Executable Component-Based Semantics

L. Thomas van Binsbergen[1], Neil Sculthorpe[2], Peter D. Mosses[3]

ltvanbinsbergen@acm.org

[1]Royal Holloway, University of London ⟶ Centrum Wiskunde & Informatica

[2]Swansea University ⟶ Nottingham Trent University

[3]Swansea University ⟶ TU Delft (Visiting)

17 March, 2020

Executable **component-based semantics**

Introduce the concept of 'code reuse' to software language semantics, in order to:
simplify *development*, *maintenance* and to formalise proven language design *principles*

Executable component-based semantics

Generating prototype language implementations enables *rapid prototyping*;
relies heavily on compositionality and *modularity*

- Component-based approach towards formal (dynamic) semantics

Main contributions:

- A library of highly reusable, *fun*damental *con*structs (*funcons*)[1]
- The meta-language CBS for defining component-based semantics[2]

[1] `https://plancomps.github.io/`

[2] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

What is the state of the funcon library?

Verified and available

`https://plancomps.github.io/`

- Procedural: procedures, references, scoping, iteration
- Functional: functions, bindings, datatypes, pattern matching
- Object-oriented: objects, classes, inheritance
- Abnormal control: exceptions, break/continue, delimited continuations[3]

Unverified as of yet (prototype phase)

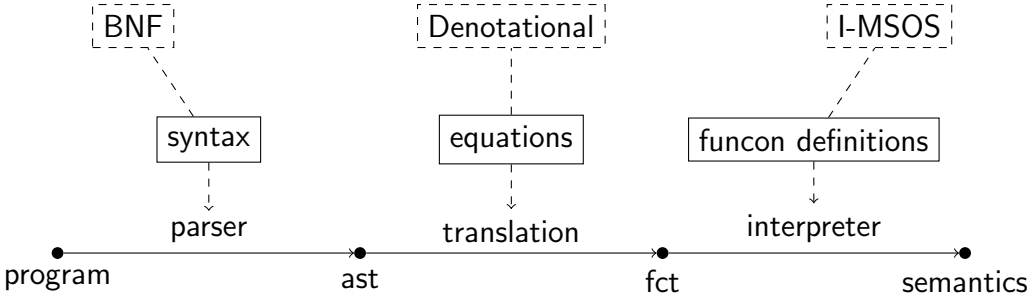
- Concurrency: multi-threading
- Logical programming: backtracking, unification
- Meta-programming: AST conversions, staged evaluation[4]

---

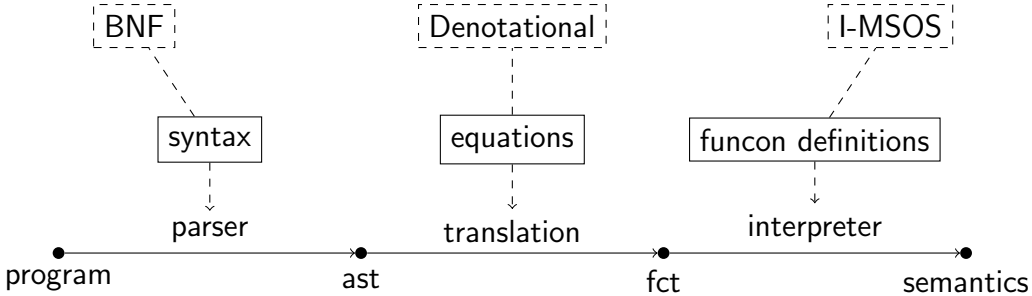
[3] *A Modular Structural Operational Semantics for Delimited Continuations*. Sculthorpe, Torrini, and Mosses. WoC 2016

[4] *Funcons for Homogeneous Generative Meta-Programming*. Van Binsbergen. GPCE 2018

