Towards Semantics Online

Peter Mosses

Swansea University

(Emeritus, visiting Delft University of Technology)

`pdmosses.github.io`

BCTCS 2020, Swansea, April 2020

BCTCS 2006

Programming Language Description Languages

Semantics online

Proposal: establish an online repository

- individual construct descriptions
  - syntax and semantics of abstract constructs
- complete language descriptions
  - translations of concrete languages to (combinations of) abstract constructs

Conclusion

Constructive semantics supports a radical change of description method:

- independent description of individual abstract constructs
- translation from concrete languages to abstract constructs

and encourages the creation of a online repository of semantic descriptions

Component-based semantics (CBS)

Conjecture

Component-based semantics can greatly reduce the effort of language specification

Component-based semantics (CBS)

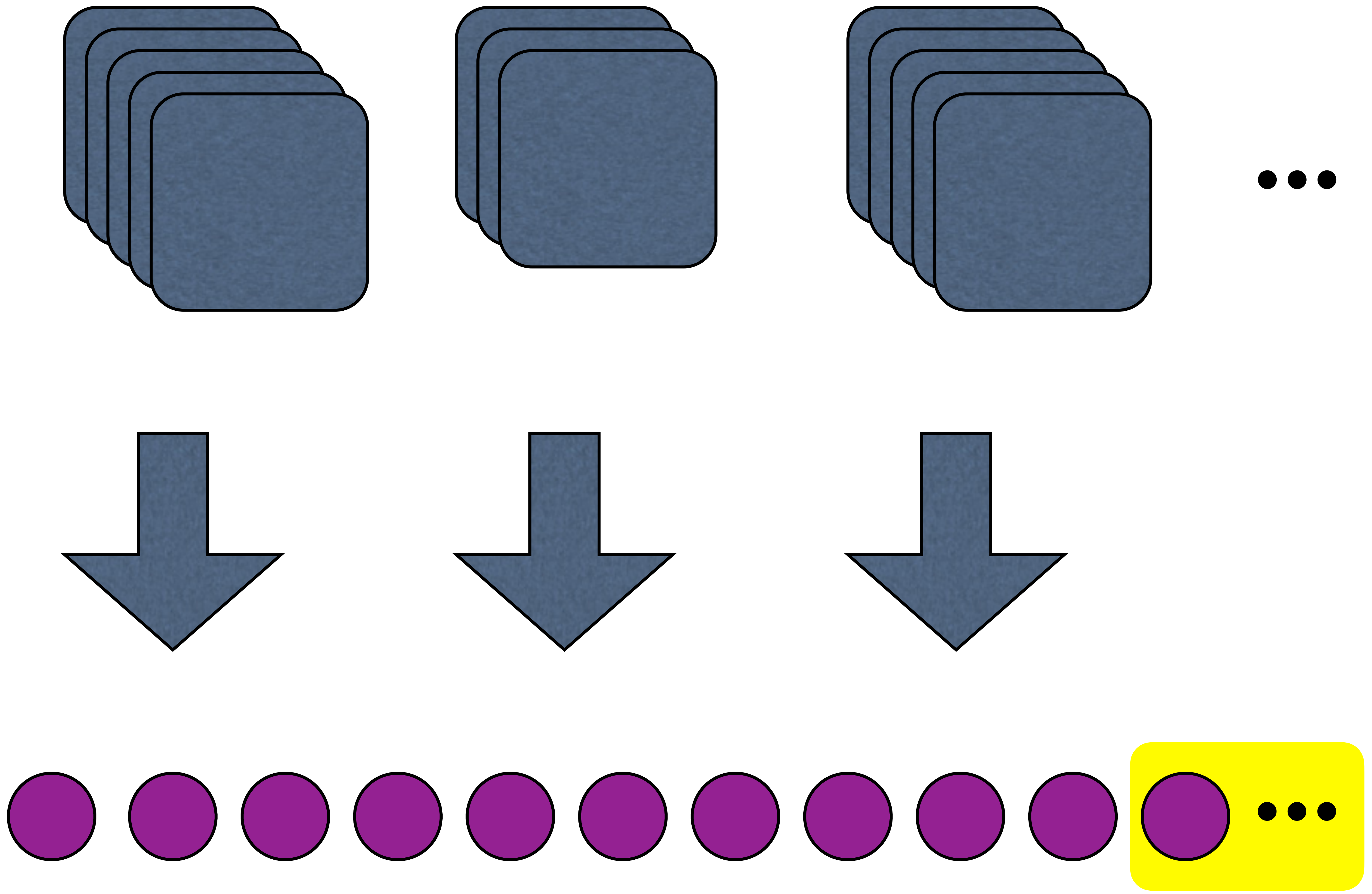
Programming languages

- specified by ***translation***

  to:

