A Language Designer's Workbench
A One-Stop-Shop for Implementation and Verification of Language Designs
Eelco Visser1, Guido Wachsmuth2, Andrew Tolmach3, Pierre Neron1, Vlad Vergu1, Augusto Passalaqua1, Gabriël Konat1

Syntax with SDF3
- The syntax of a language defines the structure of the text representation of valid programs.
- A parsing algorithm that generates the abstract syntax tree from the text source code is usually the only definition of the syntax rules.
- SDF3 uses both templates, to define context free grammar productions including layout for pretty printing, and declarative rules for disambiguation.

Name Binding with NaBL
- The name binding rules of a language describes how identifiers refer to their definition.
- A resolution algorithm is usually implicit and only appears inside the compiler or the type checker.
- NaBL uses rules relying on the following basic language independent concepts to identify definitions, references, and scopes to restrict the visibility of definitions.

Type System with TS
- The type system assign types to the different elements of a programs and describes how these elements can be connected safely.
- A derived type checking/inference algorithm can be used in the IDE and the compiler to verify the static correctness of a program.
- TS inductive rules define the type system; these rules can refer to the type of the definitions from NaBL.

Dynamic Semantics with DynSem
- The dynamic semantics of a language describe the dynamic behavior of the programs on a concrete machine.
- Often the compiler or interpreter implementation stands as the only definition of the dynamic semantics.
- In DynSem, the semantics are defined by declarative rules based on the framework of implicitly-modular structural operational semantics developed by P. Mosses.

Automatic generation of new languages machinery from simple declarative rules

Development (Eclipse)
Easily write and edit programs with an Eclipse plugin for interactive development
Spoofax extends Eclipse to connect syntactic and semantics editor services. These services give feedback as the programmer types; they include:
- syntactic highlighting
- code views
- program navigation through references
- semantic code completion
- error detection (unresolved variables, type errors...)

Execution (Java)
Efficiently execute programs with a Java-based abstract syntax tree interpreter
Implicit structural operational semantics rules from DynSem are transformed into constructor specific rules:
Ifz(e1, e2, e3) --> v
where e1 --> I(i),
[1 = 0, e2 --> v] + [i=0, e3 --> v]
The evaluation methods directly derive from such rules.

Verification (Coq)
Verify the correctness of the definitions with a model and proofs of type safety in Coq
The generated model includes:
- Term definition in a well-formedness predicate
- A lookup relation to represent name resolution
- An inductive typing predicate
- An inductive environment-based semantics relation.

inductive C := (* constructors *) | ParamC | FunC | FixC | AppC | IfzC ...
inductive term := Type := | Co : C -> list term -> term | Id : I -> term (* I is identifier type *)
inductive ws_term := sort -> term + Prop := | Co ws cn ss ts : | Co ws cn ss ts : sig cn = (ss,s) + (sig is signature +)
forall ws_term ss ts + ws_term s (Co cn ts)....
inductive typedecl := term + term + Prop := ...
inductive eval := env + term + val + Prop :=

Theorem type_preservation :
forall e v ty, ws_term Expr8 Expr8 =>
sound e => eval e v => typedecl e ty => val_type v ty