Providing Rapid Feedback in Generated Modular Language Environments

Lennart Kats (*me*)
Maartje de Jonge
Emma Nilsson-Nyman
Eelco Visser

OOPSLA 2009 **October 29, 2009**

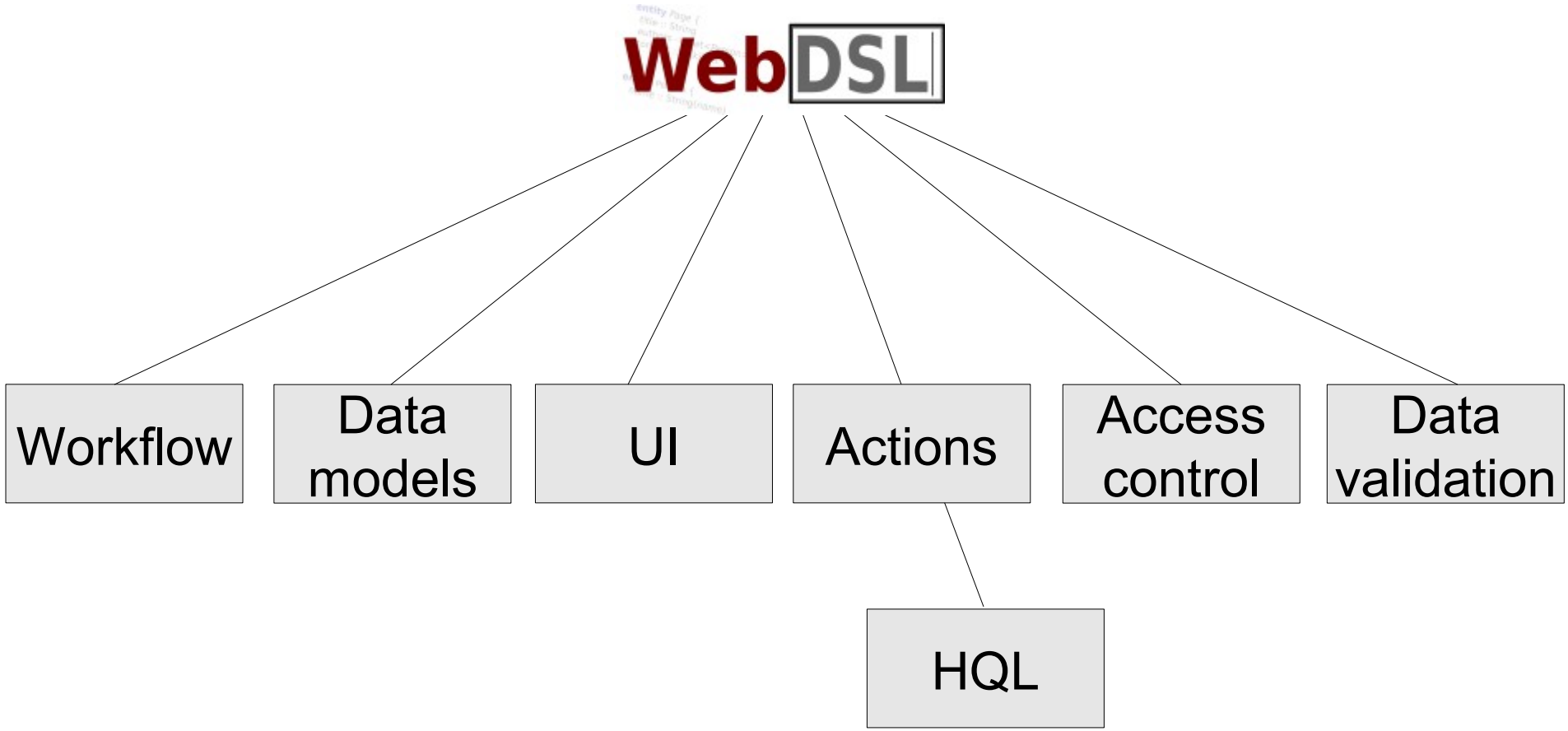
Software Engineering Research Group

TU Delft
Delft University of Technology

Domain-Specific Languages

- Encapsulate domain knowledge
- Eliminate boilerplate code
- Domain-specific analysis / verification / optimization