Components: ‘funcons’

- ***fundamental*** constructs
- ***open-ended*** library



Semantics Online requirements

Digital library

Curated repository

Language specifications

Reusable components (funcons)

Validation

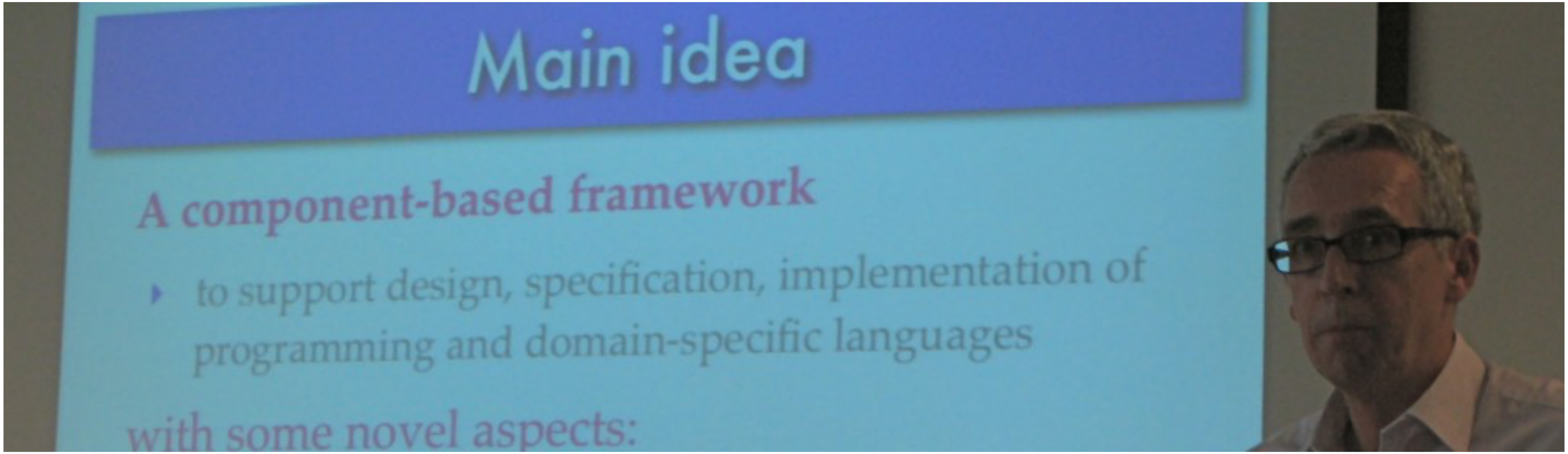
Tool support

Towards Semantics Online implementation

PLANCOMPS: Programming Language Components and Specifications

- 2011–2016: Swansea, RHUL, City, Newcastle

EPSRC
Engineering and Physical Sciences
Research Council



Towards Semantics Online implementation

PLANCOMPS: Programming Language Components and Specifications

- 2011–2016: Swansea, RHUL, City, Newcastle EPSRC Engineering and Physical Sciences Research Council
  - ***component-based framework*** (CBS meta-language, foundations)
  - ***specifications*** (example languages, reusable components)
  - ***tool support*** (IDE, parser generation, interpreter generation)
  - ***validation*** (test suites)
  - ***historical semantic descriptions library***

Component-Based Semantics (CBS)

Funcon definitions in CBS

Based on MSOS

a modular variant of Structural Operational Semantics (SOS)

- ***signatures***
  - distinguish between value and computation arguments
- ***inductive rules*** for small-step transitions
  - states: ***terms***, including computed ***values***
  - labels: collections of ***entities*** (environments, stores, signals, etc)
  - ***implicit propagation*** of unmentioned entities

Funcon definitions in CBS

Example

- ***signatures***

$$\mathit{Funcon}\ \textbf{if-true-else}(\_ : \textbf{booleans}, \_ : \Rightarrow T, \_ : \Rightarrow T) : \Rightarrow T$$

