Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A new roadmap for linking theories of programming and its applications on GCL and CSP

Jifeng He, Qin Li [*]

Shanghai Key Laboratory of Trustworthy Computing, International Research Center of Trustworthy Software, East China Normal University, Shanghai, China

A R T I C L E I N F O

Article history:
Received 25 December 2016
Received in revised form 12 October 2017
Accepted 18 October 2017
Available online xxxx

Keywords:
Linking theories of programming
Formal semantics
Program algebra
Guarded Command Language
Communicating Sequential Processes

A B S T R A C T

Formal methods advocate the crucial role played by the algebraic approach in specification and implementation of programs. Traditionally, a top-down approach (with denotational model as its origin) links the algebra of programs with the denotational representation by establishment of the *soundness* and *completeness* of the algebra against the given model, while a bottom-up approach (a journey started from operational model) introduces a variety of bisimulations to establish the equivalence relation among programs, and then presents a set of algebraic laws in support of program analysis and verification. This paper proposes a new roadmap for linking theories of programming. Our approach takes an algebra of programs as its foundation, and generates both denotational and operational representations from the algebraic refinement relation. This new approach is applied in this paper to GCL (Guarded Command Language) and CSP (Communicating Sequential Processes) to link their various semantical representations based on their algebraic semantics.

1. Introduction

Formal methods advocate the crucial role played by the algebra of programs in specification and implementation of programs. Study leads to the conclusion that both the top-down approach (with denotational model as its origin) and the bottom-up approach (a journey started from operational model) can meet in the middle:

- Top-down approach usually begins with construction of a specification-oriented model [1,2,4,12,17], then links the algebra of programs with the denotational framework by establishment of the *soundness* and *completeness* of the algebra [10,15] against the given model.
- Bottom-up approach starts with an operational semantics [14] and introduces a rich variety of bisimulations [7,13] to identify the equivalence relation among programs, and then presents a set of algebraic laws in support of program analysis and verification.

This paper proposes a new roadmap for linking theories of programming. Other than the top-down and bottom-up approach, our approach starts from the middle: it takes an algebra of programs as its basis, and generates both denotational and operational representations from the algebraic refinement relation.

[*] Corresponding author.
E-mail address: qli@sei.ecnu.edu.cn (Q. Li).

https://doi.org/10.1016/j.scico.2017.10.009
0167-6423/© 2017 Elsevier B.V. All rights reserved.

This new strategy consists of the following steps:

Step 0. Construct a program algebra $(\mathcal{P}, \sqsubseteq_A)$ consisting of a set of laws (equations or inequations) to express the properties of program behaviours. The algebra does not depend on any interpretation model. The main criteria of the algebra is that whether the laws are sufficiently powerful to convert every program in the domain $\mathcal{P}$ to a normal form. Then define a refinement order $\sqsubseteq_A$ on normal forms so that we can compare the behaviours of two programs by comparing their normal forms.[1]

Step 1. Based on the program algebra, investigate the algebraic properties of the test operator $\mathcal{T}$ which has test case tc and testing program P as its arguments

$$\mathcal{T}(tc,\ P)$$

In case of GCL (Guarded Command Language) [3], tc is represented by a total constant assignment $x, y, .., z := a, b, .., c$ and the test operator $\mathcal{T}$ composes tc and P in sequence:

$$\mathcal{T}(tc,\ P) =_{df}\ (tc; P)$$

For CSP (Communicating Sequential Processes) [9,16], a test case has the same alphabet as the testing process, and takes the form of a generalised prefix process $s \rightarrow \Phi$ where s is a sequence of events in the alphabet of the process P, and Φ a choice construct $x : X \rightarrow Stop$ which is added to test the status of P after its engagement in the events of sequence s. The test $\mathcal{T}(tc,\ P)$ behaves like the system composed of processes tc and P interacting in lock-step synchronisation

$$\mathcal{T}(tc,\ P) =_{df}\ (tc \parallel P)$$

Step 2. Explore the dependency between the test outcome with the test case in the following form

$$\mathcal{T}(tc,\ P) =_A \sqcap Obs$$

where Obs denotes the set of visible observations one can record during the execution of the test and $\sqcap$ means the non-deterministic choices.

For GCL, an observation can be either a total constant assignment or the chaotic program $\bot$ which represents the worst outcome. In case of CSP an observation has a very similar form as test case.

Step 3. Based on the algebra of test, identify a program P as a binary relation $[P]$ which relates the test case with the final observation

$$[P] =_{df}\ \{(tc,\ obs) \mid \mathcal{T}(tc,\ P) \sqsubseteq_A obs\}$$

and selects the set inclusion as the refinement relation $\sqsubseteq_{rel}$

$$P \sqsubseteq_{rel} Q =_{df}\ ([P] \supseteq [Q])$$

Based on the algebra of programs, we can prove

$$\sqsubseteq_{rel} = \sqsubseteq_A$$

Step 4. Propose an algebraic definition of the *consistency* of step relation of the transition system of programs such that any consistent transition system $(O, \sqsubseteq_O)$ satisfies

$$\sqsubseteq_O = \sqsubseteq_A$$

Furthermore, our approach shows how to *generates* the transition rules for CSP combinators directly from the closure properties of the *canonical processes* presented in the consistent criterion of the step relation.

The paper is organised in the following way:

Section 2 briefly reviews the core results we got in chapter 5 of [11] and introduces the program algebra that will be used as the start base in sections 3 and 4.

Section 3 adopts this new roadmap to re-establish the semantical models of GCL, where

- Section 3.1 provides an algebraic representation of *machine state* and examines its properties.
- Section 3.2 introduces the notion of test cases.

[1] How to construct such program algebra is not the main concern of this paper. The main contribution of this paper which starts from Step 1 is based on the program algebras studied in chapter 5 of [11] which satisfies the criteria we mentioned in Step 0.