WG 2.2 and semantics of programming languages

Peter Mosses

Swansea University, UK

IFIP WG 2.2 Meeting
15–18 September 2014, Munich

IFIP WG 2.2 pre-history

1964: *IFIP Working Conference,* Baden-bei-Wien

FORMAL LANGUAGE
DESCRIPTION LANGUAGES
FOR COMPUTER PROGRAMMING

N·H
P C

Proceedings of the
IFIP Working Conference on
Formal Language Description Languages

1966

IFIP WG 2.2 history

1962

- TC 2 – *Software: Theory and Practice*

1965(!)

- WG 2.2 – *Formal Description of Programming Concepts*
  - chair: T. B. Steel, Jr.

1967

- first WG 2.2 meeting (Alghero, Italy)

1984

- (member)

WG 2.2

AIMS

- The [primary] aim of the Working Group is to ***explicate programming concepts*** through the development, examination and comparison of various ***formal models of these concepts.***

SCOPE

- The Working Group will investigate ***formalisms and models*** which represent different approaches to ***formal specification of programming concepts.*** […]

wg22.labri.fr

Own contributions

early 1980s

- **abstract semantic algebras**

late 80s – 90s

- **action semantics**

late 90s – 2000s

- **modular SOS**

2010s

- **component-based semantics**

Further IFIP WGs

WG 1.3 (member since 1994, chair 1998–2003)

- ***Foundations of system specification***
- **CoFI**: Common Framework Initiative for algebraic specification and development of software
  - **CASL**

Further IFIP WGs

WG 2.11 (member since 2013)

- ***Program generation***

SCOPE [https://wiki.hh.se/wg211/]

- The working group covers the following research areas (and maybe others):
  - ***programming language*** design, ***semantics*** and implementation
  - ***program synthesis***
  - ***type systems*** and type theory
  - […]

Recent WG 2.2 talks

- 2010 (Warsaw):
  - ***On bisimulation and modularity***
- 2011 (Paris):
  - **PLANCOMPS** – ***Programming Language Components and Specifications***
- 2012 (Amsterdam):
  - ***Component-based semantics***
- 2013 (Lisbon):
  - ***Editor support for formal specifications***

