Tool Support for Component-Based Semantics

Thomas van Binsbergen

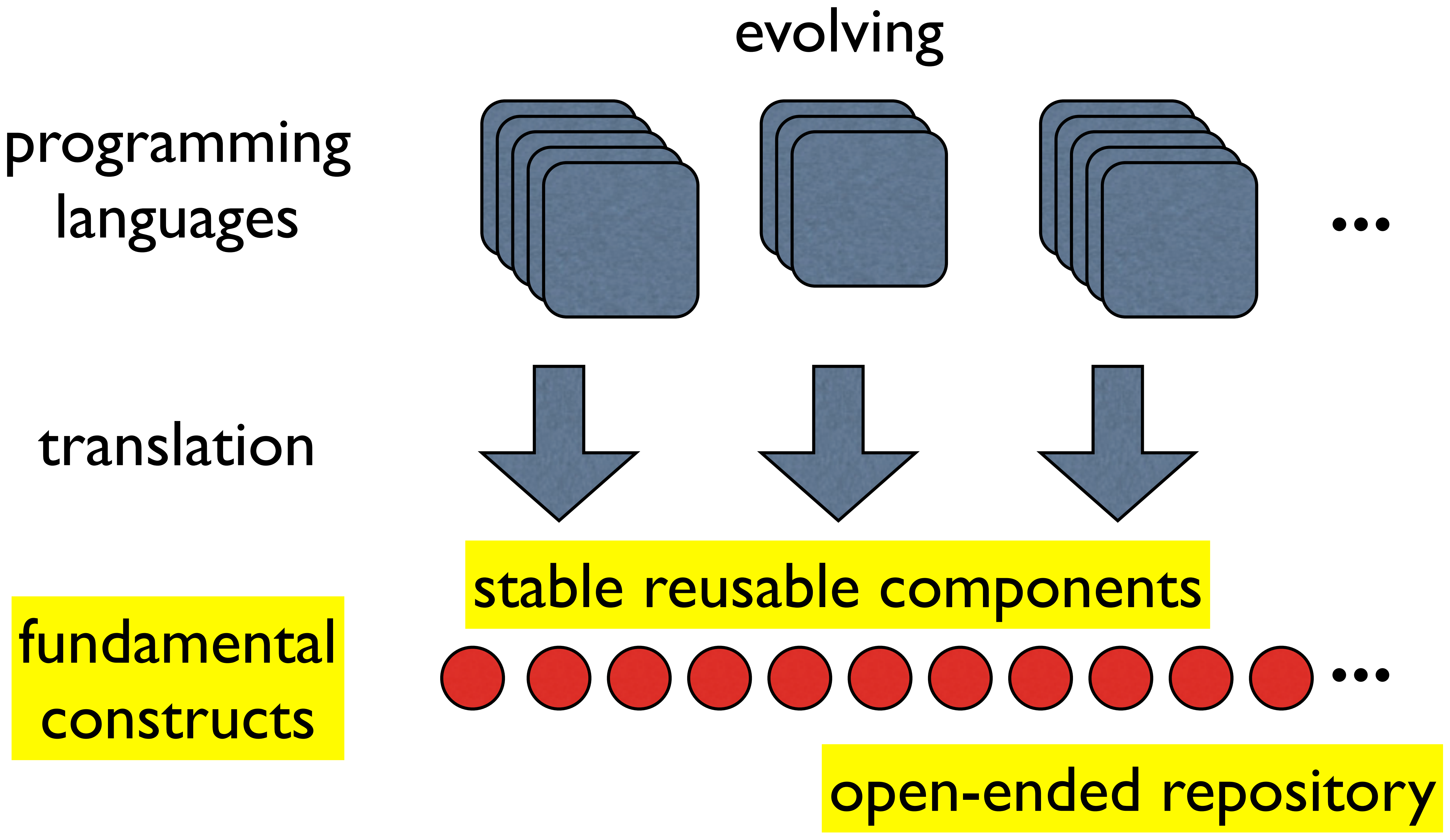
Royal Holloway, University of London, UK

Peter Mosses, Neil Sculthorpe

Swansea University, UK

NWPT 2015, Reykjavík, Iceland, October 2015

What is component-based semantics?



What are fundamental constructs?

