Science of Computer Programming 92 (2014) 179–210

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Refinement algebra with dual operator

Viorel Preoteasa

Åbo Akademi University, Department of Information Technologies, Joukahaisenkatu 3-5 A, 20520 Turku, Finland

HIGHLIGHTS

- We introduce an extension of the general refinement algebra with a dual operator.
- We introduce assertions and assumptions with simpler definitions than before.
- We defined the termination and enabledness operators in our algebra.
- We prove data refinement and Hoare rules, and we used them for an example program.
- All results are formalized in the Isabelle theorem prover.

ARTICLE INFO

Article history:
Received 10 April 2012
Received in revised form 24 June 2013
Accepted 1 July 2013
Available online 2 August 2013

Keywords:
Algebra of programming
Refinement algebra
Refinement calculus
Data refinement
Hoare logic

ABSTRACT

Algebras of imperative programming languages have been successful in reasoning about programs. In general an algebra of programs is an algebraic structure with programs as elements and with program compositions (sequential composition, choice, skip) as algebra operations. Various versions of these algebras were introduced to model partial correctness, total correctness, refinement, demonic choice, and other aspects. We introduce here an algebra which can be used to model total correctness, refinement, demonic and angelic choice. The basic model of our algebra are monotonic Boolean transformers (monotonic functions from a Boolean algebra to itself).

1. Introduction

Proving correctness of programs started with the seminal work of Floyd [1] and Hoare [2]. Floyd has introduced the so-called intermediate assertion method and using it he was proving both *functional correctness* of programs as well as *termination*. Hoare introduced the axiomatic method for proving correctness of programs. Hoare's original method did not treat the termination of programs but only their *partial correctness*. Partial correctness of a program asserts that if the program terminates, then the result calculated by the program is correct. Dijkstra introduced in [3] the weakest precondition of a program S and a postcondition q on the state of the program, as the set of the initial states from which the program S always terminates and it terminates in a state from q. Using weakest preconditions one could prove the *total correctness* of programs, that is the program terminates, and the result of the computation is correct.

Refinement calculus [4–7] is a further development of the weakest precondition theory. It is a calculus based on monotonic predicate transformers (monotonic functions mapping predicates to predicates) suitable for program development in a total correctness framework. Within this calculus various aspects of imperative programming languages can be formalized. These include total correctness, partial correctness, demonic choice, angelic choice, and unbounded nondeterminism. In a demonic choice the user of a program does not have control over the choice that is made. For example the user does not have control over the choice made by an operating system in scheduling process, or in allocating memory to a program. On

E-mail address: viorel.preoteasa@abo.fi.

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.002

the other hand in angelic choice, the user can influence the execution of the program by selecting the appropriate actions from all possible. An example of angelic choice can be the menu actions from a program with a graphical user interface. The user has full control over which action to select from those available. In unbounded nondeterminism the number of choices is infinite. Unbounded nondeterminism can be both demonic and angelic and it can occur in practice if we have for example parallel programs with the assumption of fairness [8].

Refinement calculus has introduced a simplified collection of primitive statements compared to the primitive statements found in imperative programming languages. Among them we have assertions ($\{p\}$ – assert p), assumptions ($[p]$ – assume p), demonic choices ($S \sqcap T$), angelic choices ($S \sqcup T$), iterations (S^ω), and others. The statement $\{p\}$, where p is a predicate (a condition on the values of the program variables), skips if p is true for the starting state and it fails (it does not terminate) otherwise. On the other hand $[p]$ skips when p is true, and terminates *miraculously* otherwise. A miraculous program is always correct, but it cannot be implemented. Using these primitive statements we can define the usual *if* and *while* program constructs.

In refinement calculus, the angelic and demonic choices are dual to each other in a way similar to conjunction and disjunction in logic, or existential and universal quantification. Within refinement calculus we can define a dual operator [9–11,6] which would map demonic choice into angelic choice and vice versa. For the application of the dual operator to the program $(x := 2) \sqcap (x := 3)$ returns the program $(x := 2) \sqcup (x := 3)$. The assertions and assumptions are also dual to each other and the dual of $\{p\}$ is $[p]$.