This talk

Component-based semantics

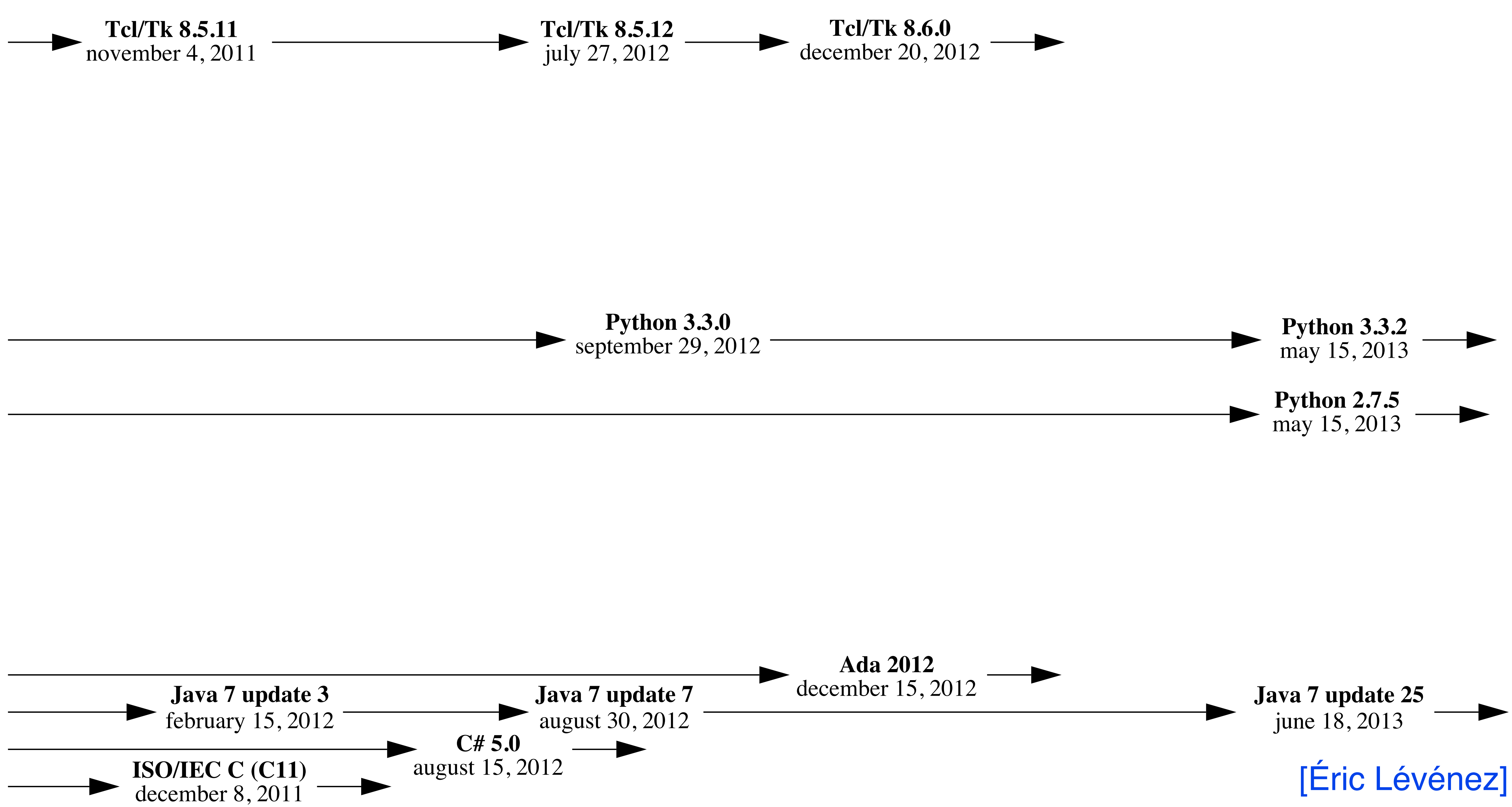
- brief recap/overview
  - motivation
  - main ideas
- PLANCOMPS project
  - progress report
  - current and planned work

Component-based semantics

Programming languages

2012 2013

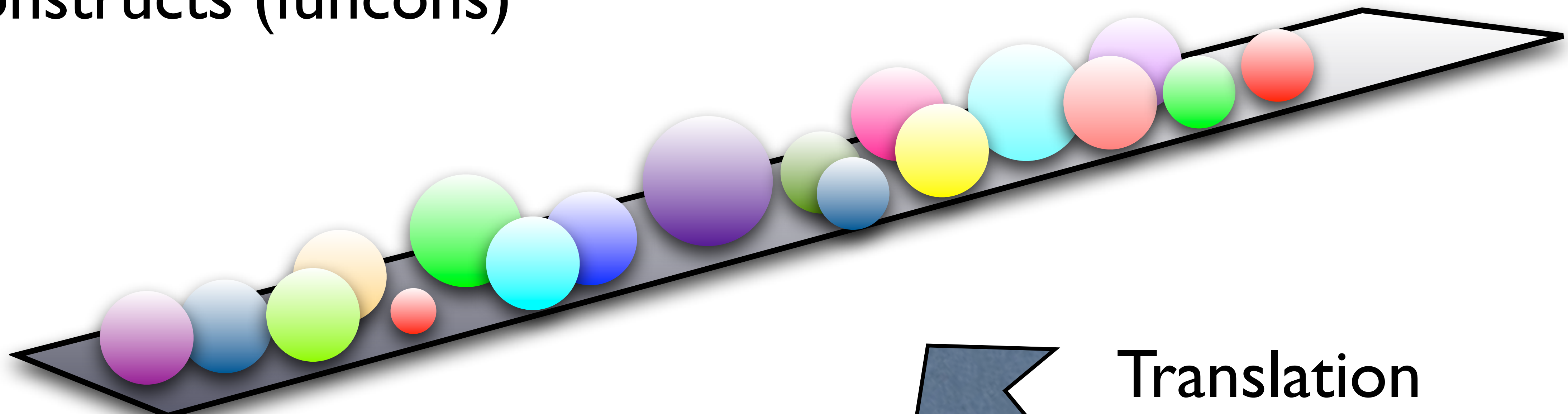
Evolution!