Integration and Composition of DSLs

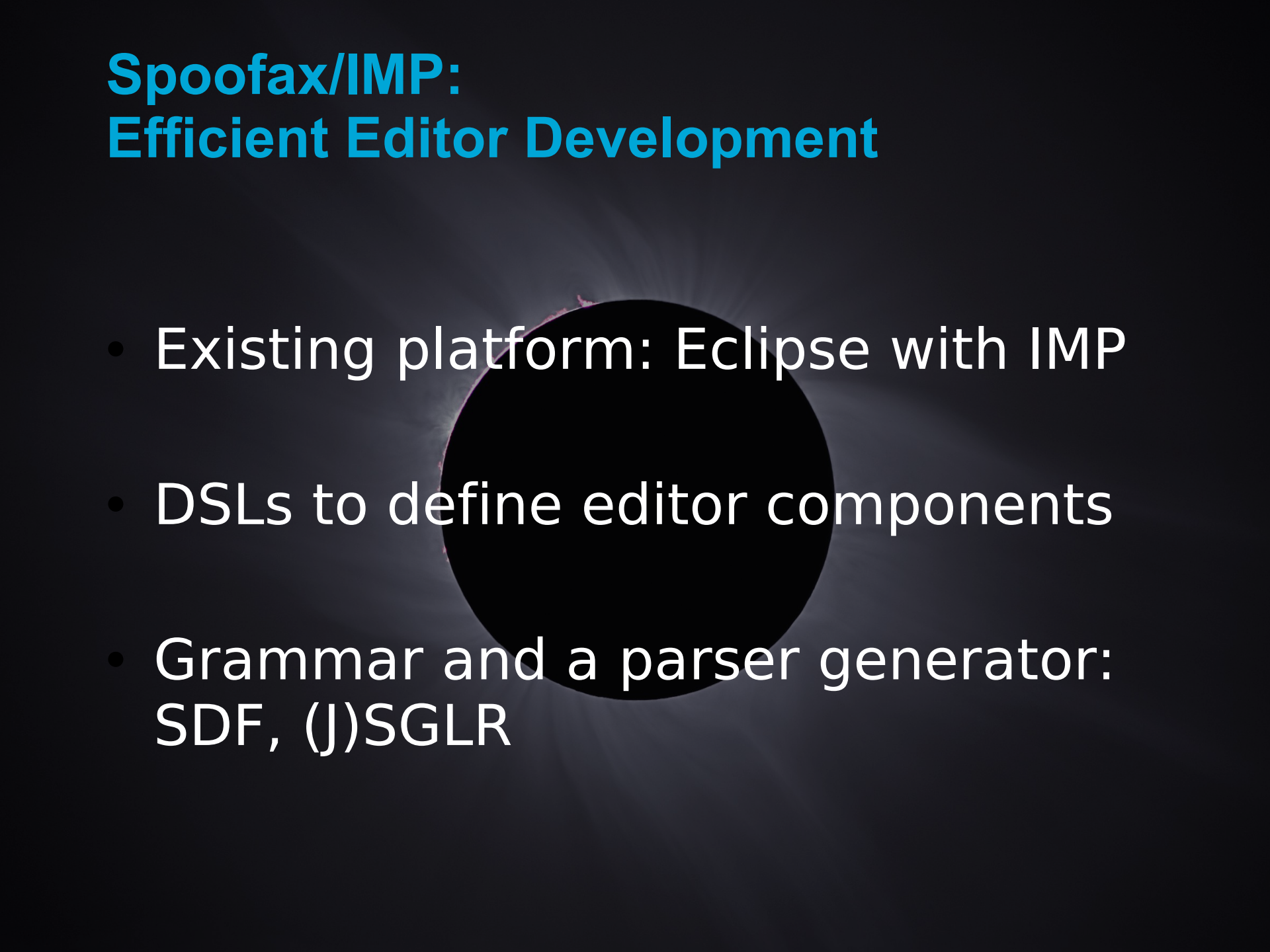


(www.webdsl.org)

