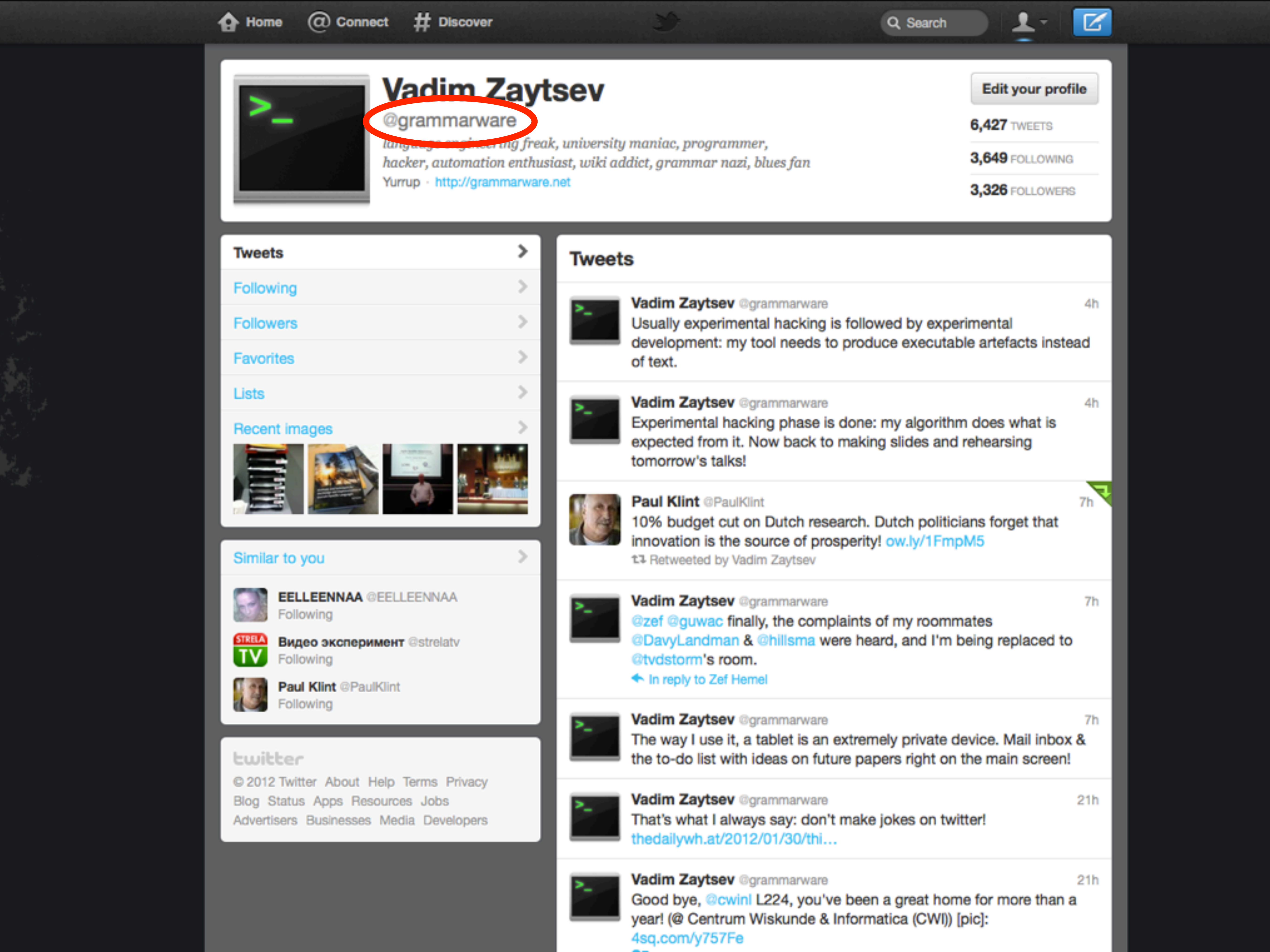
The Life Cycle of Grammarware

CWI Scientific Meeting

Vadim Zaytsev, SWAT, CWI

2012

Grammarware



Vadim Zaytsev

@grammarware

language engineering freak, university maniac, programmer, hacker, automation enthusiast, wiki addict, grammar nazi, blues fan

Yurrup · http://grammarware.net

6,427 TWEETS

3,649 FOLLOWING

3,326 FOLLOWERS

Similar to you

EELLEENNAA @EELLEENNAA
Following

Видео экспермент @strelatv
Following

Paul Klint @PaulKlint
Following

© 2012 Twitter About Help Terms Privacy
Blog Status Apps Resources Jobs
Advertisers Businesses Media Developers

Tweets

Vadim Zaytsev @grammarware 4h
Usually experimental hacking is followed by experimental development: my tool needs to produce executable artefacts instead of text.

Vadim Zaytsev @grammarware 4h
Experimental hacking phase is done: my algorithm does what is expected from it. Now back to making slides and rehearsing tomorrow's talks!

Paul Klint @PaulKlint 7h
10% budget cut on Dutch research. Dutch politicians forget that innovation is the source of prosperity! ow.ly/1FmpM5
Retweeted by Vadim Zaytsev

Vadim Zaytsev @grammarware 7h
@zef @guwac finally, the complaints of my roommates @DavyLandman & @hillsma were heard, and I'm being replaced to @tvdstorm's room.
In reply to Zef Hemel

Vadim Zaytsev @grammarware 7h
The way I use it, a tablet is an extremely private device. Mail inbox & the to-do list with ideas on future papers right on the main screen!