- ***inductive rules*** for small-step transitions and rewrites ($\longrightarrow$, $\leadsto$)

$$\mathit{Rule}\ \frac{B \longrightarrow B'}{\textbf{if-true-else}(B, X, Y) \longrightarrow \textbf{if-true-else}(B', X, Y)}$$

$$\mathit{Rule}\ \textbf{if-true-else}(\textbf{true}, X, \_) \leadsto X$$

$$\mathit{Rule}\ \textbf{if-true-else}(\textbf{false}, \_, Y) \leadsto Y$$

Language specifications in CBS

Languages are specified compositionally

- ***context-free syntax***
  - BNF, regular expressions, disambiguation (relative priorities, etc)
- ***translation functions*** : *syntax* → *funcons*
  - a ***semantic equation*** for each language construct
  - the semantics of funcons determines the language semantics

Language specifications in CBS

Example

- ***context-free syntax***

$$
\begin{aligned}
\mathit{Syntax}\ \mathit{Exp} : \mathsf{exp}\ ::=\ & \text{‘(’}\ \mathsf{exp}\ \text{‘)’} \mid \mathsf{value} \mid \mathsf{lexp} \mid \mathsf{lexp}\ \text{‘=’}\ \mathsf{exp} \mid \text{‘++’}\ \mathsf{lexp} \\
& \mid \text{‘−’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘(’}\ \mathsf{exps}^{?}\ \text{‘)’} \mid \text{‘}\textbf{sizeOf}\text{’}\ \text{‘(’}\ \mathsf{exp}\ \text{‘)’} \mid \text{‘}\textbf{read}\text{’}\ \text{‘(’}\ \text{‘)’} \\
& \mid \mathsf{exp}\ \text{‘+’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘−’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘}*\text{’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘/’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘\%’}\ \mathsf{exp} \\
& \mid \mathsf{exp}\ \text{‘<’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘<=’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘>’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘>=’}\ \mathsf{exp} \\
& \mid \mathsf{exp}\ \text{‘==’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘!=’}\ \mathsf{exp} \mid \text{‘!’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘\&\&’}\ \mathsf{exp} \mid \mathsf{exp}\ \text{‘||’}\ \mathsf{exp}
\end{aligned}
$$

Language specifications in CBS

Example

- ***translation functions*** : *syntax* → *funcons*

$$\mathit{Semantics}\ \mathit{rval}[\![\_ : \mathsf{exp}]\!] : \Rightarrow\mathbf{values}$$

- ***semantic equations***

$$\mathit{Rule}\ \mathit{rval}[\![\mathit{Exp}_1\ \text{‘\&\&’}\ \mathit{Exp}_2]\!] = \textbf{if-true-else}(\mathit{rval}[\![\mathit{Exp}_1]\!], \mathit{rval}[\![\mathit{Exp}_2]\!], \textbf{false})$$

