Interactive Disambiguation of Meta Programs with Concrete Object Syntax

Lennart Kats (TUDelft)
Karl T. Kalleberg (KolibriFX)
Eelco Visser (TUDelft)

TU Delft
Delft University of Technology

KolibriFX

Meta-programming

Meta-programming with Template Engines

```
<<FOREACH ... AS class>>
<<FILE "samples/petclinic/" + class.name + ".java">>
package samples.petclinic;

publc class <<class.name>> {
    private long id;
    public void setId(long id) {
      this.id = id;
    }
    public long getId() {
      return id;
    }
}
<<ENDFILE>>
<<ENDFOREACH>>
```

Works for any lanugage!

Only code *generation*

Lack of compositionality

Easy to read, Easy to write

Works for any language!

Meta-programming with Abstract Syntax

```
for (Output c : ...) {
  CompilationUnit u =
    newCompilationUnit(
      "samples/petclinic/" + c.name + ".java
  u.setPackageName("samples.petclinic");
  ClassDec cd = u.newClassDec(c.name);
  cd.newField(PRIVATE, long.class, "id");
  MethodDec m =
    cd.newMethod(PUBLIC, void.class, "setId"
  m.newParameter(long.class, "id");
  m.newStatement(newAssign(newFieldAccess(TH
newVariableAccess("id")));
  m = cd.newMethod(PUBLIC, long.class, "getI
  m.newStatement(newReturn(newVariableAccess("id")));
  ...
}
```

Always well-formed!

Further transformation still possible

Hard to read, Hard to write

Highly verbose

Meta-programming with Concrete Object Syntax

[Visser, GPCE'02]

Any meta language (here: Java)

```
for (Output c : ...) {
  CompilationUnit u =
    |[ package samples.petclinic;

       publc class $[c.name] {
         private long id;
         public void setId(long id) {
           this.id = id;
         }
         public long getId() {
           return id;
         }
       }
    ]|
  ...
}
```

Any object language (here: Java)

Always well-formed!

Meta-programming with Concrete Object Syntax

[Visser, GPCE'02]

```
for (Output c : ...) {
  CompilationUnit u =
    |[ package samples.petclinic;

       public class $[c.name] {
         private long id;
         public void setId(long id) {
           this.id = id;
         }
         public
       ]|
  ...
}
```

Any meta language (here: Java)

Easy to read,
Easy to write(?)

Transform

```
for (Output c : ...) {
  CompilationUnit u =
    newCompilationUnit(
      "samples/petclinic/"
va");
    setPackageName("samples
lassDec cd = u.newClassD
  cd.newField(PRIVATE, long
  MethodDec m =
    cd.newMethod(PUBLIC, void.class, "setId");
  m.newParameter(long.class, "id");
  m.newStatement(newAssign(newFieldAccess
(THIS, "id"), newVariableAccess("id")));
```

Further transformation still possible

Prerequisites

Parser for meta language + object language

created using grammar composition:

- meta language grammar
- object language grammar
- mixin grammar for quotations and anti-quotations

Example SDF Mixin Grammar

```
module Stratego-Java

imports
  Stratego
  Java

exports context-free syntax

  "|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
  "|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
  "|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}

  "$[" Term "]" -> ClassDec {cons("FromMetaExpr")}
  "$[" Term "]" -> BlockStm {cons("FromMetaExpr")}
  "$[" Term "]" -> CompUnit {cons("FromMetaExpr")}
```

Problem: Ambiguity

Which is it?

```
|[
    class X {
    }
]|
```

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Problem: Ambiguity

```
class X {

}
class Y {
}
```

Which is it?
Just another class?

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Problem: Ambiguity

```
class X {
}
```

Which is it?
Just another class?
A compilation unit?

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Problem: Ambiguity

```
void foo () {
    class X {

    }
}
```

Which is it?
Just another class?
A compilation unit?
Or a class declaration statement?

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Problem: Ambiguity

```
|[
    class X {
    }
]|
```

Which is it?
Just another class?
A compilation unit?
Or a class declaration statement?

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Disambiguation

Tag-based Disambiguation

```
"ClassDec" "|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"CompUnit" "|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"BlockStm" "|[" BlockStm "]|" -> Term {cons("ToMeta
```

```
CompUnit |[ class Foo {} ]|
```

Simple, generic

Verbose

Need to know the tag names

Type-based Disambiguation

[Bravenboer et al. '05, Vinju '05]

```
CompUnit c = |[ class Foo {} ]|;

fooMethod(|[ class Foo {} ]|);
```

More concise!