Vadim Zaytsev @grammarware 21h
That's what I always say: don't make jokes on twitter! thedailywh.at/2012/01/30/thi…

Vadim Zaytsev @grammarware 21h
Good bye, @cwinl L224, you've been a great home for more than a year! (@ Centrum Wiskunde & Informatica (CWI)) [pic]: 4sq.com/y757Fe

Software Languages

Language: make

```
all:
        make clean
        make build
        make test

test:
        ./converge.py master.bgf  base/

build:
        cp ../../convergence/fl/snapshot/*.bgf  tests/
        rsc2bgf ../../fl/rascal1/FL.rsc tests/rascal.bgf
        ls -1 tests/*.bgf | xargs -n1  ./testperform
        cp normal/*.normal.bgf  base/

clean:
        rm -f tests/*.bgf  xbgf/*.xbgf  normal/* base/*
```

Language: Java

```
import types.*;
import org.antlr.runtime.*;
import java.io.*;
public class TestEvaluator {
    public static void main(String[] args) throws Exception {
        ANTLRFileStream input = new ANTLRFileStream(args[0]);
        FLLexer lexer = new FLLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        FLParser parser = new FLParser(tokens);
        Program program = parser.program();
        input = new ANTLRFileStream(args[1]);
        lexer = new FLLexer(input);
        tokens = new CommonTokenStream(lexer);
        parser = new FLParser(tokens);
        Expr expr = parser.expr();
         Evaluator eval = new Evaluator(program);
         int expected = Integer.parseInt(args[2]);
         assert expected == eval.evaluate(expr);
}
```

Language: XML (BGF)

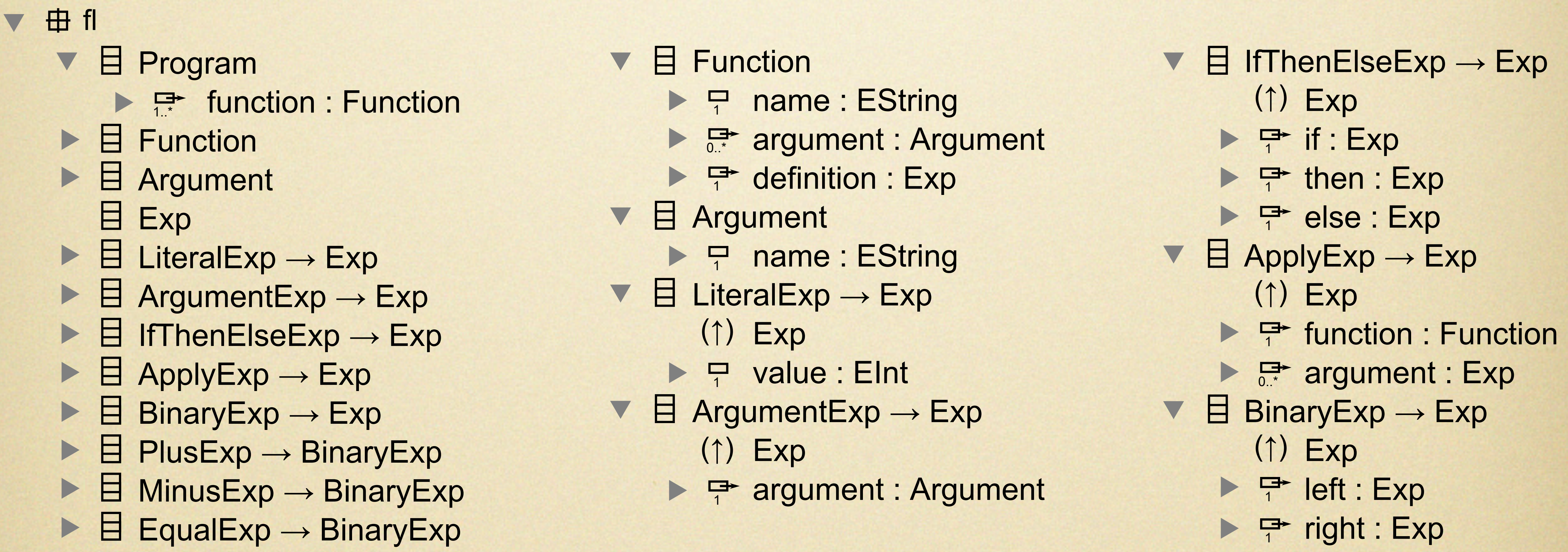
```xml
<?xml version="1.0" encoding="UTF-8"?>
<bgf:grammar xmlns:bgf="http://planet-sl.org/bgf">
    <root>Program</root>
    <root>Fragment</root>
    <bgf:production>
        <nonterminal>Program</nonterminal>
        <bgf:expression>
            <plus>
                <bgf:expression>
                    <selectable>
                        <selector>function</selector>
                        <bgf:expression>
                            <nonterminal>Function</nonterminal>
                        </bgf:expression>
                    </selectable>
                </bgf:expression>
            </plus>
        </bgf:expression>
    </bgf:production>
    <!-- ... -->
</bgf:grammar>
```