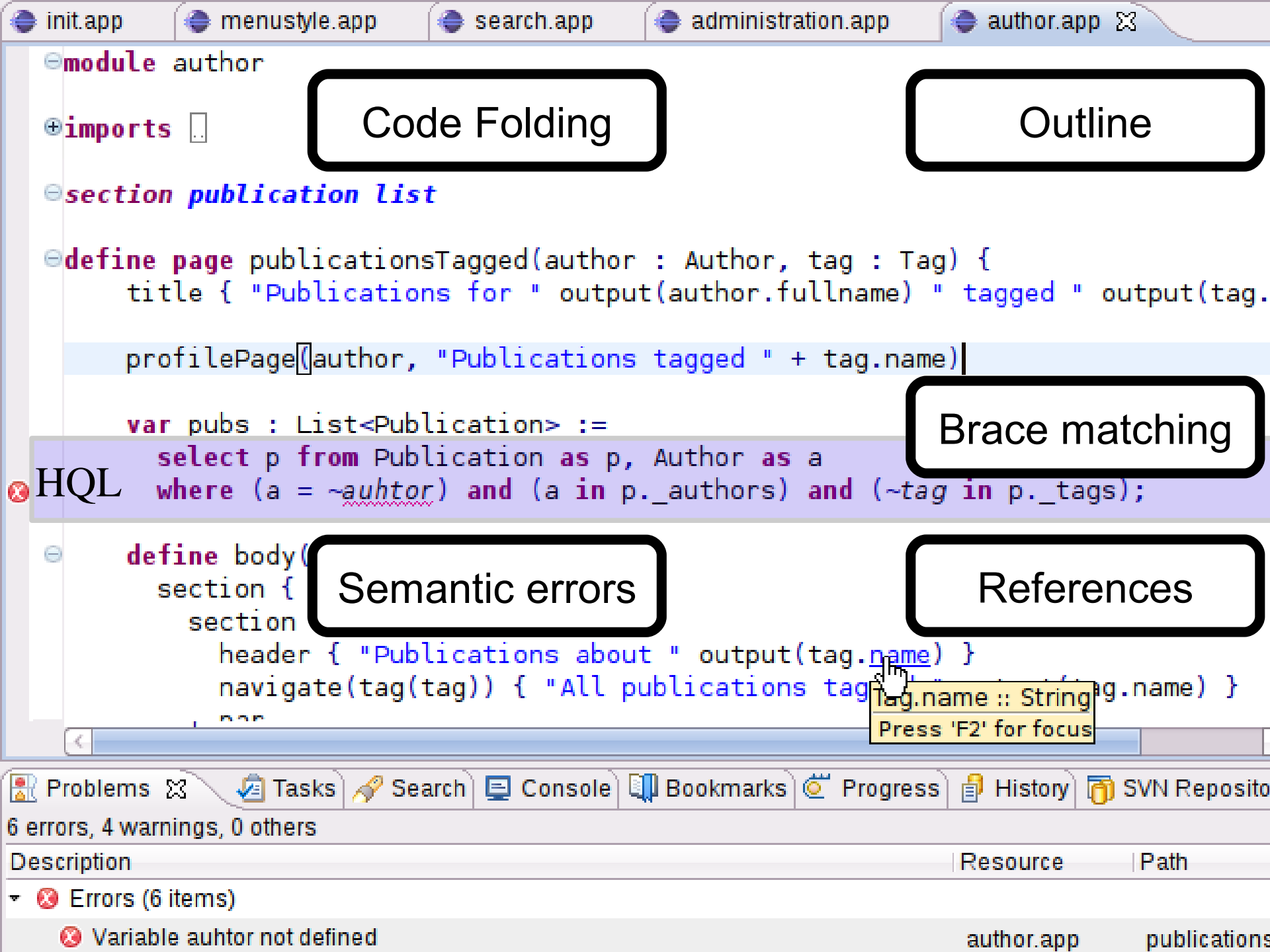
DSLs without IDE support are a *nonstarter!*