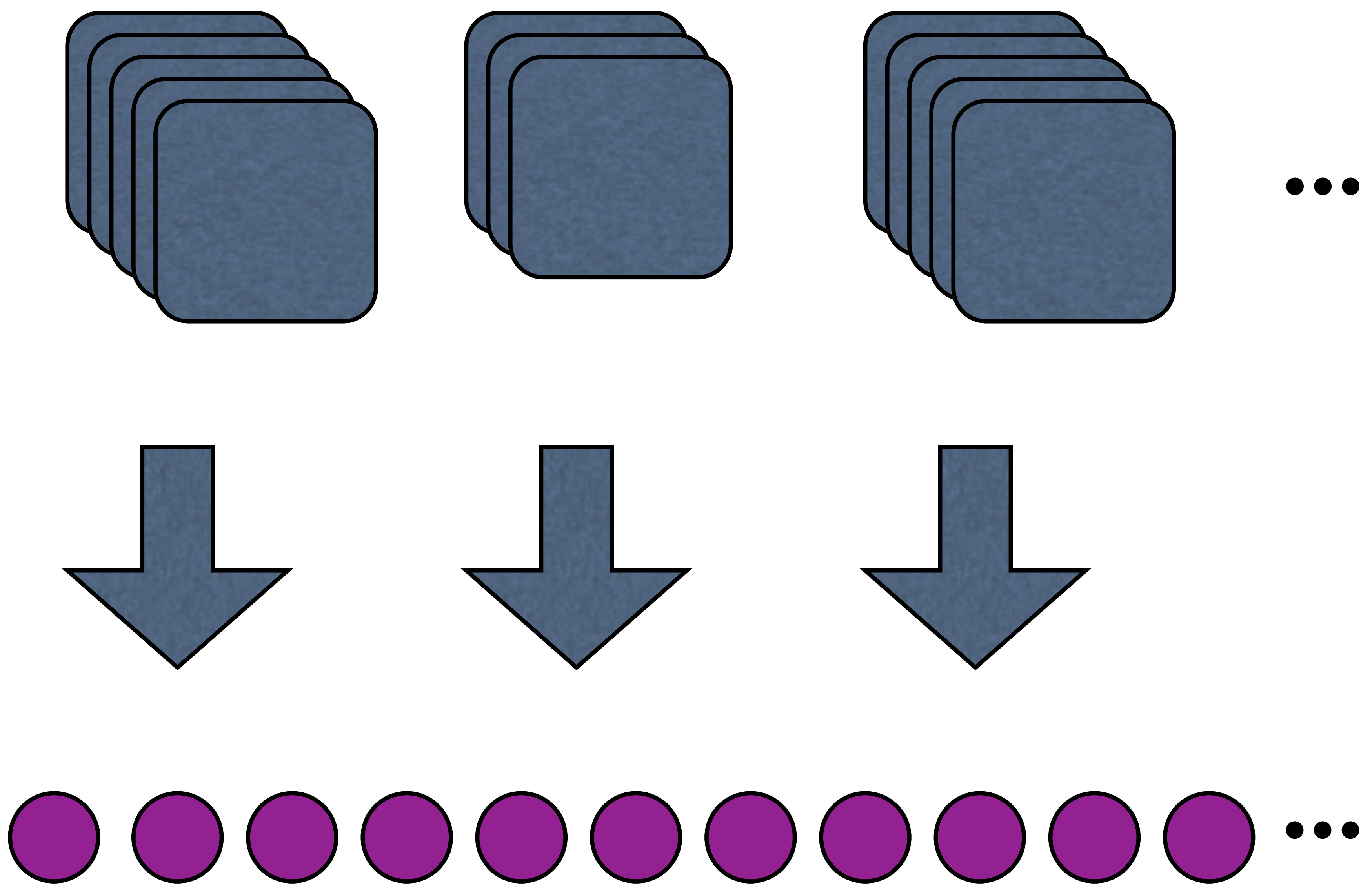
Modularity in CBS

Language specifications

- ***independent modules***

Funcons library

- ***imported***



Support for evolution in CBS

Funcon definitions

- ***funcon definitions never change or disappear!***
- ***new funcons can always be added***