Need to know the type names

Requires special type checker

Often relies on heuristics

Interactive Disambiguation

```
rules

  webdsl-action-to-java-bean:
    |[ action x_action(farg*) {stat* } ]| ->
    |[ package pkgname;

       import pkgname2.*;

       @Stateful @Name("~x_actionBean")
       public class x_ActionBean implements x_Action {

         @Logger private Log log = initLog();

         RuleManager rules;

         @PersistenceContext(type = EXTENDED)
         private EntityManager entityManager;

         public String x_action() {

         @Remove @Destroy
         public void destroy() {}
       }
    ]|
    where pkgname      := <BeanPackage>;
          pkgname2     := <DomainPackage>;
          bstm*        := <statements-to-java> stm*;
```

Addresses discovery issue

Complementary approach

Spoofax

1. User writes meta program

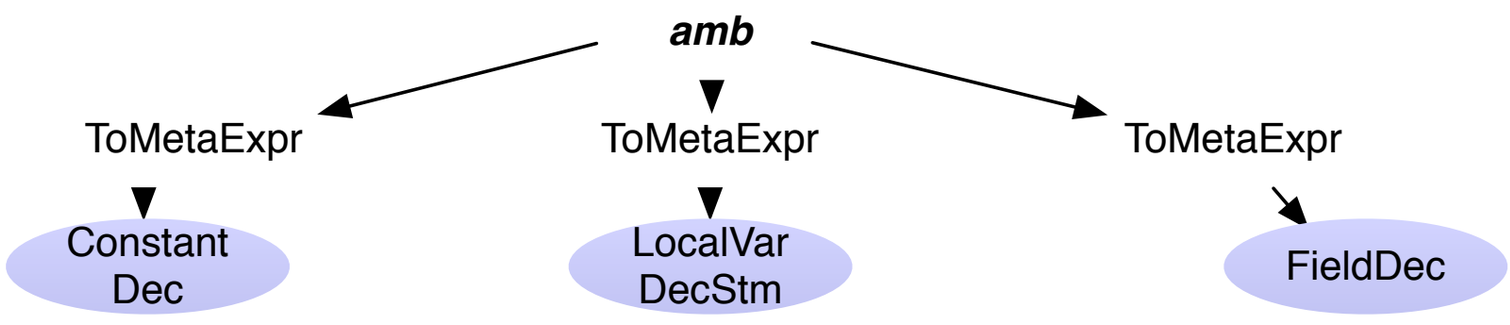
```
property-to-field:
  |[ $[x] : $[t] ]| ->
  |[ @Property $[t] $[x]; ]|
```

Parsing with Ambiguity

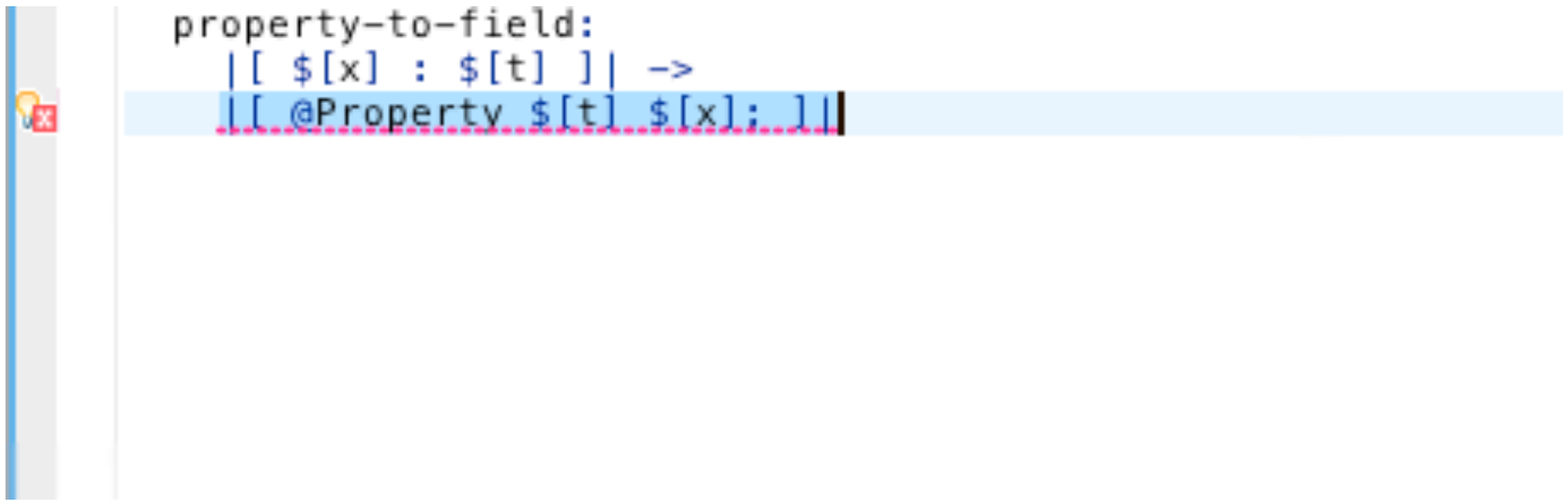
```
property-to-field:
  |[ $[x] : $[t] ]| ->
  |[ @Property $[t] $[x]; ]|
```

```
"|[" ConstantDec    "]|" -> Term {cons("ToMetaExpr")}
"|[" LocalVarDecStm "]|" -> Term {cons("ToMetaExpr")}
"|[" FieldDec       "]|" -> Term {cons("ToMetaExpr")}
```