Language: SDF

```
context-free syntax
    Function+                          -> Program
    Name Name+ "=" Expr Newline+       -> Function
    Expr Ops Expr                      -> Expr {left,prefer,cons(binary)}
    Name Expr+                         -> Expr {avoid,cons(apply)}
    "if" Expr "then" Expr "else" Expr  -> Expr {cons(ifThenElse)}
    "(" Expr ")"                       -> Expr {bracket}
    Name                               -> Expr {cons(argument)}
    Int                                -> Expr {cons(literal)}

    "-"                                -> Ops {cons(minus)}
    "+"                                -> Ops {cons(plus)}
    "=="                               -> Ops {cons(equal)}
```

Language: Ecore

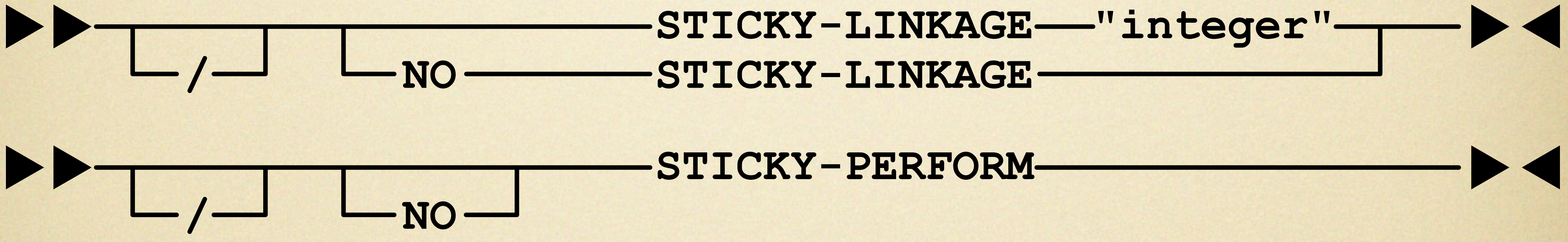


```
▼ fl
    ▼ Program
        ▶ 1..* function : Function
    ▶ Function
    ▶ Argument
      Exp
    ▶ LiteralExp → Exp
    ▶ ArgumentExp → Exp
    ▶ IfThenElseExp → Exp
    ▶ ApplyExp → Exp
    ▶ BinaryExp → Exp
    ▶ PlusExp → BinaryExp
    ▶ MinusExp → BinaryExp
    ▶ EqualExp → BinaryExp
```

```
▼ Function
    ▶ 1 name : EString
    ▶ 0..* argument : Argument
    ▶ 1 definition : Exp
▼ Argument
    ▶ 1 name : EString
▼ LiteralExp → Exp
      (↑) Exp
    ▶ 1 value : EInt
▼ ArgumentExp → Exp
      (↑) Exp
    ▶ 1 argument : Argument
```

```
▼ IfThenElseExp → Exp
      (↑) Exp
    ▶ 1 if : Exp
    ▶ 1 then : Exp
    ▶ 1 else : Exp
▼ ApplyExp → Exp
      (↑) Exp
    ▶ 1 function : Function
    ▶ 0..* argument : Exp
▼ BinaryExp → Exp
      (↑) Exp
    ▶ 1 left : Exp
    ▶ 1 right : Exp
```

Language: syntax diagram



/

NO

STICKY-LINKAGE "integer"