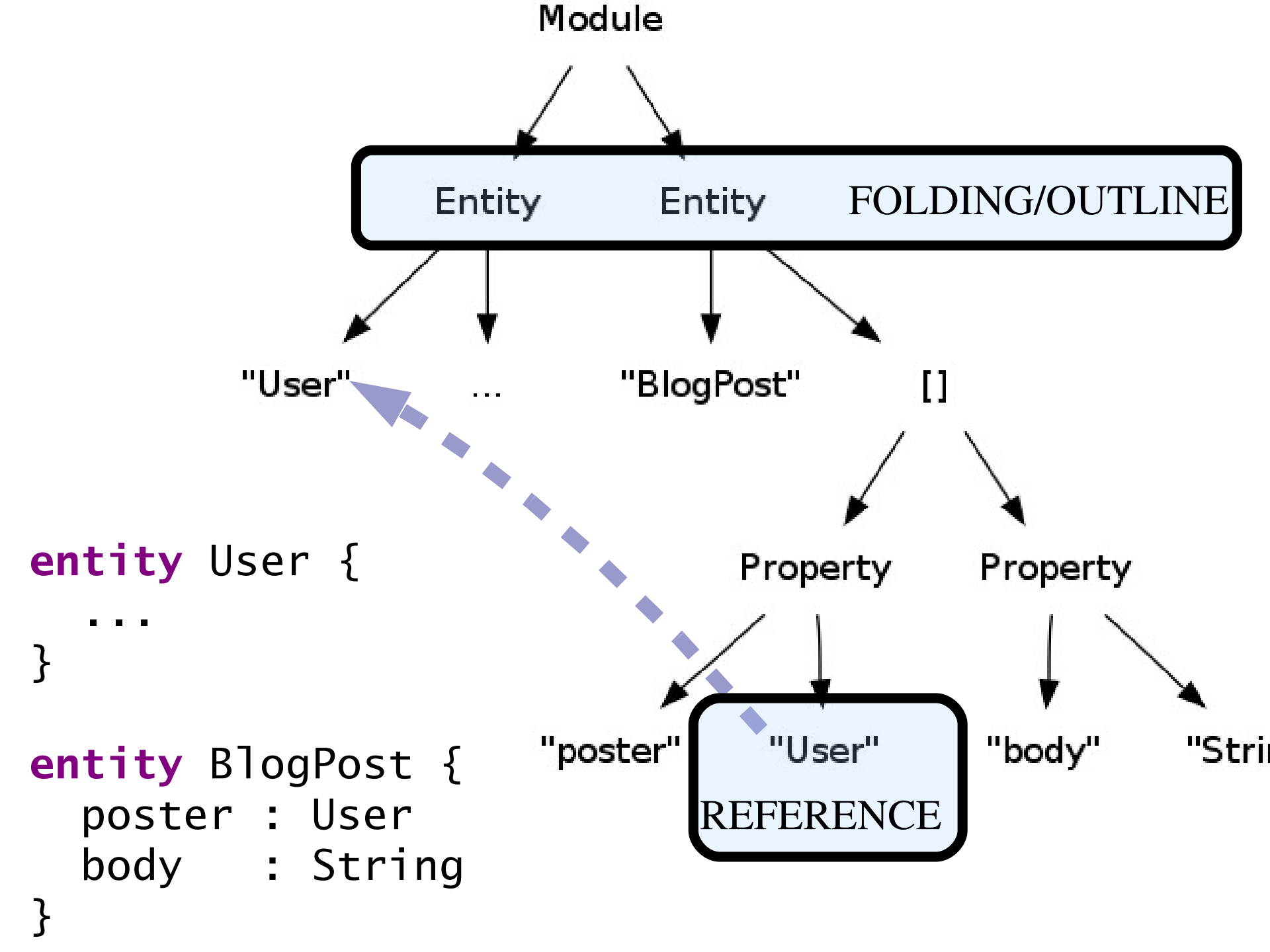
Markus Voelter [Practical Product Lines 2009]

Spoofax/IMP:
Efficient Editor Development



- Existing platform: Eclipse with IMP
- DSLs to define editor components
- Grammar and a parser generator: SDF, (J)SGLR


init.app
menustyle.app
search.app
administration.app
author.app
module author
imports ..
section publication list
define page publicationsTagged(author : Author, tag : Tag) {
title { "Publications for " output(author.fullname) " tagged " output(tag.
profilePage(author, "Publications tagged " + tag.name)
var pubs : List<Publication> :=
select p from Publication as p, Author as a
where (a = ~auhtor) and (a in p._authors) and (~tag in p._tags);
define body(
section {
section
header { "Publications about " output(tag.name) }
navigate(tag(tag)) { "All publications tag
tag.name :: String
Press 'F2' for focus
Code Folding
Outline
Brace matching
HQL
Semantic errors
References
Problems
Tasks
Search
Console
Bookmarks
Progress
History
SVN Reposito
6 errors, 4 warnings, 0 others
Description
Resource
Path
Errors (6 items)
Variable auhtor not defined
author.app
publications




```
entity User {
   ...
}

entity BlogPost {
   poster : User
   body   : String
}
```

What about incorrect and incomplete programs?

NORMAL PARSING

```
entity User {
  name     : String
  password : String
  homepage
}

entity BlogPost {
  poster : User
  body   : String
}
```

What about incorrect and incomplete programs?

```
entity User {
  name     : String
  password : String
  homepage
}

entity BlogPost {
  poster : User
  body   : String
}
```

