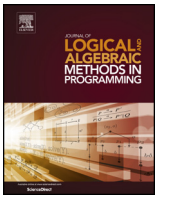
Journal of Logical and Algebraic Methods in Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Developments in concurrent Kleene algebra

Tony Hoare [a], Stephan van Staden [b], Bernhard Möller [c], Georg Struth [d], Huibiao Zhu [e,*]

[a] *Microsoft Research, Cambridge, UK*
[b] *Department of Computer Science, University College London, UK*
[c] *Institut für Informatik, Universität Augsburg, Germany*
[d] *Department of Computer Science, The University of Sheffield, UK*
[e] *Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China*

ARTICLE INFO

Article history:
Received 10 October 2014
Received in revised form 14 September 2015
Accepted 28 September 2015
Available online xxxx

Keywords:
Concurrent Kleene algebra
Laws of programming
Trace algebra
Semantic models
Refinement
Unifying theories

ABSTRACT

This report summarises the background and recent progress in the research of its co-authors. It is aimed at the construction of links between algebraic presentations of the principles of programming and the exploitation of concurrency in modern programming practice. The signature and laws of a Concurrent Kleene Algebra (CKA) largely overlap with those of a Regular Algebra, with the addition of concurrent composition and a few simple laws for it. They are re-interpreted here in application to computer programs. The inclusion relation for regular expressions is re-interpreted as a refinement ordering, which supports a stepwise contractual approach to software system design and to program debugging.

The laws are supported by a hierarchy of models, applicable and adaptable to a range of different purposes and to a range of different programming languages. The algebra is presented in three tiers. The bottom tier defines traces of program execution, represented as sets of events that have occurred in a particular run of a program; the middle tier defines a program as the set of traces of all its possible behaviours. The top tier introduces additional incomputable operators, which are useful for describing the desired or undesired properties of computer program behaviour. The final sections outline directions in which further research is needed.

1. Introduction

Concurrency has many manifestations in computer system architecture of the present day. It is provided in networks of distributed systems and mobile phones on a world-wide scale; and on a microscopic scale, it is implemented in the multi-core hardware of single computer chips. In addition to these differences of scale, there are many essential (and inessential) differences in detail. As in other areas of basic scientific research, we will postpone consideration of many interesting variations, and try to construct a mathematical model which captures the essence of concurrency at every scale and in all its variety.

Concurrency also has many manifestations in modern computer programming. It has been embedded into the structure of numerous new and experimental languages, and in languages for specialised applications, including hardware and net-

* Corresponding author.
E-mail addresses: t-tohoar@microsoft.com (T. Hoare), s.vanstaden@cs.ucl.ac.uk (S. van Staden), bernhard.moeller@informatik.uni-augsburg.de (B. Möller), g.struth@sheffield.ac.uk (G. Struth), hbzhu@sei.ecnu.edu.cn (H. Zhu).

http://dx.doi.org/10.1016/j.jlamp.2015.09.012
2352-2208/

work design. It is provided in more widely used general-purpose languages by a choice of thread packages and concurrency libraries. Further variation is introduced by a useful range of published concurrency design patterns, from which a software architect can select one that reconciles the needs of a particular application with the capabilities and performance of the particular hardware available for implementation. Our laws and principles will be illustrated by examples drawn from many of these sources, and we will refrain from designing any particular language of our own.

Concurrency is also a pervasive phenomenon of the real world in which we live. Consequently, a general mathematical model of concurrency shares many concepts and principles with human understanding of the real world. For example, we model the behaviour of an object engaging together with other objects in events of various kinds that occur at various points in space and at various instants in time. We observe also the fundamental principle of causality, which states that no event can occur before an event on which it causally depends. Another principle is that of separation, which states that separate objects occupy separate regions of space. It is these principles that guide definitions of sequential and concurrent composition of programs in a model of Concurrent Kleene Algebra (CKA) [37].

These principles also provide evidence for a claim that CKA is an algebraic presentation of common-sense (non-metric) spatial and temporal reasoning about the real world. Thus the algebra may be used to define and explore the behaviour of a computer system embedded in its wider environment, including even human users. The investigation of space and time has long been the province of philosophers, including St. Thomas Aquinas, William of Ockham, and Aristotle. We would like to think of this work as a contribution to that tradition.

1.1. Application

The purpose of CKA is to support reliable predictions about the behaviour of computer programs when executed. It presents a set of very general algebraic laws, which are intended to be applicable to the execution of all computer programs that can ever be specified, designed, implemented in its notations. The free variables of the laws therefore stand for programs, but they are equally valid for program executions and program specifications. The operators of the algebra describe the way in which programs can be composed out of other smaller subprograms. The constants (or generators) of the algebra stand for the basic or atomic commands of the program. The algebraic laws express programming concepts and principles, which feature (under various notational disguises) in widely used computer programming languages of the present day.

Algebraic laws play a fundamental role in expressing the principles that underlie many branches of human knowledge. For example the laws of arithmetic embody the principles of numerical calculation, and the algebraic axioms of group theory serve as a definition of this branch of mathematics. Fundamental discoveries in the natural sciences are often expressed as laws, which later find application in engineering. Maxwell's equations of electro-magnetism are essential in the design of the electronic components of computers, just as Boole's Laws of thought are the basis of the design of computing circuits implemented electronically. The laws of programming should perhaps be recognised as providing a similar foundation for Computer Science and its application in Software Engineering.

Algebraic laws are expressed in a very limited subset of mathematical notations. They take the form of equations (or inequalities) between terms expressed by an explicitly restricted set of functions and constants. The variables in the laws are all (implicitly) quantified universally. In addition to the axioms of transitivity and reflection, the only required rules of reasoning are the instantiation of the variables and the substitution of terms already proven to be equal or to be related by an ordering. That is why computers and even humans are so good at algebraic reasoning and some humans even appreciate its sparse elegance.

It is remarkable how many important ideas of natural science and mathematics are definable within the limitations of algebra. The universality of algebra make it applicable not only to many different phenomena in our own universe, but also to many alternative universes. Consequently, there are many features and properties of our universe that cannot be expressed algebraically. Newton's laws of motion describe the orbits of the planets in general, but they do not predict the position in the sky at which any particular planet of our own solar system can be seen at any given time. For this, it is necessary to know the mass and momentum of the heavenly bodies, and their distance from each other. This information is given by Kepler's mathematical model of the planetary system. This may be illustrated by a working model, an orrery, perhaps made in brass and driven by cogs and clockwork instead of gravity and momentum.

In any conventional scientific theory, the relevant laws are supported by mathematical models of the behaviour of aspects of one or more parts of the physical universe, which have been shown by observation and experiment to satisfy the laws. Similarly, in mathematics, the Laws of Arithmetic are supported by discovery of a set-theoretic model of the numbers to which the laws apply. The achievement of research into the Foundations of Mathematics has been to discover this model, and to prove that it satisfies the laws.

1.2. A hierarchy of models

The standard model of arithmetic is constructed in a number of tiers. For example, the real numbers occupy the top tier, then the fractions and the integers, with the natural numbers at the bottom tier. Each tier is defined in terms of the next tier below it; for example, reals are defined as downward closed sets of fractions (Dedekind cuts), and fractions are defined as pairs of integers. The operators applicable to each kind of number are similarly defined in terms of the operators of the next tier. Numbers in the higher tiers have a larger range of operators defined on them, for example, exact division

Please cite this article in press as: T. Hoare et al., Developments in concurrent Kleene algebra, Journal of Logical and Algebraic Methods in Programming (2015), http://dx.doi.org/10.1016/j.jlamp.2015.09.012