Computation primitives and combinators

- sequential, if-then-else, while, bind, bound, scope, allocate-initialised-variable, store-value, stored-value, …

Value constants, operations, and types

- booleans, is-less-or-equal, not, integers, integer-add, ( ), environments, map-unite, variables, values, types, …

Values can be implicitly lifted to computations

- e.g.: while(**not**(stored-value(bound("b"))), …)

CBS: component-based specification
– *denotational-style translation*

```
stmt ::= block
      |  id '=' aexp ';'
      |  'if' '(' bexp ')' block ('else' block)?
      |  'while' '(' bexp ')' block
      |  stmt stmt
```

abstract syntax

translation function

```
evaluate[[ _:aexp ]] : =>integers
```

```
execute[[ I '=' AExp ';' ]] =
  store-value(bound(id[[ I ]]), evaluate[[ AExp ]])
```

fundamental constructs

translation equation

Fundamental construct specifications – *using CBS variant of modular SOS*

$\boxed{\textit{Entity}\ \ \textbf{environment}(\rho : \text{environments}) \vdash \_ \rightarrow \_}$

Funcon **scope**(_ : environments, _ : ⇒*T*) : ⇒*T*

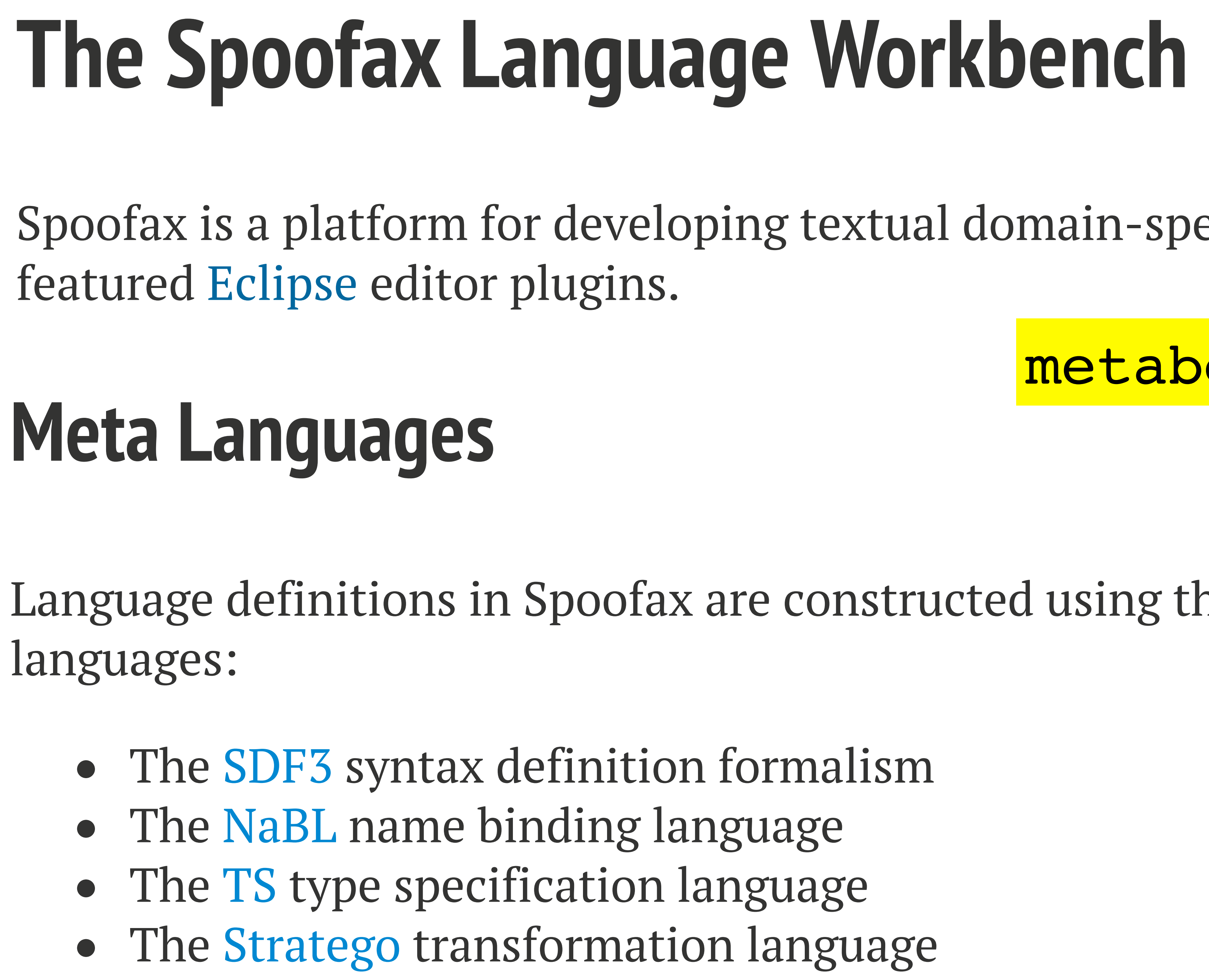
$$\frac{\text{environment}(\rho'/\rho) \vdash X \rightarrow X'}{\text{environment}(\rho) \vdash \textbf{scope}(\rho', X) \rightarrow \textbf{scope}(\rho', X')}$$

$\textbf{scope}(\rho, V{:}\text{values}) \rightarrow V$

a highly reusable component

Tool support

Tool support for CBS: IDE



The Spoofax Language Workbench

Spoofax is a platform for developing textual domain-specific languages with full-featured Eclipse editor plugins.