NORMAL PARSING

ERROR

NEVER PARSED

Parse failure:
No abstract syntax tree

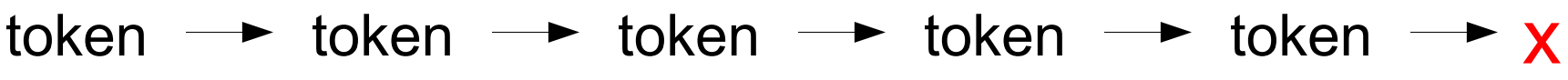
Mini-Demo

Error Recovery in
a data modeling language

Parsing with SDF and SGLR

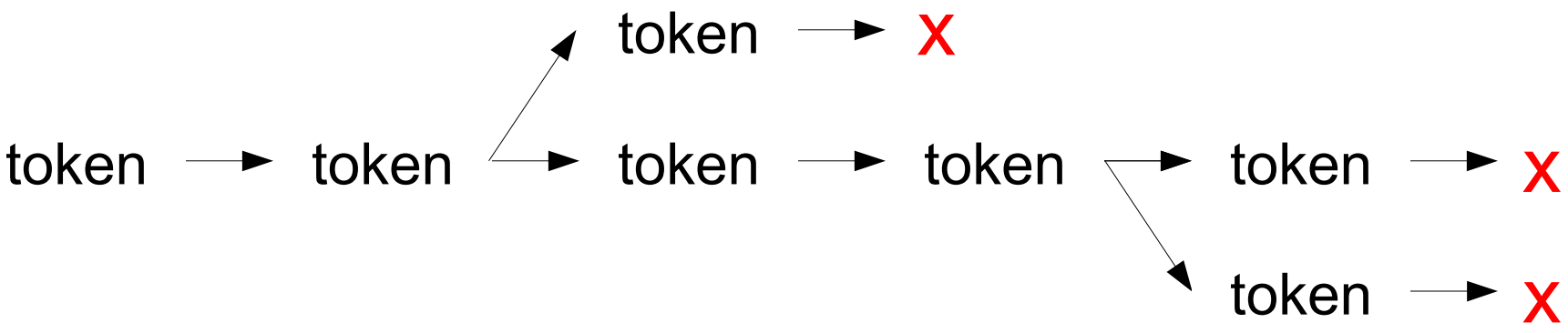
- Language composition without shift/reduce conflicts
- Ambiguities can be inspected
- Declarative disambiguation

Normal LR Parser



Recovery:
Backtracking,
skip/insert tokens, etc.

(S)GLR Parser



Recovery: ?

Do we really need to dive into this intimidating algorithm?

Island Grammars

[Van Deursen & Kuipers, 1999]

- Parse only interesting bits (*Islands*)
- Skip the remaining characters (*Water*)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EXAMPLE.
PROCEDURE DIVISION.                                   WATER
   CALL X.
   YADA.
   YADA YADA.                                         WATER
   CALL Y.
```

Grammar-based
“error recovery”!

Island Grammars

[Van Deursen & Kuipers, 1999]

```
~[\ \t\n]+ → WATER {avoid}
```

Running Example: Java

Error Recovery Recipe

1. Take the entire Java grammar
2. Add water
3. Mix it together

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX))
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

```
// missing {
```

ERROR

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;
```

NORMAL PARSING

```
  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX))
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

```
// missing {
```

ERROR

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

ERROR

NORMAL PARSING

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
  }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

ERROR

NORMAL PARSING

UNEXPECTED KEYWORD

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

New production:

WATER → MethodDec {cons("WATER"),recover}

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }
    return;
  }                                          WATER

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

New production:

```
WATER → Stm {cons("WATER"),recover}
```

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }                                    WATER
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

New production:

WATER → Stm {cons("WATER"),recover}

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX))
      fridge.startCooling();
    } PROBLEMATIC TOKEN
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

// missing {
LAYOUT

WATER

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX))
      fridge.startCooling();
    } PROBLEMATIC TOKEN
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

```
// missing {
```

LAYOUT

WATER

New production:

WATER → LAYOUT {cons("WATER"),recover}

Mixing Java with Water

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    } WATER
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

LAYOUT

New production:

WATER → LAYOUT {cons("WATER"),recover}

Reflection:
Water-based Recovery Rules

Works with *any* existing grammar

Can only *remove* fragments

Danger of Floods

```
public class Fridge {
  public void startCooling() {
    cooling.commence();
  // missing }

  public void onResetButtonPress() {
    log.message("Reset button pressed");
    power.reboot();
  }
}
```

Danger of Floods

```
public class Fridge {
  public void startCooling() {
    cooling.commence();
  // missing }

  public void onResetButtonPress() {
    log.message("Reset button pressed");
    power.reboot();
  }
}
```

WATER

So why not *parse* methods with missing closing brackets?

Productions for Missing Literals

why not add rules like this:

```
“if” “(” Expr      Stm → Stm {cons(“If”), recover}
```

and this, and this, and this:

```
“if”    Expr “)” Stm → Stm {cons(“If”), recover}
“if”    Expr     Stm → Stm {cons(“If”), recover}
“while” “(“ Expr Stm → Stm {cons(“While”), ...}
...
```

not going to scale.

Productions for Missing Literals

```
“if” “(” Expr “)” Stm → Stm {cons(“If”)}
```

what it means internally:

```
IF BOPEN … BCLOSE … → …
[\i][\f]                    → IF
[\(]                        → BOPEN
[\)]                        → BCLOSE
```

so, we can write (using the literal instead of BCLOSE):

```
                    → “)”   {recover}
                    → “}”   {recover}
```

Applying Insertion Rules

```
public class Fridge {
  public void startCooling() {
    cooling.commence();
  // missing }

  public void onResetButtonPress() {
    log.message("Reset button pressed");
    power.reboot();
  }
}
```

New production:

→ "}" {recover}

Applying Insertion Rules