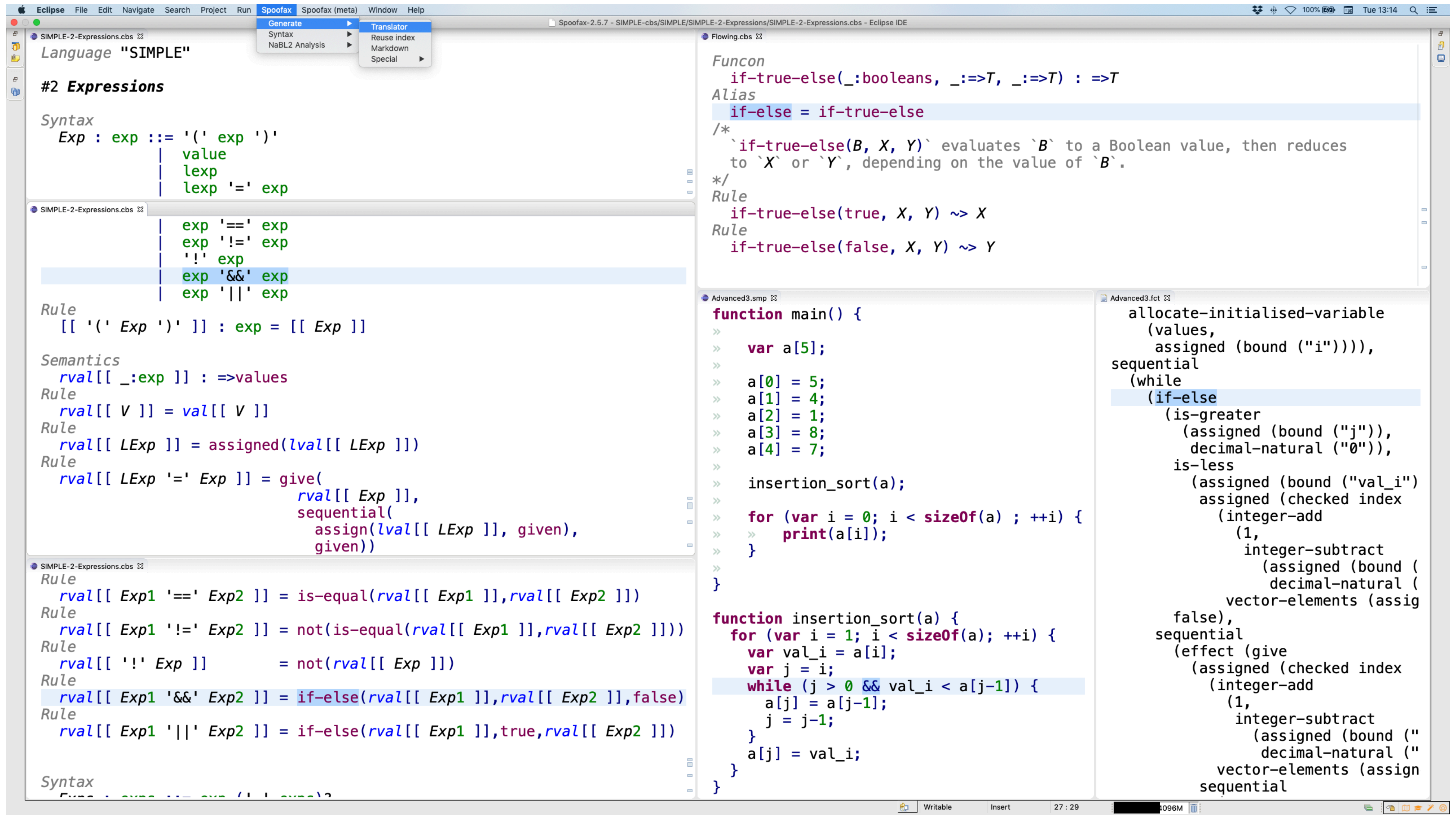
Language specifications

- ***co-evolve*** with language design
- ***not*** reusable components

Tool support for CBS specifications

IDE for creating, editing, browsing

- *grammars, translations, funcons*

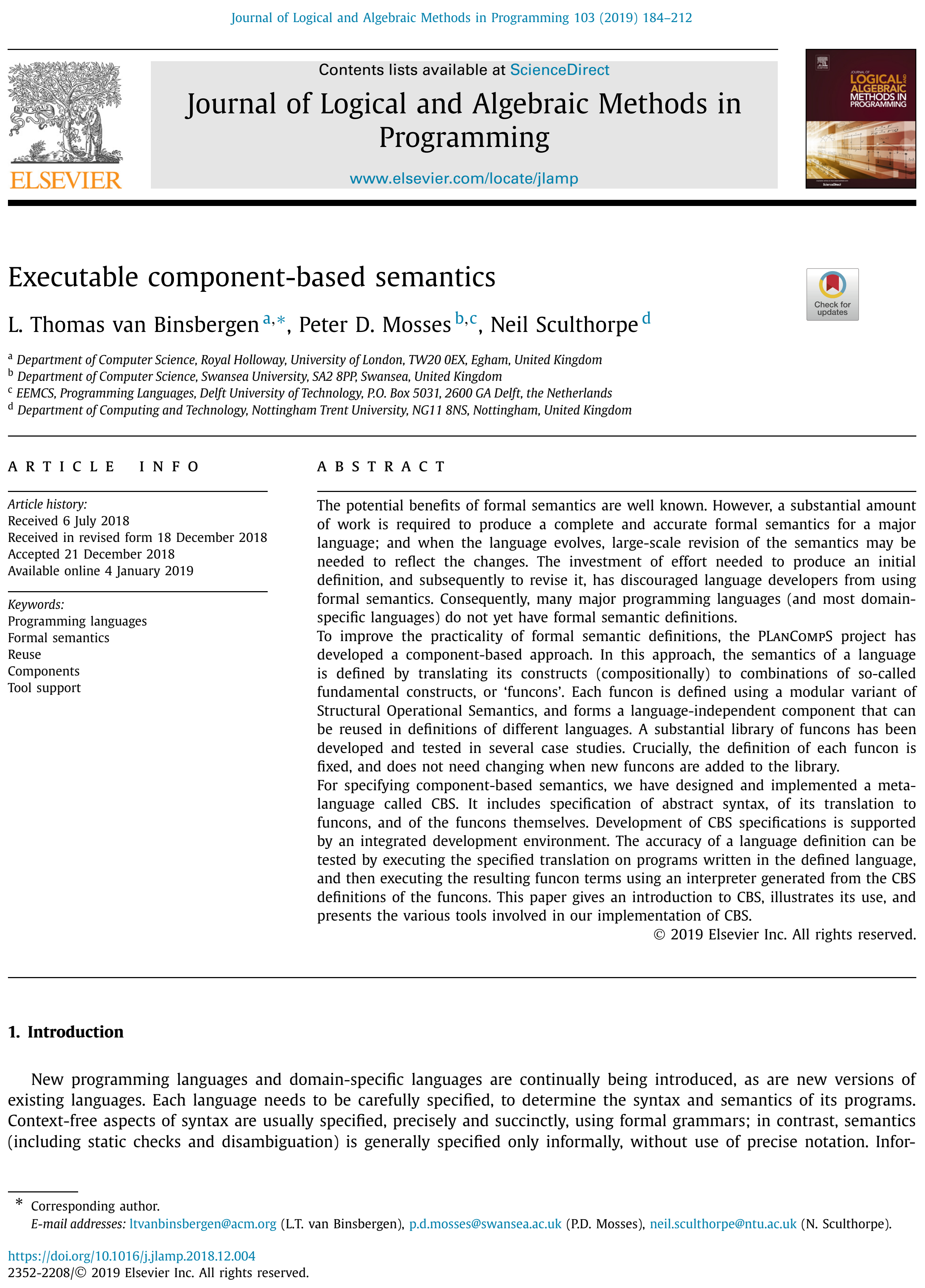


Generating prototypes

- language ***parser***
- funcon ***interpreter***
- ***translator*** : language → funcons
  - *hence program execution*

Recent references for CBS

- Executable component-based semantics



Journal of Logical and Algebraic Methods in Programming 103 (2019) 184–212

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Executable component-based semantics

L. Thomas van Binsbergen [a,*], Peter D. Mosses [b,c], Neil Sculthorpe [d]

[a] Department of Computer Science, Royal Holloway, University of London, TW20 0EX, Egham, United Kingdom
[b] Department of Computer Science, Swansea University, SA2 8PP, Swansea, United Kingdom
[c] EEMCS, Programming Languages, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, the Netherlands
[d] Department of Computing and Technology, Nottingham Trent University, NG11 8NS, Nottingham, United Kingdom

ARTICLE INFO

Article history:
Received 6 July 2018
Received in revised form 18 December 2018
Accepted 21 December 2018
Available online 4 January 2019

Keywords:
Programming languages
Formal semantics
Reuse
Components
Tool support

ABSTRACT

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.

