van Binsbergen, L. Thomas; Mosses, Peter D.; Sculthorpe, Neil


Summary: The potential benefits of formal semantics are well known. However, a substantial amount of work is required to produce a complete and accurate formal semantics for a major language; and when the language evolves, large-scale revision of the semantics may be needed to reflect the changes. The investment of effort needed to produce an initial definition, and subsequently to revise it, has discouraged language developers from using formal semantics. Consequently, many major programming languages (and most domain-specific languages) do not yet have formal semantic definitions.

To improve the practicality of formal semantic definitions, the PLANCOMPS project has developed a component-based approach. In this approach, the semantics of a language is defined by translating its constructs (compositionally) to combinations of so-called fundamental constructs, or ‘funcons’. Each funcon is defined using a modular variant of Structural Operational Semantics, and forms a language-independent component that can be reused in definitions of different languages. A substantial library of funcons has been developed and tested in several case studies. Crucially, the definition of each funcon is fixed, and does not need changing when new funcons are added to the library.

For specifying component-based semantics, we have designed and implemented a meta-language called CBS. It includes specification of abstract syntax, of its translation to funcons, and of the funcons themselves. Development of CBS specifications is supported by an integrated development environment. The accuracy of a language definition can be tested by executing the specified translation on programs written in the defined language, and then executing the resulting funcon terms using an interpreter generated from the CBS definitions of the funcons. This paper gives an introduction to CBS, illustrates its use, and presents the various tools involved in our implementation of CBS.

MSC:

68Q55 Semantics in the theory of computing
68N15 Theory of programming languages

Keywords:
programming languages; formal semantics; reuse; components; tool support

Software:
MMT; K Prover; Ott; Maude; PLT Redex; CoreASM; Hackage; Spoofax; Haskell

Full Text: DOI

References:
[7] PLanCompS project, CBS and funcons beta release, GitHub, (2018), Online
[38] Danvy, O.; Nielsen, L. R., Refocusing in Reduction Semantics, (2004), Department of Computer Science, Aarhus University, BRICS Research Series RS-04-26


[40] van Binsbergen, L. T.; Sculthorpe, N., Funcons values and value operations, Hackage, (2018), Online

[41] van Binsbergen, L. T.; Sculthorpe, N., Funcon interpreter for the SIMPLE language, Hackage, (2018), Online

[42] The $\mathbb{K}$ framework, GitHub, (2018), Online


This reference list is based on information provided by the publisher or from digital mathematics libraries. Its items are heuristically matched to zbMATH identifiers and may contain data conversion errors. It attempts to reflect the references listed in the original paper as accurately as possible without claiming the completeness or perfect precision of the matching.