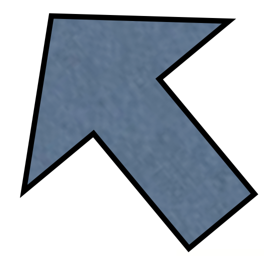
[Éric Lévénez]

Component-based semantics

Fundamental programming constructs (funcons)



Components-off-the-shelf (digital library)



Translation (reduction)

Evolving languages

Reusable components

Fundamental constructs (funcons)

- correspond to programming constructs
  - ***directly*** (**if-true**), or
  - ***special case*** (**apply**), or
  - ***implicit*** (**bound-value**)
- and have *(when validated and released)*
  - ***fixed*** notation, and
  - ***fixed*** behaviour, and
  - ***fixed*** algebraic properties

specified/proved once and for all!

Component reuse

Language construct:

- $exp ::= exp\ \mathbf{?}\ exp\ \mathbf{:}\ exp$

Translation to funcons:

- $expr[\![\ E_1\ \mathbf{?}\ E_2\ \mathbf{:}\ E_3\ ]\!] =$
  $\mathbf{if\text{-}true}(expr[\![\ E_1\ ]\!], expr[\![\ E_2\ ]\!],\quad expr[\![\ E_3\ ]\!])$

For languages with non-Boolean tests:

- $expr[\![\ E_1\ \mathbf{?}\ E_2\ \mathbf{:}\ E_3\ ]\!] =$
  $\mathbf{if\text{-}true}(\mathbf{not}(\mathbf{equal}(expr[\![\ E_1\ ]\!], 0)),$
  $expr[\![\ E_2\ ]\!],\quad expr[\![\ E_3\ ]\!])$

Component reuse

Language construct:

- stm ::= `if(`exp`)` stm `else` stm

Translation to funcons:

- $comm[\![$ `if(`E_1`)` S_2 `else` S_3 $]\!]$ =
  if-true($expr[\![\ E_1\ ]\!]$, $comm[\![\ S_2\ ]\!]$, $comm[\![\ S_3\ ]\!]$)

For languages with non-Boolean tests:

destructive change

- $comm[\![$ `if(`E_1`)` S_2 `else` S_3 $]\!]$ =
  if-true(**not**(**equal**($expr[\![\ E_1\ ]\!]$, 0)),
  $comm[\![\ S_2\ ]\!]$, $comm[\![\ S_3\ ]\!]$)

Component specification

modular extension

Notation

$$\textbf{if-true}(\text{boolean}, \text{comp}(T), \text{comp}(T)) : \text{comp}(T)$$

Static semantics

$$\frac{E : \textbf{boolean}, \quad X_1 : T, \quad X_2 : T}{\textbf{if-true}(E, X_1, X_2) : T}$$

specified once and for all!

Dynamic semantics

$$\textbf{if-true}(\textbf{true}, X_1, X_2) \rightarrow X_1$$

$$\textbf{if-true}(\textbf{false}, X_1, X_2) \rightarrow X_2$$

PLANCOMPS project

PLANCOMPS – foundations

FOSSACS'13:

- bisimilarity congruence format
- preservation by disjoint extension

Modular Bisimulation Theory for Computations and Values

Martin Churchill and Peter D. Mosses
{m.d.churchill,p.d.mosses}@swansea.ac.uk

Department of Computer Science, Swansea University, Swansea, UK

Abstract. For structural operational semantics (SOS) of process algebras, various notions of bisimulation have been studied, together with rule formats ensuring that bisimilarity is a congruence. For programming languages, however, SOS generally involves auxiliary entities (e.g. stores) and computed values, and the standard bisimulation and rule formats are not directly applicable.
Here, we first introduce a notion of bisimulation based on the distinction between computations and values, with a corresponding liberal congruence format. We then provide metatheory for a modular variant of SOS (MSOS) which provides a systematic treatment of auxiliary entities. This is based on a higher order form of bisimulation, and we formulate an appropriate congruence format. Finally, we show how algebraic laws can be proved sound for bisimulation with reference only to the (M)SOS rules defining the programming constructs involved in them. Such laws remain sound for languages that involve further constructs.