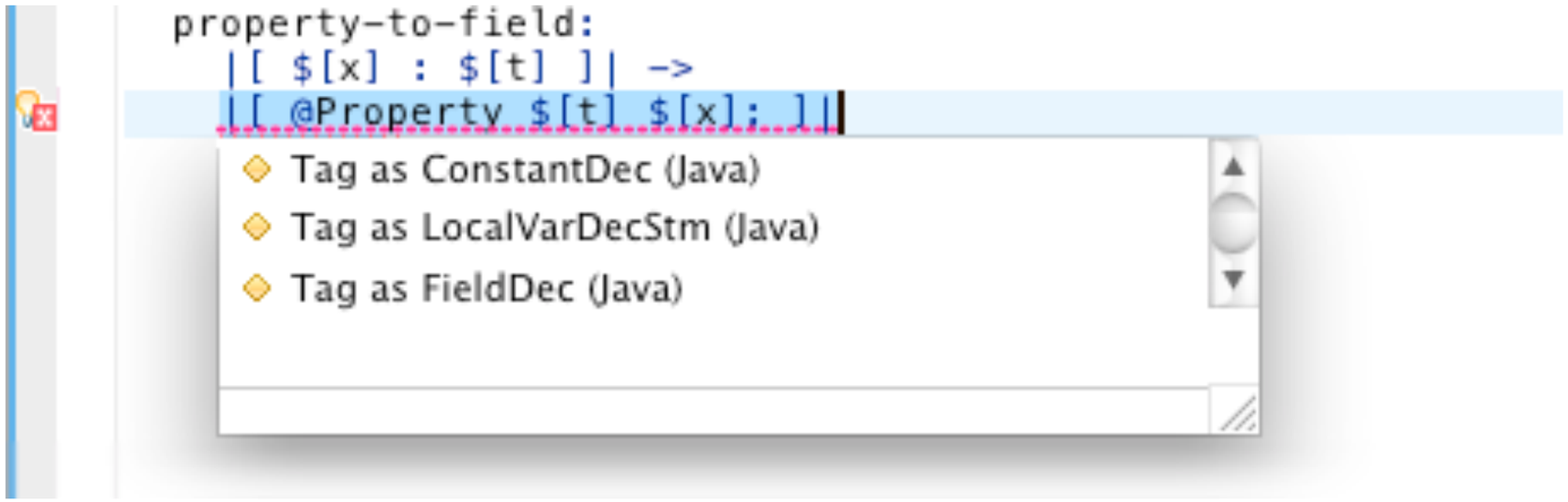
Parse forest from generalized parser (SGLR):



2. IDE detects ambiguity



3. User selects intention



4. IDE explicitly disambiguates meta program

```
property-to-field:
  |[ $[x] : $[t] ]| ->
    FieldDec |[ @Property $[t] $[x]: ]|
```

Simple, generic

Addresses name discovery

No heuristics

Mixin Grammar for Interactive Disambiguation

Discovery

```
"|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

```
"ClassDec" "|[" ClassDec "]|" -> Term {cons("ToMetaExpr")}
"CompUnit" "|[" CompUnit "]|" -> Term {cons("ToMetaExpr")}
"BlockStm" "|[" BlockStm "]|" -> Term {cons("ToMetaExpr")}
```

Disambiguation

Perspective

Here, applied with tag-based disambiguation

Also useful with type-based disambiguation

- Get rid of heuristics
- Different transformations needed

Perspective

Link to projectional editing

```
rule typeof_InputFieldReference {
  applicable for concept = InputFieldReference as reference
  overrides false
  child type restrictions << ... >>

  do {
    typeof(reference) :==: <IntegerType
  }
}
```

```
IntegerConceptProperty                    lang: j.m.lang.struc
IntegerConceptPropertyDeclaration         lang: j.m.lang.struc
IntegerConstant                           lang: j.mps.baseLang
IntegerLiteral                            lang: j.mps.baseLang
IntegerType                               lang: j.mps.baseLang
Interface                                 lang: j.mps.baseLang
InterfaceConceptDeclaration               lang: j.m.lang.struc
InterfaceConceptReference                 lang: j.m.lang.struc
InternalSequenceOperation   lang: j.m.baseLanguage.collect
IntersectOperation          lang: j.m.baseLanguage.collect
```

Conclusion

- Leverage IDE
- Useful for meta programs. Programs also?
- Addresses discovery issue
- Avoids heuristics

- www.spoofax.org