplete and infinite executions. It generalises previous algebras that describe the states from which such executions exist.
- Instances of the new algebra and previously introduced algebras for iteration, consequences of which include separation and refinement theorems and various program transformations.
- Capped omega algebras, which generalise omega algebras to describe fixpoints of bounded affine functions.

Section 2 recalls the basic algebraic structures used in the remainder of this paper and basic properties of relations. Section 3 introduces our model of computations with progress and shows how it generalises previous models. Section 4 defines the basic operations of choice, conjunction and sequential composition. Section 5 introduces an algebra with an operation that describes the incomplete and infinite executions, and instantiates it with our model of computations with progress. Section 6 obtains an approximation order for the semantics of recursion for our computations. Section 7 introduces capped omega algebras and instantiates previous algebras for iteration.

All algebraic structures axiomatised in this paper, but not the concrete models, have been implemented in Isabelle/HOL [33], making heavy use of its integrated automated theorem provers and SMT solvers [34,3]. The theories contain proofs of Theorems 3, 4, 6 and 8 as well as hundreds of other properties including, for example, a fixpoint calculus, separation theorems and Back's atomicity refinement theorem [14,15,18]. By instantiating these algebras we automatically inherit these results. The Isabelle/HOL theory files are available at http://www.csse.canterbury.ac.nz/walter.guttmann/algebra/.

2. Algebraic structures for sequential computations

In this section we axiomatise the operations of non-deterministic choice, conjunction and sequential composition, and various forms of iteration featured by many computation models. Our presentation follows [17]. We also recall basic definitions and properties of relations.