THE SPOOFAX NAME BINDING LANGUAGE

Name Binding

Name binding comprises the association of uses of names – such as variables, methods and classes – to their definitions. The purpose is performing static name analysis and providing IDE services such as error checking, code navigation and code completion.

At source code level the structure of a program communicates the associations and named scopes intuitively. For example, classes open a scope for methods and fields, methods open a scope for local variables.

Navigation

Reference resolution permits IDE users to navigate code from variable, method or class use-site to definition location. Name analysis results in navigable associations.

```
interface Env {
  Env add(string id, int val);
}

interface Expr {
  int eval(Env env);
}

class Plus : Expr {
  Expr l;
  Expr r;
  int eval(Env env) {
    return l.eval(env) + r.eval(env);
  }
}
```

refers to

refers to

refers to

Error checking

Usage of names that require associations to missing or invisible definitions are marked in the editor and reported as errors.

```
class Let : Exp {
  sting name;
  Expr t;
  Expr body;

  class add(Env env) {
    Env newEnv = env.add(nam,
                         t.eval(env));
    return body.eval(newEnv);
  }
}
```

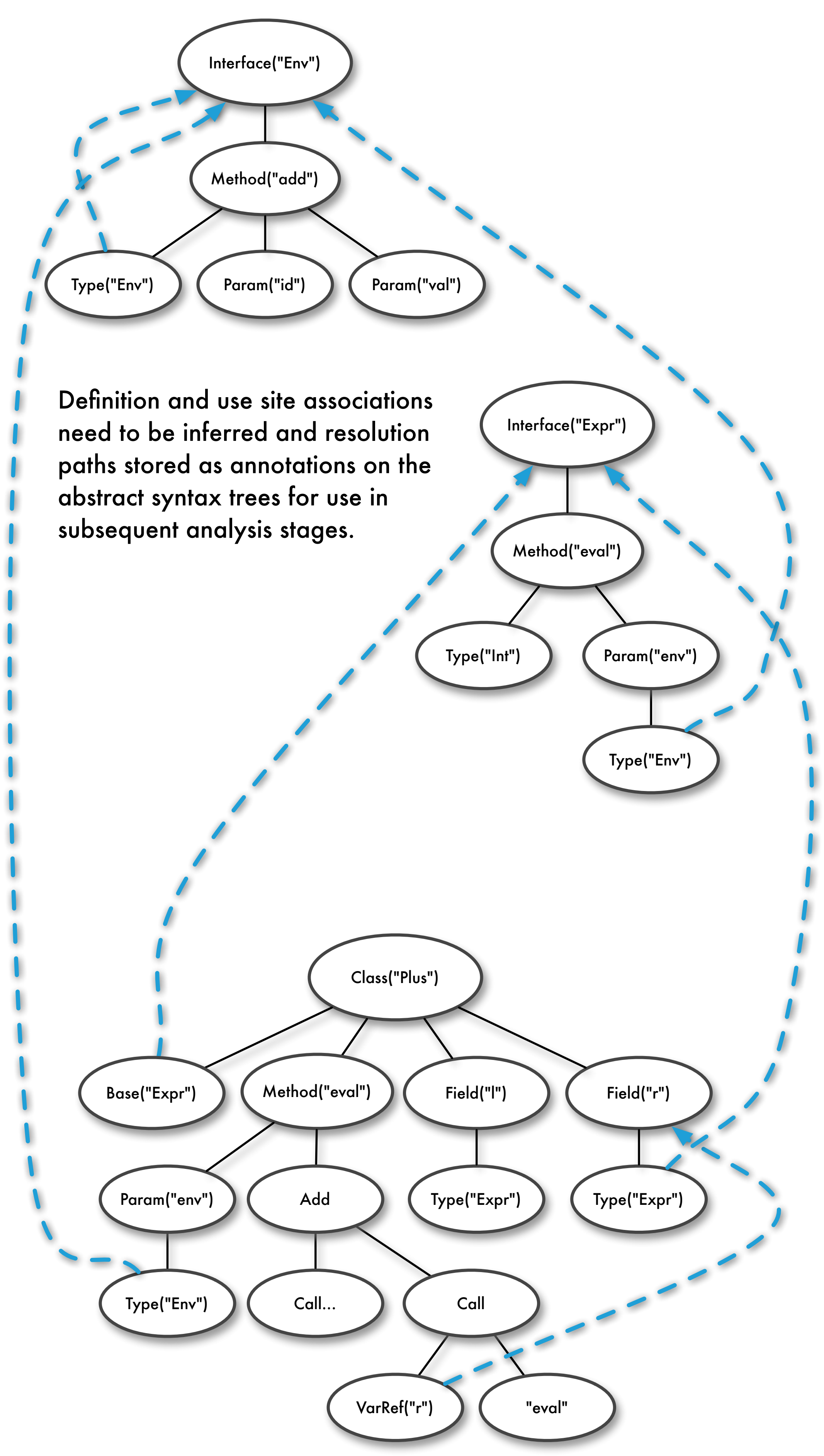
Code completion

Inline code completion provides suggestions while editing. The inline code completion service provides suggestions while editing. Name analysis is required to determine the defined entities that are applicable at the completion location.

```
class Let : Exp {
  Expr r;
  int eval(Env env) {
    return l.e(env) + r.eval(env);
  }        eval
}
```

Name Resolution

Name resolution is a program analysis that resolves names in abstract syntax trees, resulting in (some representation of) a tree with references from uses to definitions.