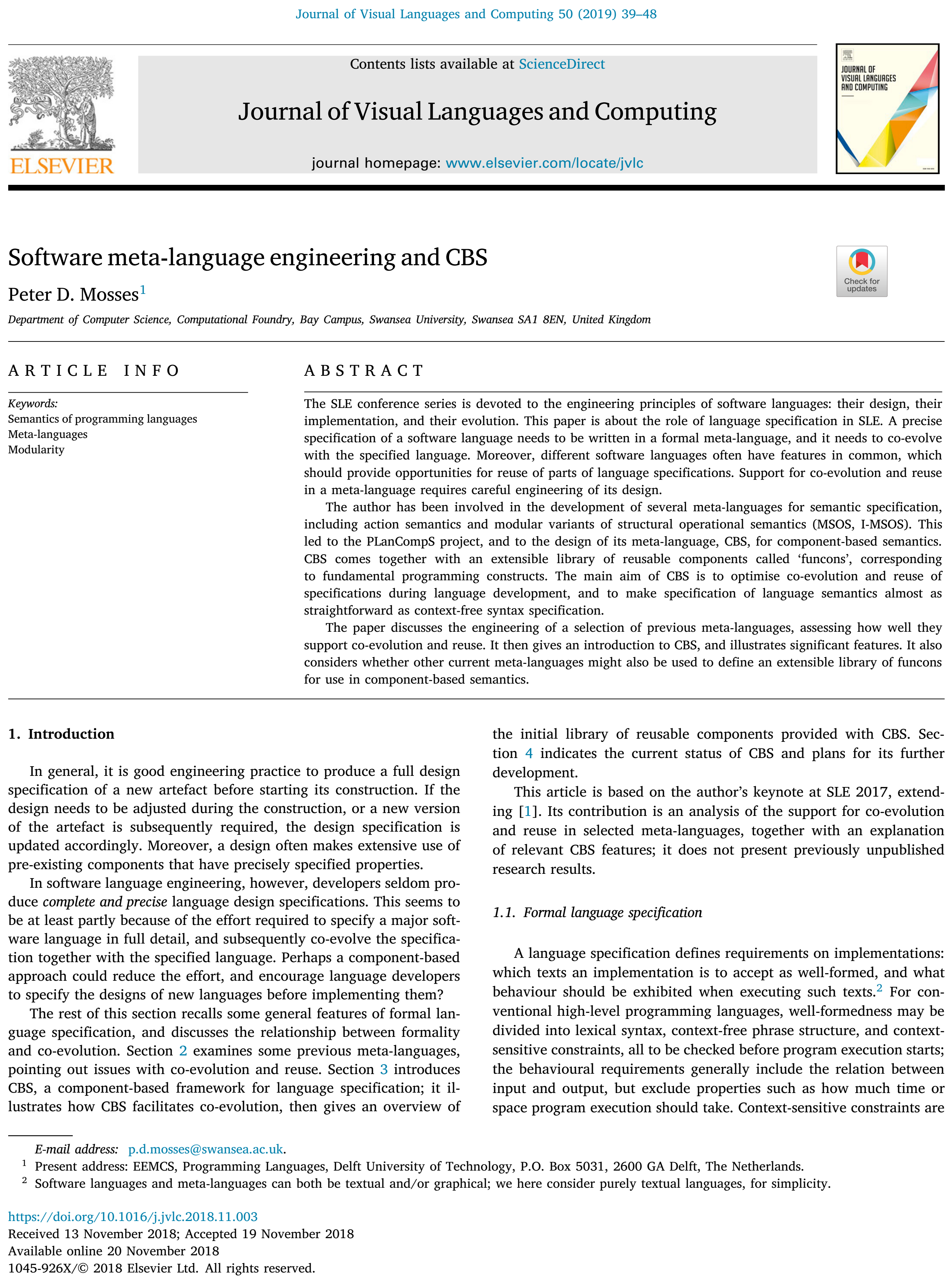
1. Introduction

New programming languages and domain-specific languages are continually being introduced, as are new versions of existing languages. Each language needs to be carefully specified, to determine the syntax and semantics of its programs. Context-free aspects of syntax are usually specified, precisely and succinctly, using formal grammars; in contrast, semantics (including static checks and disambiguation) is generally specified only informally, without use of precise notation. Infor-

* Corresponding author.
E-mail addresses: ltvanbinsbergen@acm.org (L.T. van Binsbergen), p.d.mosses@swansea.ac.uk (P.D. Mosses), neil.sculthorpe@ntu.ac.uk (N. Sculthorpe).

https://doi.org/10.1016/j.jlamp.2018.12.004

- Software meta-languages and CBS



Journal of Visual Languages and Computing 50 (2019) 39–48

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Software meta-language engineering and CBS

Peter D. Mosses[1]

Department of Computer Science, Computational Foundry, Bay Campus, Swansea University, Swansea SA1 8EN, United Kingdom

ARTICLE INFO

Keywords:
Semantics of programming languages
Meta-languages
Modularity

ABSTRACT

The SLE conference series is devoted to the engineering principles of software languages: their design, their implementation, and their evolution. This paper is about the role of language specification in SLE. A precise specification of a software language needs to be written in a formal meta-language, and it needs to co-evolve with the specified language. Moreover, different software languages often have features in common, which should provide opportunities for reuse of parts of language specifications. Support for co-evolution and reuse in a meta-language requires careful engineering of its design.

The author has been involved in the development of several meta-languages for semantic specification, including action semantics and modular variants of structural operational semantics (MSOS, I-MSOS). This led to the PLanCompS project, and to the design of its meta-language, CBS, for component-based semantics. CBS comes together with an extensible library of reusable components called 'funcons', corresponding to fundamental programming constructs. The main aim of CBS is to optimise co-evolution and reuse of specifications during language development, and to make specification of language semantics almost as straightforward as context-free syntax specification.

The paper discusses the engineering of a selection of previous meta-languages, assessing how well they support co-evolution and reuse. It then gives an introduction to CBS, and illustrates significant features. It also considers whether other current meta-languages might also be used to define an extensible library of funcons for use in component-based semantics.

1. Introduction

In general, it is good engineering practice to produce a full design specification of a new artefact before starting its construction. If the design needs to be adjusted during the construction, or a new version of the artefact is subsequently required, the design specification is updated accordingly. Moreover, a design often makes extensive use of pre-existing components that have precisely specified properties.

In software language engineering, however, developers seldom produce *complete and precise* language design specifications. This seems to be at least partly because of the effort required to specify a major software language in full detail, and subsequently co-evolve the specification together with the specified language. Perhaps a component-based approach could reduce the effort, and encourage language developers to specify the designs of new languages before implementing them?

The rest of this section recalls some general features of formal language specification, and discusses the relationship between formality and co-evolution. Section 2 examines some previous meta-languages, pointing out issues with co-evolution and reuse. Section 3 introduces CBS, a component-based framework for language specification; it illustrates how CBS facilitates co-evolution, then gives an overview of the initial library of reusable components provided with CBS. Section 4 indicates the current status of CBS and plans for its further development.