```
public class Fridge {
  public void startCooling() {
    cooling.commence();
    // missing }                                INSERT }

  public void onResetButtonPress() {
    log.message("Reset button pressed");
    power.reboot();
  }
}
```

New production:

→ "}" {recover}

Recovery Rules

Water

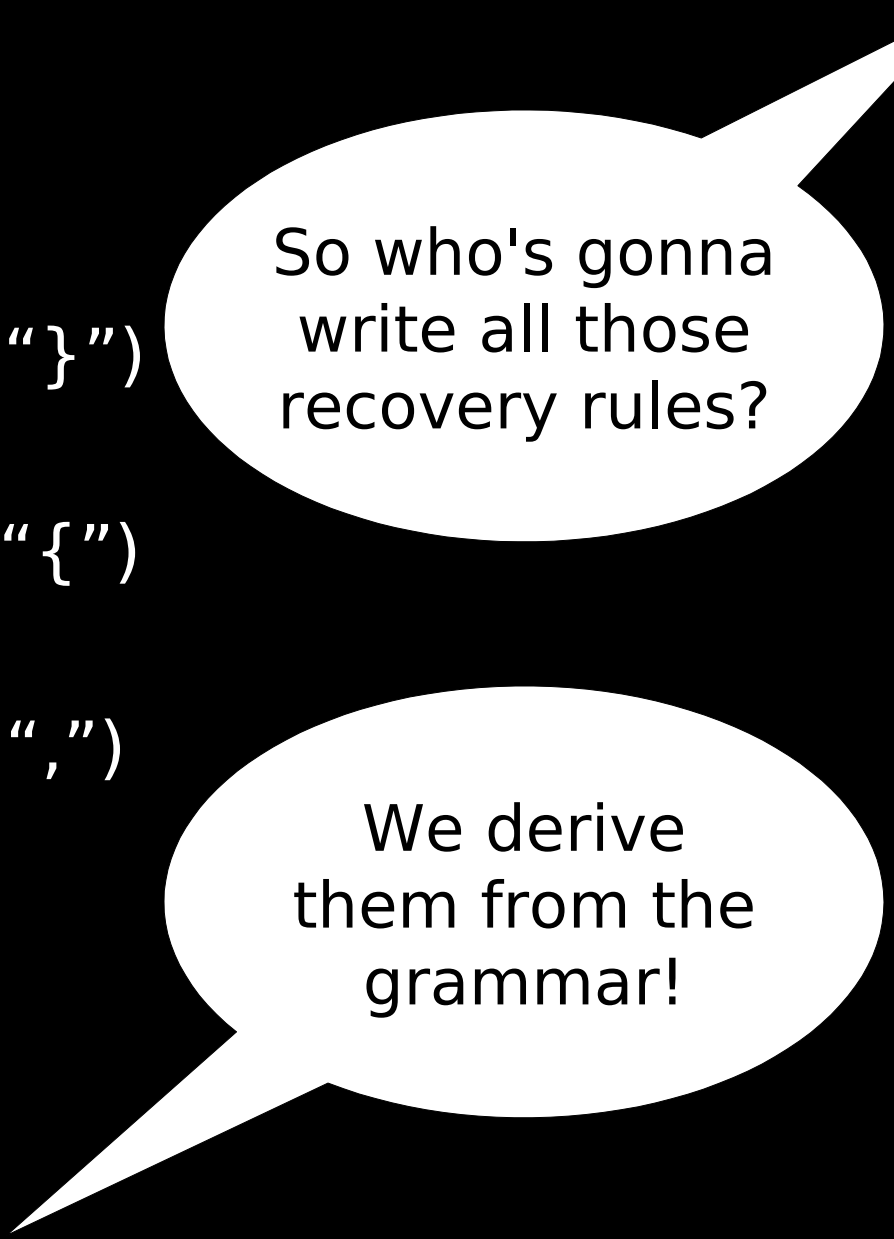
Closing brackets (“}”)

Opening brackets (“{”)

Separators (“,”)

Comments

String literals



Customization of Recovery Rules