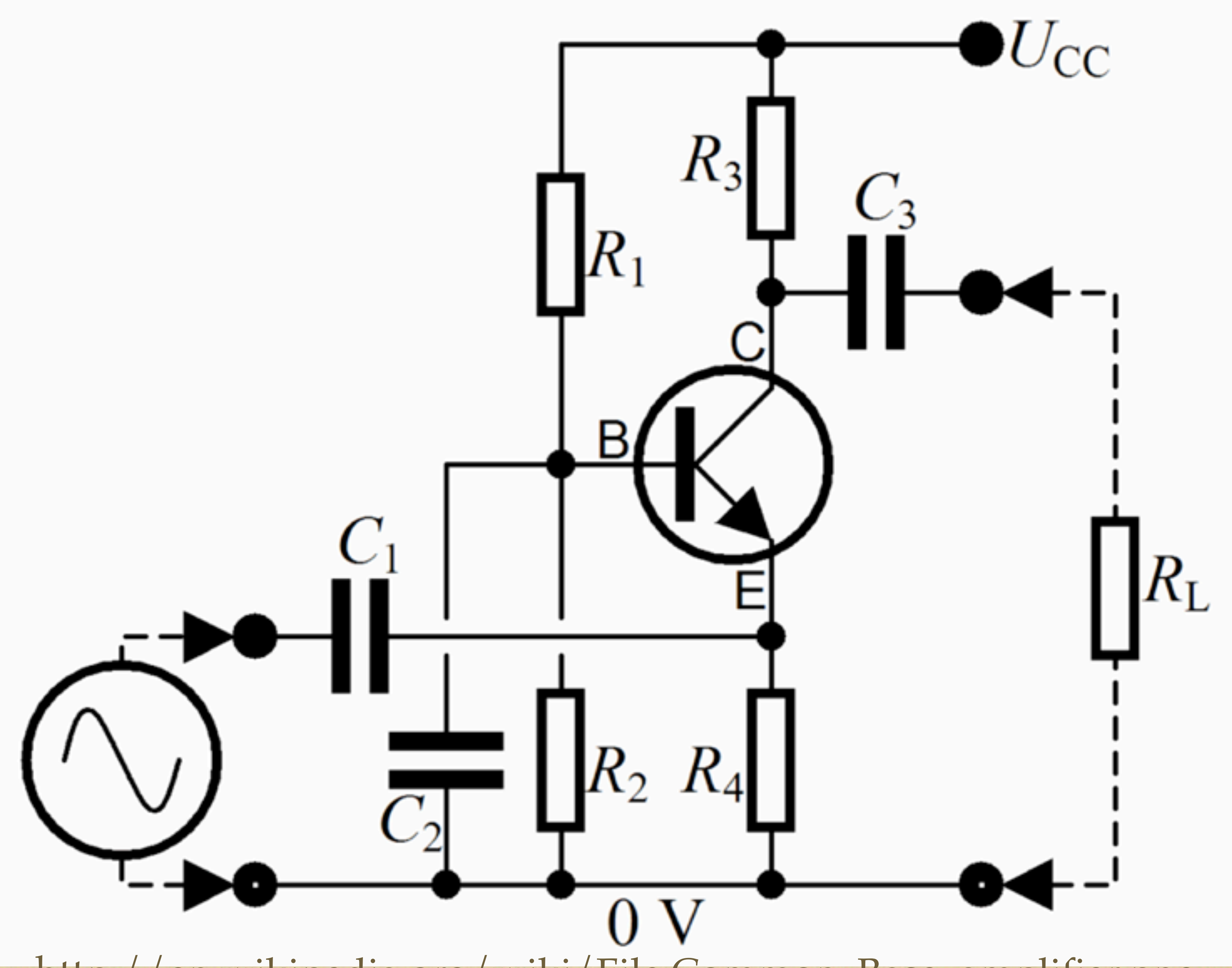
STICKY-LINKAGE

/

NO

STICKY-PERFORM

Also a language



http://en.wikipedia.org/wiki/File:Common_Base_amplifier.png

Formal Languages

Known from theory (1/2)

- What is a language?
  - set of allowed words
  - often requires set comprehension to define
  - infinite for realistic cases
- How to document a software language?
- How to express language evolution?

Known from theory (2/2)

- What is a grammar?
  - a set of nonterminals
  - a set of terminals
  - a set of production rules
  - a start symbol
- What is a good grammar?
- How to combine or decompose grammars?

Software languages

- General purpose programming languages
- Domain specific languages
- Modelling and metamodelling languages
- Data description languages, data models, schemata
- Ontologies
- APIs and libraries

Grammarware

- Parser
- Compiler
- Interpreter
- Pretty-printer
- Scanner
- Browser
- Static checker
- Structural editor
- IDE
- DSL framework
- Preprocessor
- Postprocessor
- Model checker
- Refactorer
- Code slicer
- API
- XMLware
- Modelware
- Language workbench
- Reverse engineering tool
- Benchmark
- Recommender
- Renovation tool

Life Cycle

★★★★★★

Grammar recovery

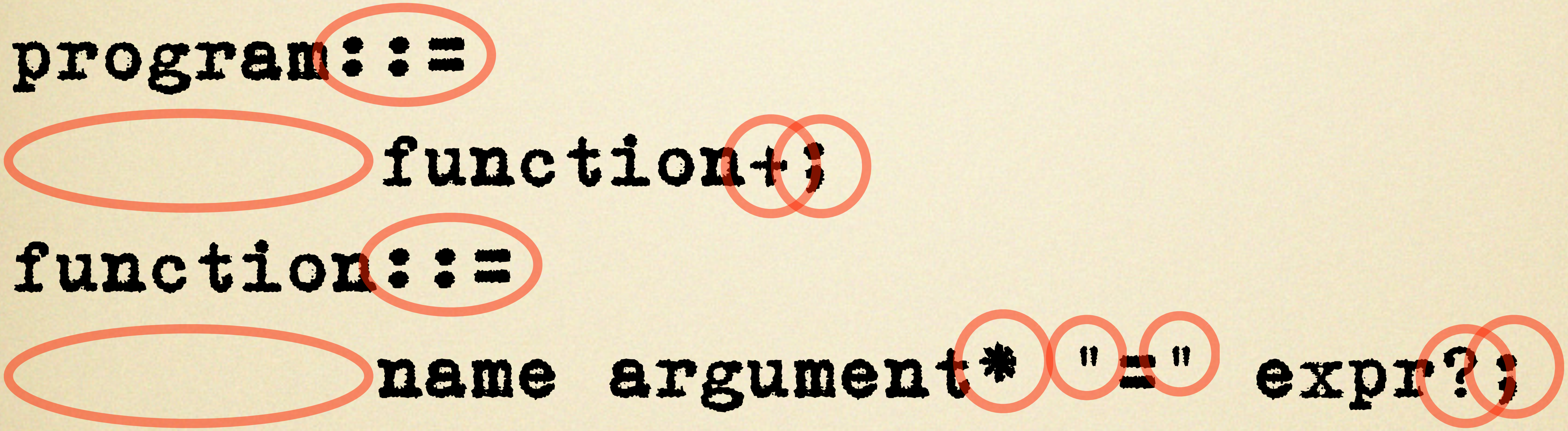
- Given is an artefact containing grammar knowledge:
  - a grammar
  - a parser specification
  - a metamodel
  - grammarware source code
  - a data schema
  - documentation
- Question: how to extract a grammar from it?
- Answer: with tolerant grammar recovery techniques!