Definition and use site associations need to be inferred and resolution paths stored as annotations on the abstract syntax trees for use in subsequent analysis stages.

Abstract syntax trees communicate name scopes and associations less intuitively than source code. The implementation of name binding is typically much more intricate than manual examination in the presence of visual structural cues.

```
Class("Plus", "Expr"
 , [ Field(Type("Expr"), "l")
   , Field(Type("Expr"), "r")
   , Method(
       Type("Int")
     , "eval"
     , [Param(Type("Env"), "env")]
     , [ Return(
           Add(
             Call(VarRef("l"), "eval",
               [VarRef("env")])
           , Call(VarRef("r"), "eval",
               [VarRef("env")])
         ))])])
```

Classical Approaches

Existing approaches for definition of name binding and scope, use programmatic encodings of *name resolution algorithms*, which hide binding and scope concepts. DSLs for compiler construction have focused on reducing the overhead and accidental complexity of these programmatic encodings. Some examples:

Inference Rules

The inference rules featured in mathematical language definitions encode binding and scope by carrying around name binding environments.

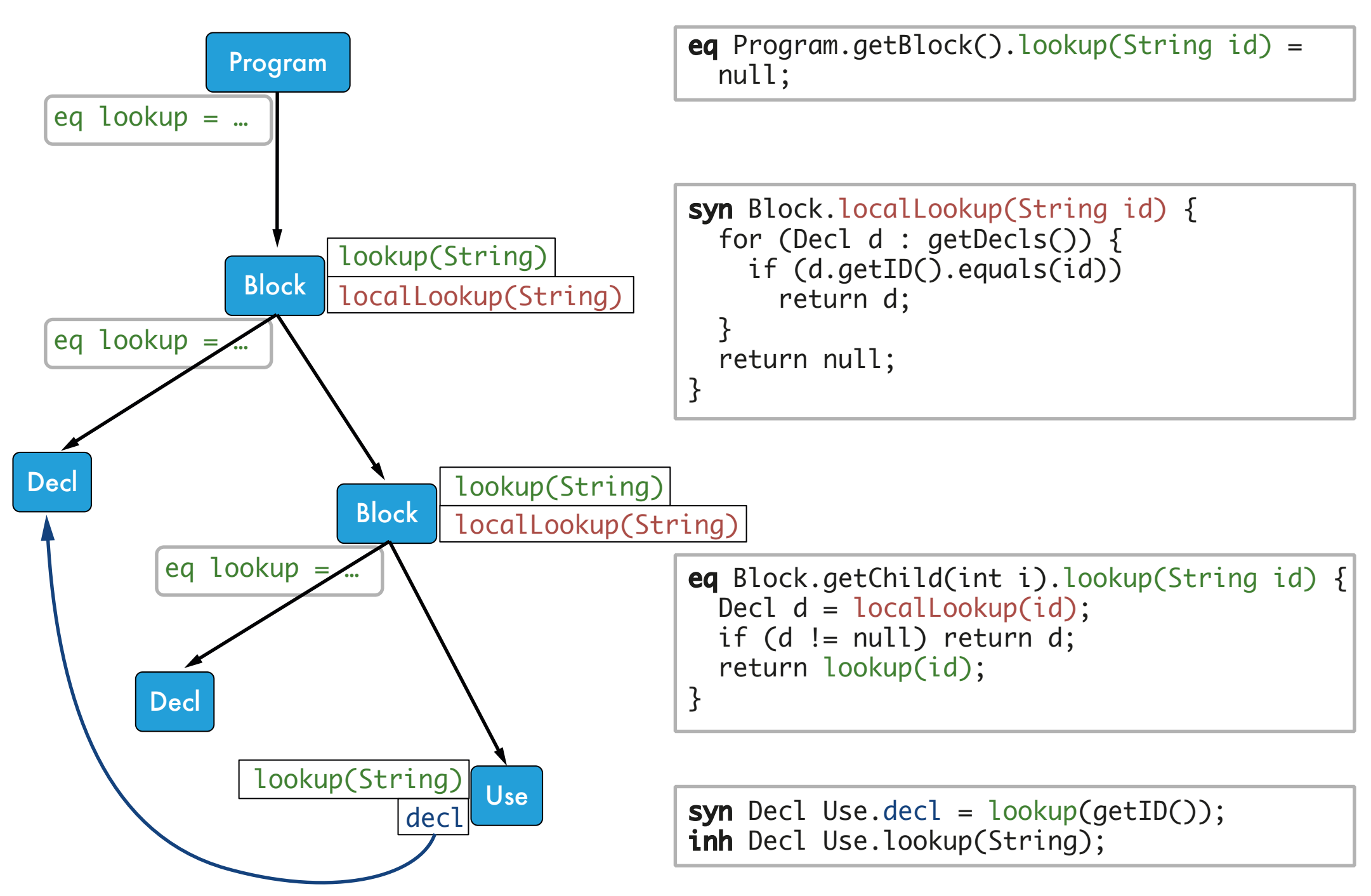
$$\text{(Env } \emptyset\text{)}\quad \frac{}{\emptyset \vdash \diamond} \qquad \text{(Env } x\text{)}\quad \frac{\Gamma \vdash A \qquad x \notin dom(\Gamma)}{\Gamma, x{:}A \vdash \diamond}$$

