Refinement algebra for probabilistic programs

Larissa Meinicke¹ and Kim Solin²

¹ Åbo Akademi, Åbo, Finland. E-mail: larissa.meinicke@abo.fi
² Uppsala University, Uppsala, Sweden. E-mail: kim.solin@filosofi.uu.se

Abstract. We identify a refinement algebra for reasoning about probabilistic program transformations in a total-correctness setting. The algebra is equipped with operators that determine whether a program is enabled or terminates respectively. As well as developing the basic theory of the algebra we demonstrate how it may be used to explain key differences and similarities between standard (i.e. non-probabilistic) and probabilistic programs and verify important transformation theorems for probabilistic action systems.

Keywords: Refinement algebra, Probability, Kleene algebra, Action systems, Data refinement, Atomicity refinement

1. Introduction

Probabilistic programs, programs in which probabilistic choices may be made, have applications in areas such as distributed computing and reliable systems modelling. In order to be able to reason about such programs one would like a technique for understanding how and when it is possible to transform one probabilistic program into another while preserving some notion of correctness. For example, suppose we have a program

\[\text{do } e \sqcap (a \text{ do } b \text{ od}) \sqcap l \sqcap r \text{ od}\] (1)

in which \(e, a, b, l\) and \(r\) represent programs which may include discrete probabilistic choices (we use \(x \oplus y\) to denote the discrete probabilistic choice in which \(x\) is executed with probability \(p\) and \(y\) is taken with probability \(1 - p\)), in addition to the more commonplace operators sequential composition, \(;\), and nondeterministic choice, \(\sqcap\), which may be used to represent design freedom in specifications, or uncertainty. These programs may implicitly contain a guard, which denotes when they are enabled to be executed. Program (1), which we shall refer to as a probabilistic action system [ST96], may be used to represent the concurrent execution of atomic actions, \(e, (a \text{ do } b \text{ od}), l\) and \(r\) by an unfair scheduler: on each iteration the scheduler nondeterministically selects an enabled action for execution; it continues execution indefinitely, or until all of the actions become disabled. We may like to know under what circumstances is it possible to replace Program (1) by another, say

\[\text{do } e \sqcap a \sqcap b \sqcap l \sqcap r \text{ od}\] (2)

in which atomic action \((a \text{ do } b \text{ od})\) has been decomposed, thereby increasing the amount of parallelism in the system. Theorems for reasoning about transformations of this kind are often referred to as separation and reduction or atomicity refinement theorems, and have been shown to play a useful role in the development and

Correspondence and offprint requests to: L. Meinicke and K. Solin, E-mail: larissa.meinicke@abo.fi; kim.solin@filosofi.uu.se
verification of non-probabilistic distributed systems [Doe77, Lip75, LS89, Bac89, BvW99, Coh00]. We may also be interested in knowing when, and in what context, Program (2) may be replaced by replaced by another

\[ \text{do } e' \cap a' \cap b' \cap l' \cap r' \text{ od} \]

which uses a different data representation. Theorems of this variety, data refinement theorems, may be used to develop abstract program specifications into concrete program implementations.

The answers to these questions are not as straightforward and simple as they may seem. Although transformation rules for non-probabilistic programs are well understood, transformation rules for probabilistic programs differ in subtle, but important ways from their better-known non-probabilistic counterparts [MW05, MCM06, MH06, MH08b, Mei08]. A technique for justifying and explaining such transformations should aspire to being both reliable and uncomplicated to use. Which methods are at hand?

Abstract algebra has a solid mathematical underpinning and simultaneously provides a perspicuous notation that allows for simple symbol pushing instead of tedious model-theoretic reasoning. Several examples have demonstrated how abstract algebra may be employed as an efficient tool for reasoning about programs. For example, in early work, Kozen used Kleene algebra with tests for proving transformation rules of loops in a partial-correctness framework [Koz97]. Solin and von Wright have then used refinement algebras for program reasoning in a total-correctness environment [vW02, vW04, SvW06, Sol06]. In addition to providing a suitable level of abstraction at which to perform and explain proofs, such algebras are a convenient way to describe similarities and dissimilarities between different program models and to reuse results proved in an algebra over different models for which the axiomatisation is sound. Recent results have also shown that abstract-algebraic proofs may be simple to automate [HS08].

McIver et al. [MW05, MCM06] and Meinicke and Hayes [MH06, MH08b, Mei08] have identified that algebraic reasoning also works well where probabilistic programs are concerned. In their work, McIver et al. [MW05, MCM06] identified a relaxation of Kleene algebra suitable for probabilistic programs, known as probabilistic Kleene algebra, and used it to derive a probabilistic separation and reduction theorem which has been applied in the verification of a protocol in [MCM06]. This algebra, like Kleene algebra, contains a weak iteration (or Kleene star) operator, \( * \), which may be used to represent finite (or terminating) iterations, but it does not contain operators for representing possibly infinite iterations. For this reason, we say that it is suitable for reasoning about partial—but not total—program correctness. In related work, Meinicke and Hayes [MH06, MH08b, Mei08] explored algebraic properties of probabilistic programs within a total-correctness framework, in which possibly infinite iterations are expressible, and used them to derive transformation theorems for probabilistic loops and action systems within a particular program model.

In this paper, we identify and explore a very general abstract algebra for reasoning about probabilistic programs in a total-correctness framework. That is, we lift the concrete-algebraic approach to probabilistic programs of Meinicke and Hayes [MH06, MH08b] to a more abstract level, in the same way that Solin and von Wright [vW02, vW04, SvW06] lifted the concrete-algebraic approach to non-probabilistic programs of Back and von Wright [BvW99]. Unlike probabilistic Kleene algebra [MW05], this algebra contains both operators for expressing terminating and possibly non-terminating iterations. We consider the ability to express programs that are possibly non-terminating as important: in non-reactive programs a non-terminating loop is a classical programming error. Also, reactive programs—programs in which the behaviour over time is visible—may exhibit non-terminating behaviours.

One very important feature of this algebra is its simplicity. Like probabilistic Kleene algebra, the algebra has operators to represent sequential composition, choice and iterations, but it does not contain a probabilistic choice operator, or other probabilistic-program specific attributes. This decision reflects an important observation: many non-trivial transformation rules for probabilistic systems, such as the data refinement and separation and reduction rules we derive in Sect. 8, may in fact be specified and verified without having to reason directly about probabilistic choices or other probabilistic-program specific attributes. The generality of the algebra not only allows us to hide unnecessary details, but it allows us to use the algebra to capture similarities between different models—both probabilistic and non-probabilistic. This implies that results verified in the algebra are valid across a range of models, including

- Non-reactive non-probabilistic program models like the isotone predicate transformers which may be used to model programs with two forms of nondeterministic choice, angelic and demonic choice, in a total-correctness framework. Given a game-theoretic interpretation, demonic nondeterministic choice, \( \cap \), represents a choice which cannot be controlled by someone observing the execution of a program, and which, if possible, will be made to her disadvantage, while an angelic choice, \( \cup \), can be favourably influenced by the same observer in