The original approach [8] of using predicate transformers for modeling programs required them to satisfy a number of healthiness conditions like *strictness* ($S(\mathsf{false}) = \mathsf{false}$), *conjuctivity* ($S(p \wedge q) = S(p) \wedge S(q)$), and *continuity*. These conditions were imposed because practical programs satisfy them, and the semantics of the programing constructs was easier to introduce. However, strictness excludes miracles, conjuctivity excludes programs with angelic choice, and continuity excludes unbounded nondeterminism.

Abstract algebra is a useful tool in mathematics. Rather than working with specific models like natural numbers and algebra of truth values, one could reason in a more abstract setting and obtain results which are more general and applicable in different models. Algebras of logics are very important tools in studying various aspects of logical systems. Algebras of programming theories have also a significant contribution to the simplification of reasoning about programs, and they abstract further the notions of program correctness. Programs are elements of an algebra and program compositions and program constants (sequential composition, choice, iteration, skip, fail) are the operations of the algebra. These operations satisfy a number of relations which are used for reasoning about programs.

Kleene algebra with tests (KAT) [12] is an extension of Kleene algebra [13] and it is suitable for reasoning about programs in a partial correctness framework. In KAT, tests are similar to assumptions from refinement calculus. In case a test p is not true, then p does not terminate, and because we are in a partial correctness framework this is equivalent to a miracle. Always, a nonterminating program is partially correct.

Various versions of Kleene algebras have been introduced, ranging from Kleene algebra with domain [14] and concurrent Kleene algebra [15] to an algebra for separation logic [16]. Kleene algebra with modal operators has been used in [17,18] to model partial correctness as well as different notions of termination.

Demonic refinement algebra (DRA) was introduced in [19,20] as a variation of KAT to allow also reasoning about total correctness. The intended model of DRA is the set of conjunctive predicate transformers and this algebra cannot represent angelic choice. General refinement algebra (GRA) was also introduced in [20], but few results were proved and they were mostly related to iteration. Although the intended model for GRA is the set of monotonic predicate transformers, GRA does not include the angelic choice operator. Both DRA and GRA have been further extended with enabledness and termination operators in [21] and [22], respectively. More properties of GRA were proved in [22] and it was used for probabilistic programs. In [23], a refinement algebra with negation operator is introduced, and in [24] the dual operator is defined using the negation. The negation operator does not preserve monotonicity, and the iteration operators are introduced only for monotonic elements of the algebra.

The contribution of this paper is an extension of GRA with a dual operator. The intended model for our algebra is the set of monotonic Boolean transformers (monotonic functions from a Boolean algebra to itself). In GRA assertions (assumptions) are introduced as disjunctive (conjunctive) elements which have complement. Formally this definition requires an existential quantifier. Using the dual operator we characterize these assertions (assumptions) using a conjunction of (in)equations which is simpler than the usual definition from GRA and KAT, and we do not need an existential quantifier. We prove that the assertions (assumptions) form a Boolean algebra. Moreover, we also prove that the assertions defined in the algebra are exactly the program assertions in the model of monotonic Boolean transformers. Having the dual operator and the demonic choice operator we automatically obtain also the angelic choice operator. In [21,22] the enabledness and termination operators are introduced using axioms for DRA and GRA respectively. The termination operator applied to a program returns the states from which the program always terminates, and the enabledness operator applied to a program returns the states from which the program does not terminate miraculously. These operators can be defined in our algebra, and their axioms can be proved as theorems. The elements of our algebra correspond to the modal operators over a Kleene algebra from [17,18].

In DRA [20], a pre-post specification statement is introduced and it is used to prove that a program refines a pre-post specification statement if and only if the program is correct with respect to the pre and post conditions. The proof of this fact requires the assumption that all programs are conjunctive, a fact which does not hold for arbitrary monotonic predicate