Extraction of Java grammars

	 impl1	 impl2	 impl3	 read1	 read2	 read3	Total
Arbitrary lexical decisions	2	109	60	1	90	161	423
Well-formedness violations	5	0	7	4	11	4	31
Indentation violations	1	2	7	1	4	8	23
Recovery rules	3	12	18	2	59	47	141
∘ Match parentheses	0	3	6	0	0	0	9
∘ Metasymbol to terminal	0	1	7	0	27	7	42
∘ Merge adjacent symbols	1	0	0	1	1	0	3
∘ Split compound symbol	0	1	1	0	3	8	13
∘ Nonterminal to terminal	0	7	3	0	8	11	29
∘ Terminal to nonterminal	1	0	1	1	17	13	33
∘ Recover optionality	1	0	0	0	3	8	12
Purge duplicate definitions	0	0	0	16	17	18	51
Total	11	123	92	24	181	238	669

Lämmel, Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SQJ 19:2, March 2011.

Notation-parametric recovery



```
program::=
                function+;
function::=
                name argument* "=" expr?;
```

- Compose a notation specification
- Perform a robust heuristic-based recovery process
- Successful for grammars of Ada, C, C++, C♯, Dart, Eiffel, Modula, MediaWiki, LLL, EBNF, etc.

Zaytsev, *What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*, SAC/PL 2012.
Zaytsev, *Notation-Parametric Grammar Recovery*, LDTA 2012, April 2012.

Language evolution

- Given is a (baseline) grammar
- The grammar needs to be engaged in:
  - correction
  - evolution
  - adaptation
  - beautification
- Question: how to apply stable, disciplined, reproducible, automated, possibly bidirectional transformations?
- By programmable grammar transformations!

XBGF transformation suite

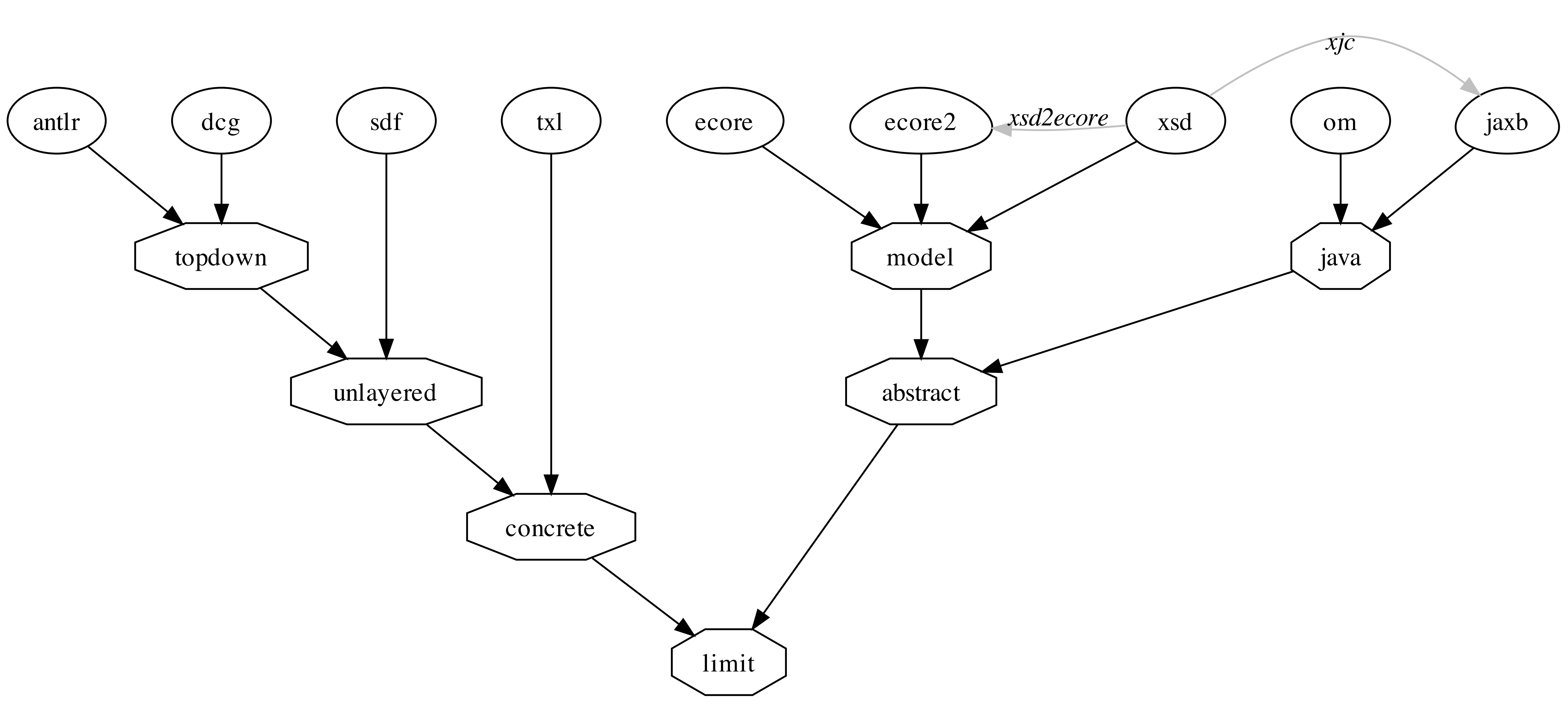
- Semantics-preserving operators:
    - fold, unfold, rename, factor, massage, ...
- Semantics-increasing/decreasing operators:
    - appear/disappear, narrow/widen, add/remove, ...
- Semantics-revising operators
    - inject, permute, replace, redefine, ...

Zaytsev, *BGF transformation operator suite v.1.0*, July 2010.