```
      → “class” {reject}

“[|” → “|[“ {recover}

“|]” → “]|” {recover}
```

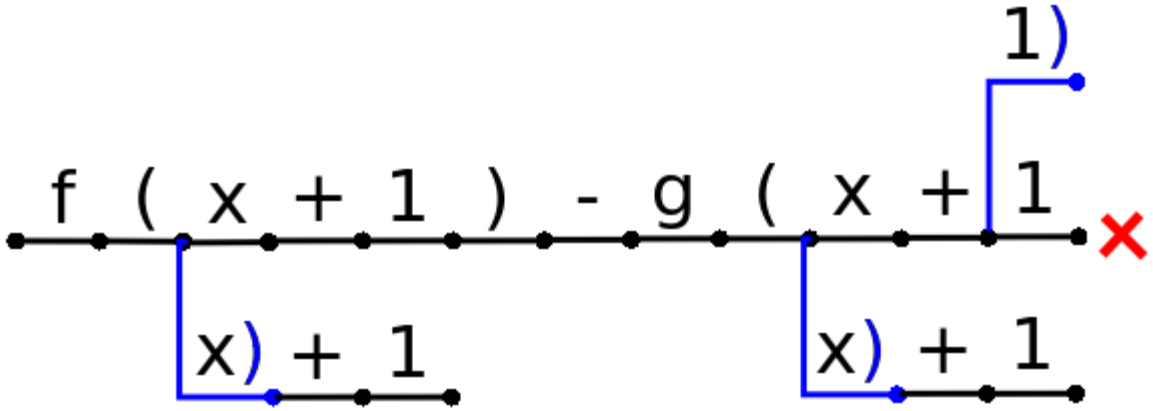
Putting Things Together

- Water
- Insertion: “)”

```
y =      x           ;
    f ( x    + 1 ) ;
    f (        1 ) ;
y     ( x    + 1 ) ;
y     ( x ) ;
y     ()    ;
y     (        1 ) ;
y =            1   ;
               ;
```

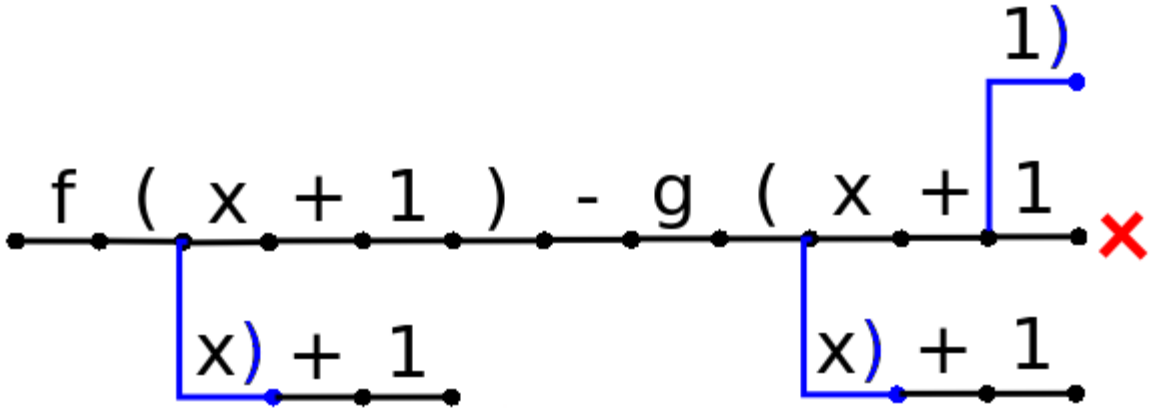
```
y = f ( x    + 1    ;
y = f ( x    + 1 ) ;
y = f ( x ) + 1    ;
y = f      + 1    ;
y = f (      1 ) ;
y =   ( x ) + 1    ;
y =   ( x    + 1 ) ;
y =      x   + 1    ;
y = f                ;
y =   ( x )          ;
y = f ( )            ;
```

Putting Things Together



For recovery, parallel parsing does not scale...

Putting Things Together



Why not do *backtracking* for recovery?

Recovery Using Backtracking

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

ERROR

Recovery Using Backtracking

```
public class TemperatureMonitor {
  private Fridge fridge;
```

1. NORMAL PARSING

```
  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX))
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

```
// missing {
```

ERROR

Recovery Using Backtracking

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();                    ERROR
    }                                  1. NORMAL PARSING
    return;                          POINT OF DETECTION
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

Recovery Using Backtracking

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) // missing {
      fridge.startCooling();
    }                                    1. NORMAL PARSING
    return;                              2. BACKTRACKING
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

Recovery Using Backtracking

```
public class TemperatureMonitor {
  private Fridge fridge;

  public void trigger(Temperature temp) {
    if (temp.greaterThan(MAX)) {
      fridge.startCooling();
    }
    return;
  }

  public TemperatureMonitor(Fridge fridge) {
    this.fridge = fridge;
  }
}
```

1 RULE
2 RULES
3 RULES
4 RULES

Evaluation

Implementation:

- JSGLR
- Spoofax/IMP (WebDSL, Stratego, PIL, ...)