$$\text{(Type Const)}\quad \frac{\Gamma \vdash \diamond \qquad K \in Basic}{\Gamma \vdash K} \qquad \text{(Type Arrow)}\quad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \rightarrow B}$$

$$\text{(Val } x\text{)}\quad \frac{\Gamma', x{:}A, \Gamma'' \vdash \diamond}{\Gamma', x{:}A, \Gamma'' \vdash x{:}A} \qquad \text{(Val Fun)}\quad \frac{\Gamma, x{:}A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \rightarrow B} \qquad \text{(Val Appl)}\quad \frac{\Gamma \vdash M : A \rightarrow B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$$

Reference attribute grammars

Reference attribute grammar [3] (e.g. as supported by the JastAdd [2]) avoid carrying around environments by defining lookup attributes for individual names, persisting the results as reference attributes pointing to the definition of a name. However, these attribute definitions are still concerned with defining lookup algorithms.



```
eq Program.getBlock().lookup(String id) =
  null;
```

```
syn Block.localLookup(String id) {
  for (Decl d : getDecls()) {
    if (d.getID().equals(id))
      return d;
  }
  return null;
}
```

```
eq Block.getChild(int i).lookup(String id) {
  Decl d = localLookup(id);
  if (d != null) return d;
  return lookup(id);
}
```

```
syn Decl Use.decl = lookup(getID());
inh Decl Use.lookup(String);
```

Rewriting strategies with Dynamic Rules [1]

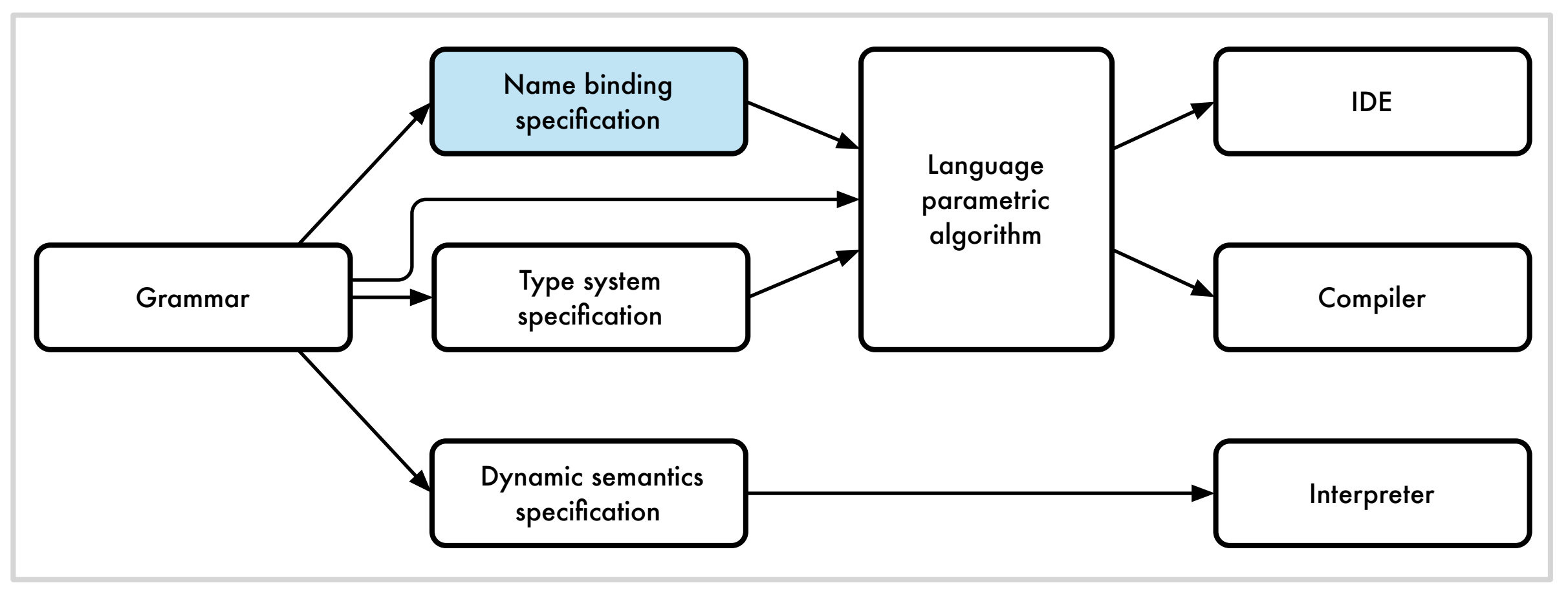
Strategic rewriting as provided by the Stratego transformation language [6], avoids the overhead of abstract syntax tree traversal by means of generic traversal strategies, and uses scoped dynamic rewrite rules to define mappings from names to unique names or types. However, the resulting definitions are rather algorithmic and tangle multiple concerns.

```
rename-top = alltd(rename)
rename :
  |[ var x : srt ]| -> |[ var y : srt ]|
  where y := <add-naming-key(|srt)> x
rename :
  |[ define page x (farg1*) { elem1* } ]| ->
  |[ define page x (farg2*) { elem2* } ]|
  where {| Rename
        : farg2* := <map(rename-page-arg)> farg1*
        ; elem2* := <rename-top> elem1*
        |}
rename = Rename
add-naming-key(|srt) : x -> y
  where y := x{<newname> x}
      ; rules (
          Rename : Var(x) -> Var(y)
          TypeOf : y -> srt
        )
```