Language convergence

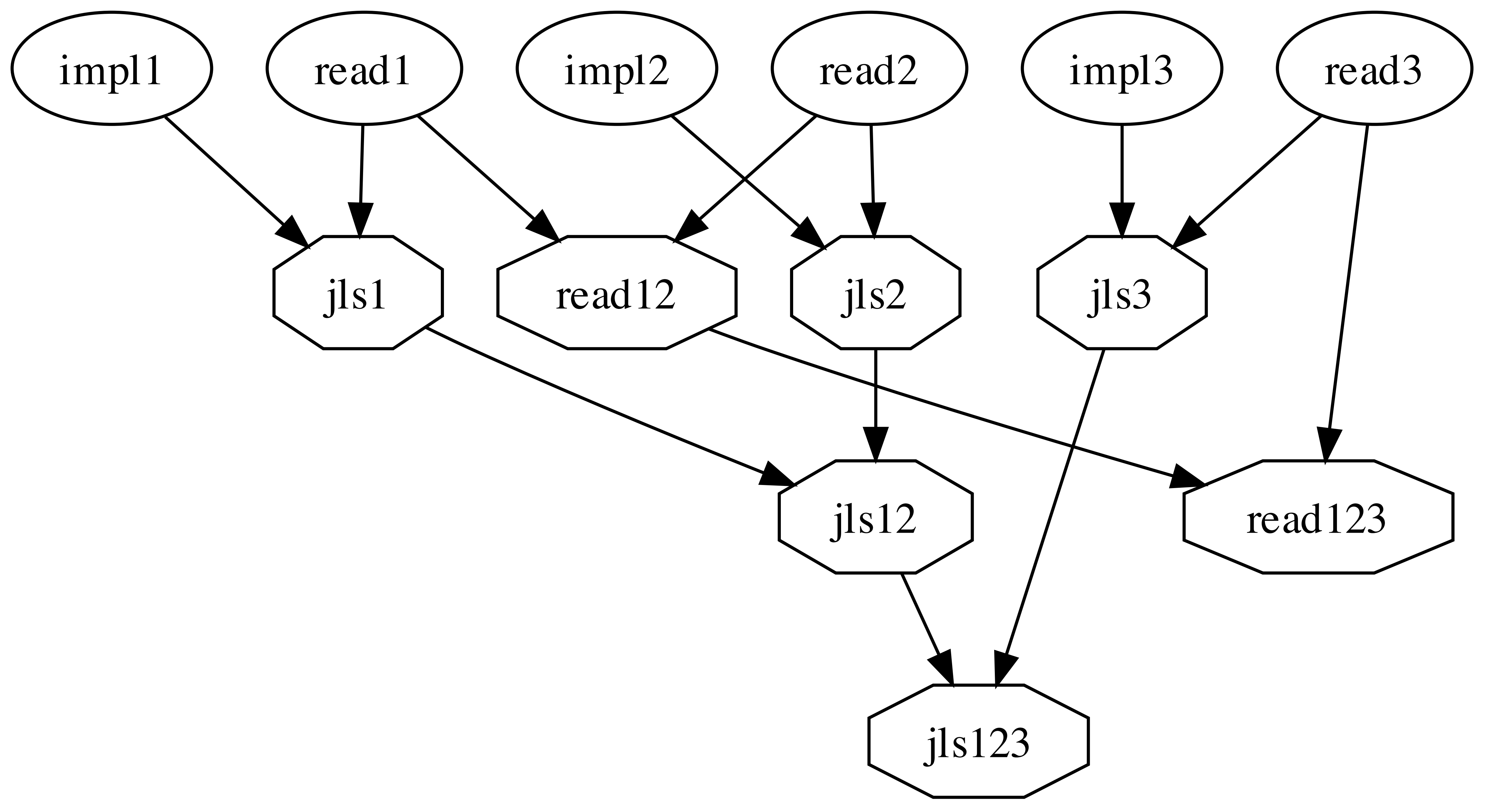
- Given are several grammars of related languages
- We want to investigate their relationships:
  - equivalent
  - dialects
  - backward compatible versions
- Question: how to reverse engineer grammar relationships?
- With grammar convergence!

Convergence tree



Lämmel, Zaytsev, *An Introduction to Grammar Convergence*, IFM 2009, LNCS 5423, February 2009.