Test grammars:

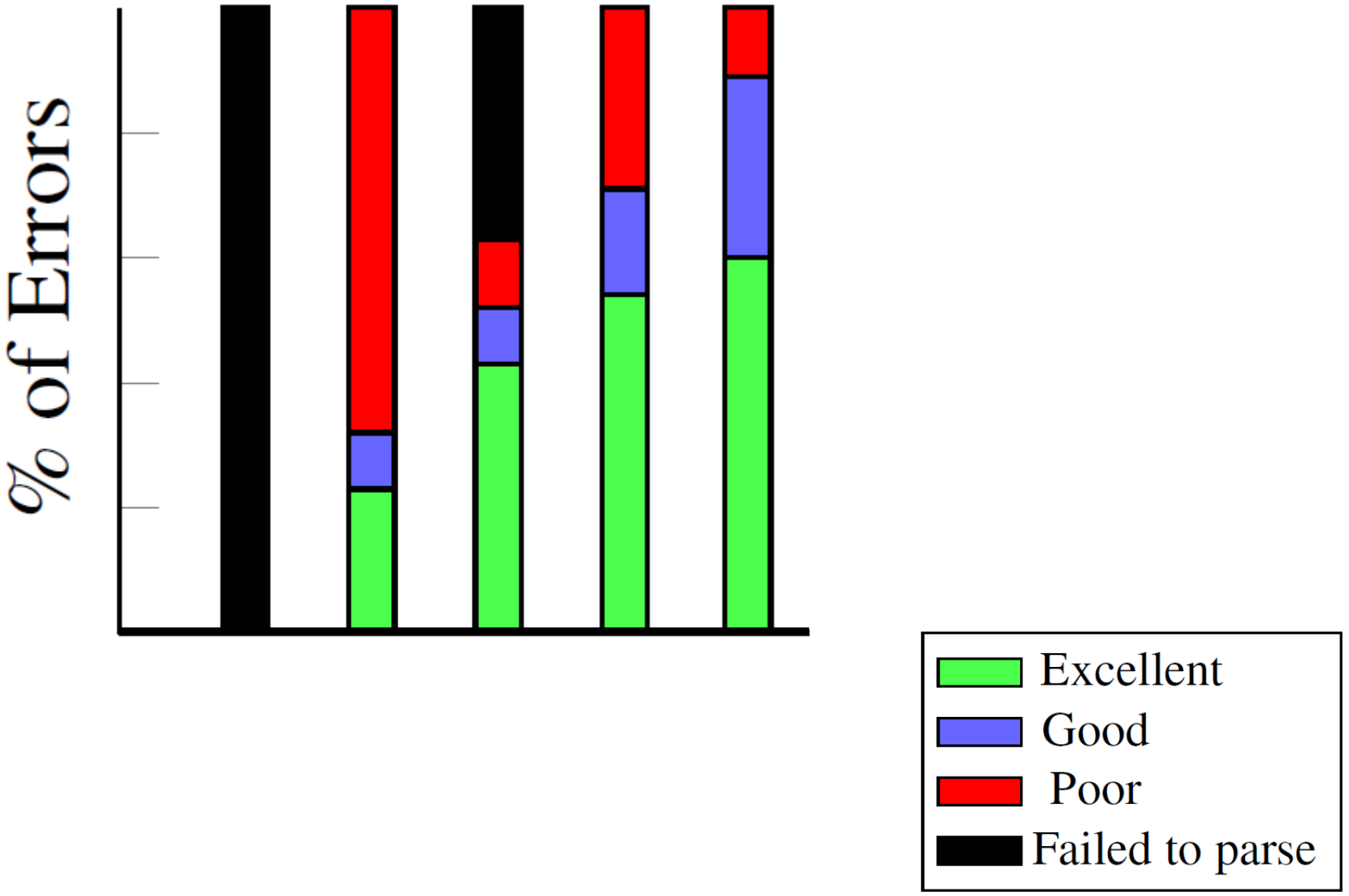
- Java
- Java-SQL
- Stratego
- Stratego-Java

Evaluation

Measurements:

- qualitative [Penello & DeRemer '78]:
  excellent / good / poor
- tree alignment distance [Jian et al '94]
- textual 'diffs'

Recovery Quality (Stratego-Java)



Continued Work

"Natural and Flexible Error Recovery for Generated Parsers" [SLE 2009]

- indentation-based region selection
- fall-back mechanism
- bridge parsing [Nilsson-Nyman et al, 2008]

(www.strategoxt.org/Stratego/PermissiveGrammars)

Continued Work

"Natural and Flexible Error Recovery for Generated Parsers" [SLE 2009]

- comparison with JDT
- performance, quality
- synergy between recovery techniques

(www.strategoxt.org/Stratego/PermissiveGrammars)

Conclusion

SGLR/SDF

- modular specifications
- composable languages

Recovery production rules

- transparent
- flexible
- language-independent

www.strategoxt.org/Stratego/PermissiveGrammars
www.strategoxt.org/Stratego/Spoofax-IMP
www.lennartkats.net