Name Binding Language

The Spoofax Name Binding Language (NBL) [5] is a metalanguage for declaratively specifying name binding and scope rules. Language engineers need not be concerned with the mechanics of name binding algorithms, but can focus on the name binding concepts of the language. NBL is integrated in the Spoofax Language Workbench [4], but aspires to be a universal language for name binding, as BNF is for syntax.

NBL currently generates language specific implementations in Stratego of a language-parametric algorithm for static name resolution, error checking and contextual content completion.



Grammar
Name binding specification
Type system specification
Dynamic semantics specification
Language parametric algorithm
IDE
Compiler
Interpreter

In NBL, name binding is specified in terms of namespaces, definition sites, use sites and scopes. Naming rules are specified in terms of pattern matches on abstract syntax tree nodes and support integration with the type system.

```
namespaces Class Method Field Variable

rules // Classes

  Class(x, _, _) :
    defines Class x of type Type(x)
    scopes Field, Method

  Type(x):
    refers to Class x
```

NBL supports declarative specification of scope composition and inheritance. Resolution paths are statically composed from hierarchical scope references.

```
  Base(x) :
    refers to Class x
    imports Field from Class x {transitive}
    imports Method from Class x {transitive}
```

The integration with the type system allows full specification of name resolution paths, while resolution path prioritization and fine-grained scoping control permits correctness without code duplication.

```
  Field(t, x) :
    defines Field x of type t

  FieldAccess(exp, f) :
    refers to Field f in Class e
    where exp has type Type(e)

  Var(t, x, _) :
    defines Variable x of type t
                  in subsequent scope

  VarRef(x) :
    refers to Variable x
    otherwise refers to Field x
```

http://www.Spoofax.org

Gabriël D. P. Konat — g.d.p.konat@student.tudelft.nl
Vlad A. Vergu — v.a.vergu@tudelft.nl
Lennart C. L. Kats — l.c.l.kats@tudelft.nl
Guido H. Wachsmuth — g.h.wachsmuth@tudelft.nl
Eelco Visser — e.visser@tudelft.nl

References

1. M. Bravenboer, A. van Dam, K. Olmos, and E. Visser. Program transformation with scoped dynamic rewrite rules. FUIN (2006)
2. T. Ekman and G. Hedin. Modular name analysis for Java using JastAdd. GTTSE (2006)
3. U. Kastens and W. M. Waite. Modularity and reusability in attribute grammars. Acta Informatica (1994)
4. L. C. L. Kats and E. Visser. The Spoofax language workbench: rules for declarative specification of languages and IDEs. OOPSLA (2010)
5. G. Konat, L. Kats, G. Wachsmuth, and E. Visser. Language-parametric name resolution based on declarative name binding and scope rules. SLE (2012)
6. E. Visser. Program transformation with Stratego/XT: Rules, strategies, tools, and systems in Stratego/XT 0.9. (2003)
7. L. Cardelli. Type Systems. ACM Computing Surveys (1996)

TU Delft — Delft University of Technology