This article is based on the author's keynote at SLE 2017, extending [1]. Its contribution is an analysis of the support for co-evolution and reuse in selected meta-languages, together with an explanation of relevant CBS features; it does not present previously unpublished research results.

1.1. Formal language specification

A language specification defines requirements on implementations: which texts an implementation is to accept as well-formed, and what behaviour should be exhibited when executing such texts.[2] For conventional high-level programming languages, well-formedness may be divided into lexical syntax, context-free phrase structure, and context-sensitive constraints, all to be checked before program execution starts; the behavioural requirements generally include the relation between input and output, but exclude properties such as how much time or space program execution should take. Context-sensitive constraints are

E-mail address: p.d.mosses@swansea.ac.uk.
[1] Present address: EEMCS, Programming Languages, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands.
[2] Software languages and meta-languages can both be textual and/or graphical; we here consider purely textual languages, for simplicity.

https://doi.org/10.1016/j.jvlc.2018.11.003
Received 13 November 2018; Accepted 19 November 2018
Available online 20 November 2018

Towards Semantics *Online*

Historical semantic descriptions

http://plancomps.org/semantic-descriptions-library/

Not Secure — homepages.cs.ncl.ac.uk

CBJ's contribution to "Semantics Library" · Cliff B Jones

Cliff B Jones

Semantic descriptions library

These are my (current, evolving) contributions to a "library of semantics". This material is being extended when time and resources allow. (The work was initiated during the PLanCompS project.)

Formal descriptions of ALGOL-60

Thanks to painstaking work by Roberta Velykiene, the following scanned PDFs have an overlay which makes searching possible (even for Greek letters!)

- *Peter Lauer's VDL description of ALGOL 60* (TR 25.088)
- *A `functional' semantics of ALGOL 60* (Notice that this scanned version deliberately omits the pages that contained the ALGOL report that were lined-up with the corresponding formulae)
- *Peter Mosses' (Oxford) Denotational descriptionof ALGOL 60*
- *A (actually, the second) VDM description of ALGOL 60*
- *A re-LaTeXed version of the ALGOL 60 report*

An organisation for Semantics Online
`plancomps.github.io`

PLANCOMPS: Programming Language Components and Specifications

- 2011–2016: Swansea, RHUL, City, Newcastle EPSRC Engineering and Physical Sciences Research Council

Since 2016:

- more specifications (e.g., threads)
- more tool support (e.g., Markdown generation)
- ***a website*** for browsing languages and funcons

Towards a website for Semantics Online

`plancomps.github.io/CBS-beta/`

plancomps.github.io

Languages-beta - CBS-beta

CBS-beta

Search CBS-beta

CBS-beta on GitHub

- Funcons-beta
- **Languages-beta**
  - IMP
  - SIMPLE
  - MiniJava
  - SL
  - OC-L
  - Funcon reuse
- Unstable-Funcons-beta
- Unstable-Languages-beta
- Browsing guide
- Publications
- Changes
- Downloads

This site uses Just the Docs, a documentation theme for Jekyll.

Languages-beta

This page provides access to the beta-release of five examples of language specifications in CBS, based on an initial collection of so-called ‘funcons’ (fundamental programming constructs). Funcons are reusable components of programming language specifications: in the CBS framework, a language is specified by defining a translation from its constructs to funcon terms, reusing (by reference) the definitions of the required funcons.

Before proceeding, take a look at the navigation and language definition sections of the Browsing guide.

See Funcons-beta for the beta-release of an initial collection of funcons. Those funcons used in the CBS of a particular language are listed in an index, accessed from the dropdown menu shown when browsing its specification.

The specification of each funcon has been validated by basic unit tests that exercise all its rules. Moreover, funcons that are reused in two or more of the illustrative language specifications have been exercised in different contexts.

The current level of validation of the illustrative language specifications varies considerably, and the coverage of the test suites has not yet been analysed. The colophon on the start page of each language specification includes the size of the current test suite and an indication of its expected coverage. Further tests are to be made during the beta-release review period; the test suites will subsequently be added to this website.

Conclusion

Towards Semantics Online – the story so far:

- 2006: Semantics Online proposed in BCTCS talk
- 2011: PLANCOMPS project started
- 2016: CBS framework established
- 2018: CBS-beta funcons and languages available for review on GitHub
- 2020: PLANCOMPS organisation on GitHub

To be continued – new participants are welcome! Email `plancomps@gmail.com`