PLANCOMPS – foundations

SLE'13:

- addressing deficiency of disambiguation annotations in SDF, Rascal, Spoofax

Safe Specification of Operator Precedence Rules

Ali Afroozeh[1], Mark van den Brand[3], Adrian Johnstone[4], Elizabeth Scott[4], and Jurgen Vinju[1,2]

[1] Centrum Wiskunde & Informatica, 1098 XG Amsterdam, The Netherlands
[2] INRIA Lille Nord Europe, France
{ali.afroozeh,jurgen.vinju}@cwi.nl
[3] Eindhoven University of Technology, NL-5612 AZ Eindhoven, The Netherlands
m.g.j.v.d.brand@tue.nl
[4] Royal Holloway, University of London, Egham, Surrey, TW20 0EX, UK
{a.johnstone,e.scott}@rhul.ac.uk

Abstract. In this paper we present an approach to specifying operator precedence based on declarative disambiguation constructs and an implementation mechanism based on grammar rewriting. We identify a problem with existing generalized context-free parsing and disambiguation technology: generating a correct parser for a language such as OCaml using declarative precedence specification is not possible without resorting to some manual grammar transformation. Our approach provides a fully declarative solution to operator precedence specification for context-free grammars, is independent of any parsing technology, and is safe in that it guarantees that the language of the resulting grammar will be the same as the language of the specification grammar. We evaluate our new approach by specifying the precedence rules from the OCaml reference manual against the highly ambiguous reference grammar and validate the output of our generated parser.

PLANCOMPS – foundations

ESOP'14:

- refocusing small-step (M)SOS rules

Deriving Pretty-Big-Step Semantics from Small-Step Semantics

Casper Bach Poulsen and Peter D. Mosses

Department of Computer Science, Swansea University, Swansea, UK,
cscbp@swansea.ac.uk, p.d.mosses@swansea.ac.uk

Abstract. Big-step semantics for languages with abrupt termination and/or divergence suffer from a serious duplication problem, addressed by the novel 'pretty-big-step' style presented by Charguéraud at ESOP'13. Such rules are less concise than corresponding small-step rules, but they have the same advantages as big-step rules for program correctness proofs. Here, we show how to automatically derive pretty-big-step rules directly from small-step rules by 'refocusing'. This gives the best of both worlds: we only need to write the relatively concise small-step specifications, but our reasoning can be big-step as well as small-step. The use of strictness annotations to derive small-step congruence rules gives further conciseness.

PLANCOMPS – mini case study

MODULARITY'14:

- component-based semantics of Caml Light
- partially validated (by empirical testing)
- detailed introduction to the approach
- preliminary tool support

Reusable Components of Semantic Specifications

Martin Churchill Peter D. Mosses Paolo Torrini

PLANCOMPS – tool support

Preliminary tool chain:

- **SPOOFAX**
  - parsing programs (SDF3, disambiguation, AST creation)
  - translating ASTs to funcon terms (SDF3, Stratego-1.2)
  - browsing and editing component-based specifications (SDF3, NaBL, Stratego-1.2)
- **PROLOG**
  - translating MSOS rules for funcons to PROLOG
  - running funcon terms

PLANCOMPS – current, planned

Case studies:

- C# *(started)*, Java

Improved tool support:

- generating SDF3 and Prolog from CBS
- refocusing MSOS rules

Digital library:

- historic semantics documents *(Cliff Jones, Newcastle)*
  - *ALGOL 60 in various frameworks (scan, OCR to PDF)*
- browsing and searching an open-access repository

Conclusion

Component-based semantics

- translating programs to funcons
- specifying funcons in (I-)MSOS

PLANCOMPS project (2011-2015)

- foundations
- case studies
- tool support
- digital library