Executable language specification with CBS and Funcons

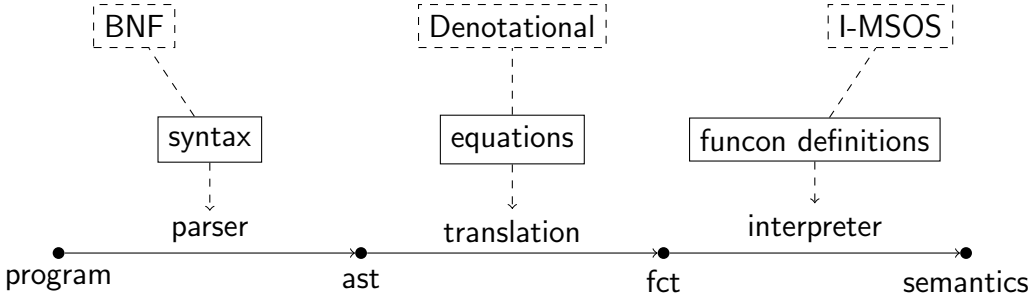


Executable language specification with CBS and Funcons



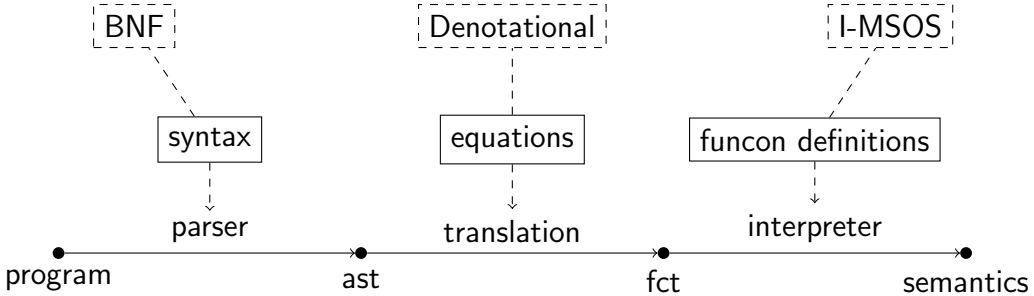
- CBS language definitions: syntax, translation equations and funcon definitions

Executable language specification with CBS and Funcons



- CBS language definitions: syntax, translation equations and funcon definitions
- CBS syntax and CBS translation equations implemented as Spoofax editor-project

Executable language specification with CBS and Funcons



- CBS language definitions: syntax, translation equations and funcon definitions
- CBS syntax and CBS translation equations implemented as Spoofax editor-project
- CBS funcon definitions implemented by a separate CBS compiler (generates Haskell)

SIMPLE – language definition fragments

Syntax

```
Exp : exp ::= '(' exp ')'
            | value
            | lexp
            | lexp '=' exp
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).

[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

```
Semantics
  rval[[ _:exp ]] : =>values
Rule
  rval[[ V ]] = val[[ V ]]
Rule
  rval[[ LExp ]] = assigned(lval[[ LExp ]])
Rule
  rval[[ LExp '=' Exp ]] = give(
                             rval[[ Exp ]],
                             sequential(
                               assign(lval[[ LExp ]], given),
                               given))
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).

[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

Syntax

```
LExp : lexp ::= id | lexp '[' exps ']'
```

Rule

```
[[ LExp '[' Exp ',' Exps ']' ]] : lexp =
[[ LExp '[' Exp ']' '[' Exps ']' ]]
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).

[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

```
Semantics
  lval[[ _:lexp ]] : =>variables
Rule
  lval[[ Id ]] = bound(id[[ Id ]])
Rule
   lval[[ LExp '[' Exp ']' ]] =
     checked index(integer-add(1,rval[[ Exp ]]),vector-elements(rval[[ LExp ]]))
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).

[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

Rapid **prototyping**

1. Define syntax of (additional) language construct
2. Define translation of new syntax based on high-level understanding of funcons
3. Test extended language by running example programs
4. Optionally reconsider specifics (based on detailed understanding of funcons)
5. Repeat with additional or alternative constructs

SIMPLE – language definition fragments

```
Funcon
  bound-value(I:identifiers) : =>values
   ~> follow-if-link(bound-directly(I))
Alias
  bound = bound-value
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).
[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

```
Funcon
  bound-directly(_:identifiers) : =>values
```

```
Rule
  lookup(Rho, I) ~> (V:values)
  ------------------------------------------------------------
  environment(Rho) |- bound-directly(I:identifiers) ---> V
Rule
  lookup(Rho, I) ~> ( )
  ------------------------------------------------------------
  environment(Rho) |- bound-directly(I:identifiers) ---> fail
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).
[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

```
Entity
  environment(_:environments) |- _ ---> _
```

```
Alias
  env = environment
```

```
Type
  environments ~> maps(identifiers, values?)
Alias
  envs = environments
```

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).