Convergence graph

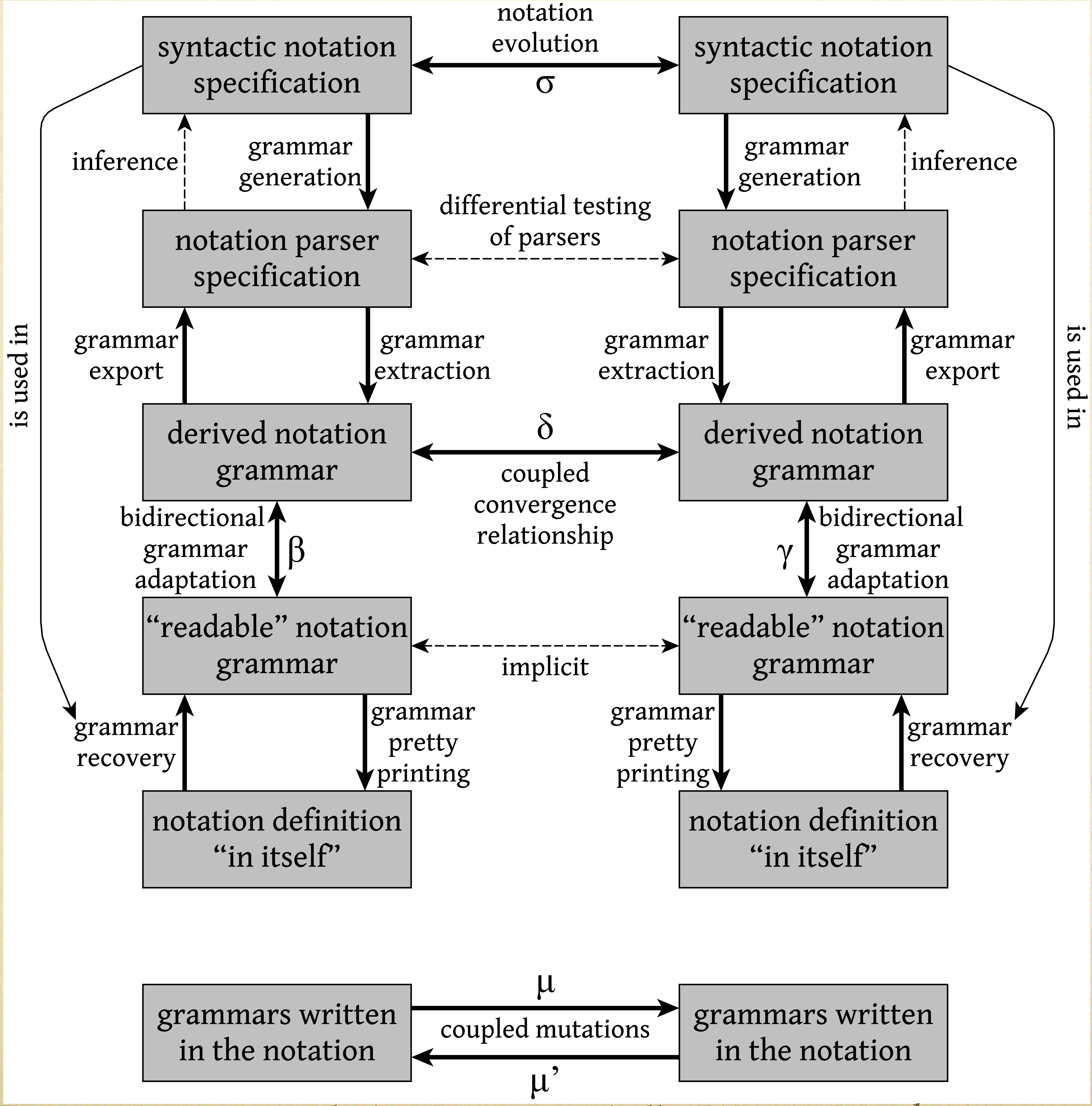


Lämmel, Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SCAM 2009, September 2009.

Metalanguage evolution

- Given is a grammar written in a specific notation
- This notation can be different:
  - lexically
  - semantically
  - expressively
- Question: how to migrate grammars to another notation?
- By coupled evolution of syntax and metasyntax!