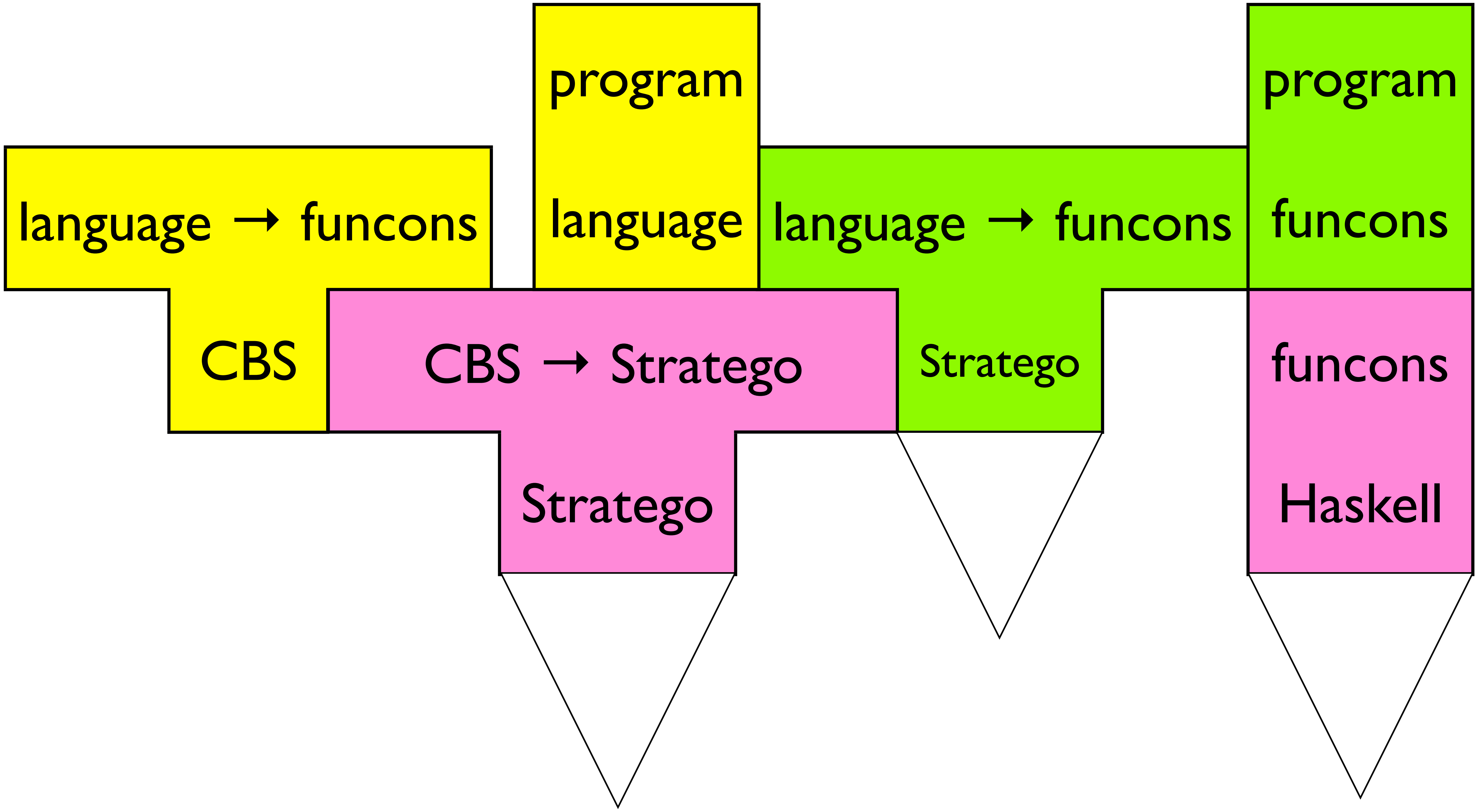
`metaborg.org/spoofax`

Meta Languages

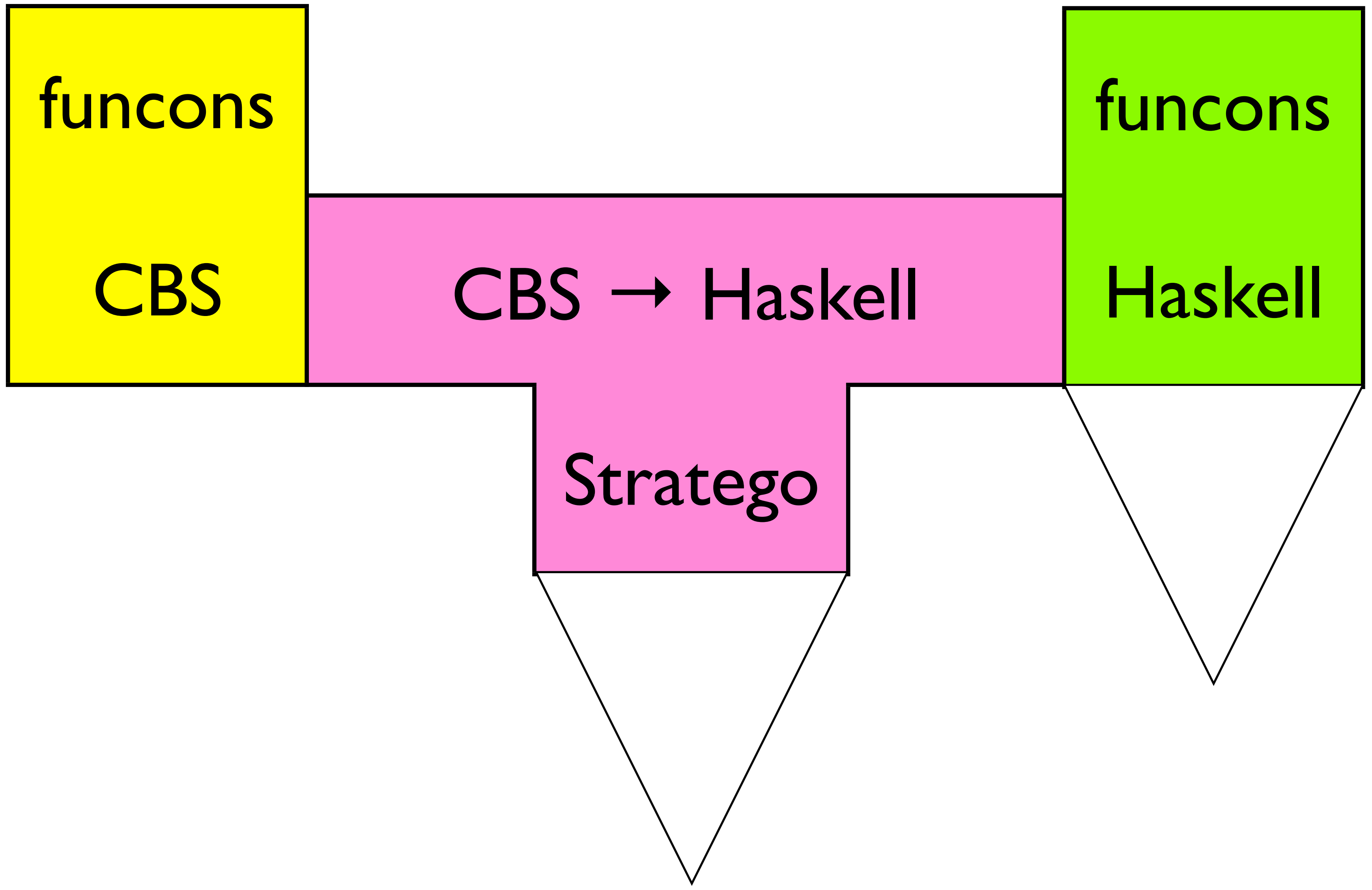
Language definitions in Spoofax are constructed using the following meta-languages:

- The SDF3 syntax definition formalism
- The NaBL name binding language
- The TS type specification language
- The Stratego transformation language

Current tool support: CBS-based program execution



Future tool support: CBS-based interpreter generation



Demo

- browsing/editing CBS specifications
- translating programs to funcons
- executing funcons
- generating translators

Conclusion

Current version of CBS tools available for download

- `www.plancomps.org/nwpt2015-tsc`
- tested with pre-packaged Spoofax/Eclipse distribution

CBS scales up to larger languages

- CAML LIGHT [*Modularity'14 special issue, Trans. AOSD, 2015*]
- C# [*work in progress*]

Fundamental constructs (funcons) appear to be

highly reusable components