[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

SIMPLE – language definition fragments

```
Entity
  environment(_:environments) |- _ ---> _
```

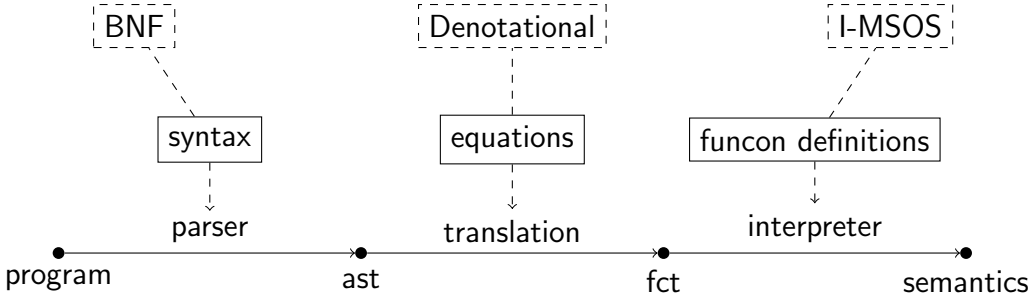
```
Alias
  env = environment
```

```
Type
  environments ~> maps(identifiers, values?)
Alias
  envs = environments
```

- Propagation of *auxiliary entities* such as environments is the key modularity challenge

[5] $\mathbb{K}$ *Overview and SIMPLE case study*. Roșu and Șerbănuță. ENTCS 304 (2014).
[6] *Executable Component-Based Semantics*. Van Binsbergen, Sculthorpe, Mosses. JLAMP 2019

Executable language specification with Funcons



- CBS implementation partially in Spoofax and Haskell
- PhD thesis implementation: Haskell libraries (EDSLs) for parsing and funcons[7]
- The Rascal meta-programming can also be used to define syntax and translations[8]

[7] http://ltvanbinsbergen.nl/thesis
[8] https://github.com/cwi-swat/rascal-funcons-beta

Evaluating the Haskell Funcon Framework 1/2

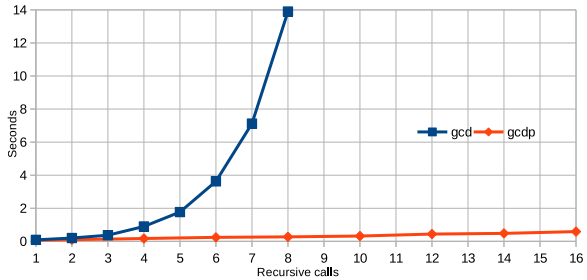
Test Program	Generated Interpreter		$\mathbb{K}$ Tool
	Unoptimised	Refocussing Enabled	
`exception tests 1 to 15`	11.4	1.67	30.3
`div-nondet`	0.2	0.06	1.9
`factorial`	2.8	0.18	1.9
`collatz`	11.3	0.53	1.9
running total	25.6	2.43	35.9
`higher-order`	-	3.02	10.1
`matrix`	-	5.84	2.2
`sortings`	-	6.11	2.9
running total	-	17.4	51.1

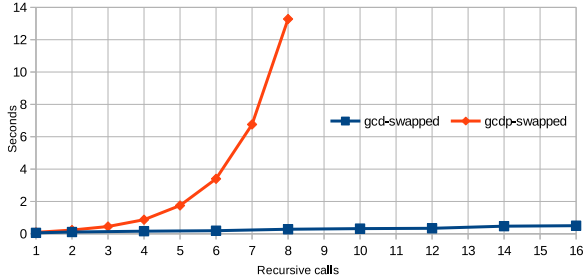
Evaluating the Haskell Funcon Framework 2/2

```
1 procedure gcd (var p, var q) begin
2         if (p == q) then return p;
3   else if (p < q)  then return (gcd (q, p));
4   else                  return (gcd (p-q,q));
5 end
6 print (gcd(22,8));
```

```
1 procedure gcdp(var p, var q) begin
2         if (p == q) then print p;
3   else if (p < q)  then gcdp (q, p);
4   else                  gcdp (p-q,q);
5 end
6 gcdp(22,8);
```

Evaluating the Haskell Funcon Framework 2/2





Conclusions

- **CBS** and **Funcons-beta** have successfully been applied in a number of case studies: OCaml-Light, MiniJava, SIMPLE, ...
- The **Funcons-beta** library covers all features of the imperative, procedural, object-oriented, and functional programming paradigms and more
- The **Haskell Funcon Framework** is used to experiment with funcon definitions and to test language definitions

Please approach or e-mail me when interested in applying funcons!

ltvanbinsbergen@acm.org

Executable Component-Based Semantics

L. Thomas van Binsbergen[1], Neil Sculthorpe[2], Peter D. Mosses[3]

ltvanbinsbergen@acm.org

[1]Royal Holloway, University of London ⟶ Centrum Wiskunde & Informatica

[2]Swansea University ⟶ Nottingham Trent University

[3]Swansea University ⟶ TU Delft (Visiting)

17 March, 2020