Coupled evolution megamodel

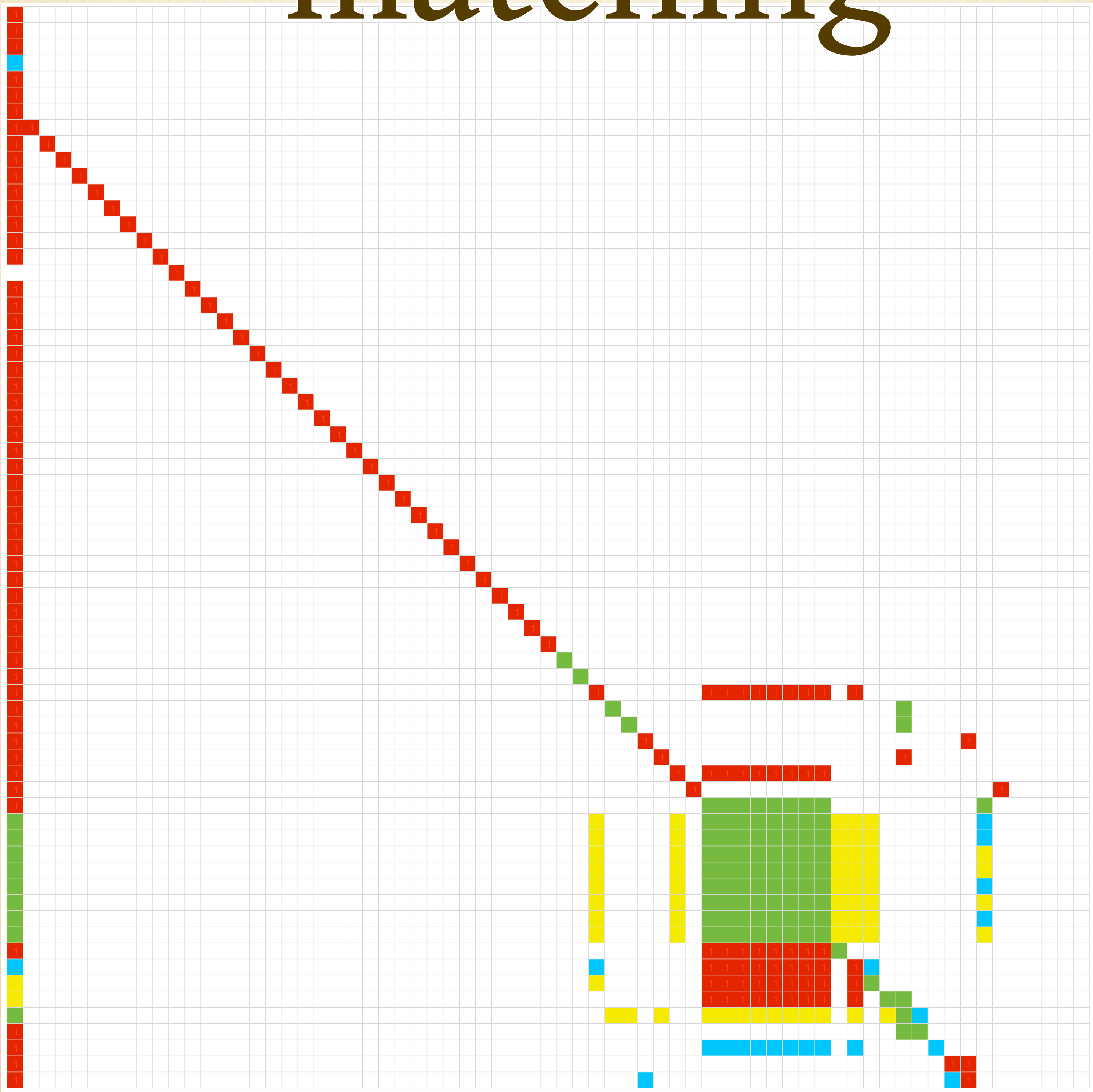

syntactic notation specification
notation evolution
σ
syntactic notation specification
inference
grammar generation
grammar generation
inference
differential testing of parsers
notation parser specification
notation parser specification
is used in
grammar export
grammar extraction
grammar extraction
grammar export
is used in
derived notation grammar
δ
coupled convergence relationship
derived notation grammar
bidirectional grammar adaptation
β
γ
bidirectional grammar adaptation
"readable" notation grammar
implicit
"readable" notation grammar
grammar recovery
grammar pretty printing
grammar pretty printing
grammar recovery
notation definition "in itself"
notation definition "in itself"
grammars written in the notation
μ
coupled mutations
μ'
grammars written in the notation

Zaytsev, *Language Evolution, Metasyntactically*, BX 2012, March 2012.

Matching metamodel entities

- Given are two grammars that model the “same” language
- Grammars have entities (e.g., nonterminals) that have:
  - different names
  - different structure
- Question: how to match such entities?
- With grammar-based combinatorial differential testing!

Visualised nonterminal matching

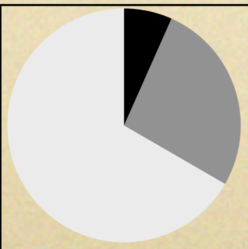
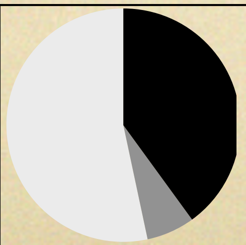
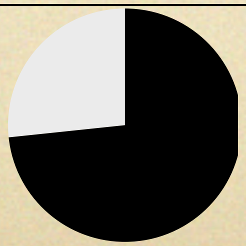
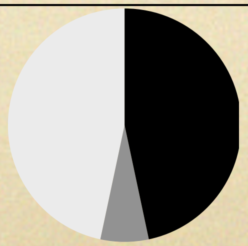
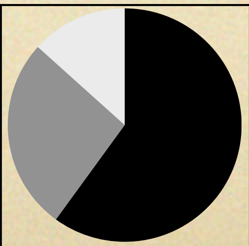
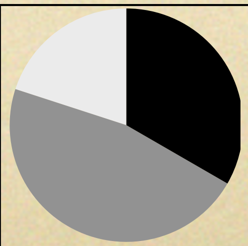
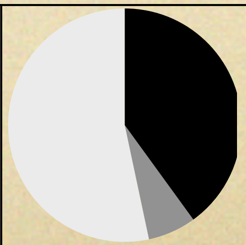
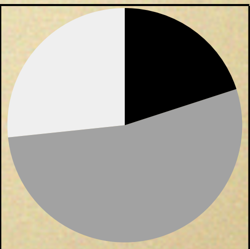


Fischer, Lämmel, Zaytsev, Comparison of Context-free Grammars Based on Parsing Generated Test Data, SLE 2011, LNCS 6940, 2012.

Language documentation

- Given are:
  - a grammar for a software language
  - explanations in a natural language
  - executable code samples
  - known relationships between concepts
- Question: how to do language documentation properly?
- Answer: by generating it from structured data!

Unified model for language docs

Domain concept	IAL [Bac60]	Jovial [MIL84]	Design Patterns [GHJV95]	Smalltalk [Sha97]	Informix [IBM03]	C# [Sta06]	MOF [MOF06]	XPath [BBC$^+$07]
synopsis	—	∼	intent	synopsis	∼	∼	∼	—
description	∼	—	motivation	definition	usage	∼	—	∼
syntax	—[a]	syntax	structure	∼	∼	∼	—	[NN][b]
constraints	—	constraints	applicability	errors	restrictions	∼	constraints	∼
references	—	—	related patterns	—	references	∼	—	∼
relationship	—	—	consequences	return value, refinement	related	return type	—	∼
semantics	—	semantics	collaborations	—	important	∼	semantics	∼
rationale	∼	notes	implementation	rationale	GLS, ES[c]	note	rationale	note
example	examples	examples	sample code, known uses	—	∼	example	—	∼
update	—	—	—	—	—	—[d]	changes	—
default	—	—	—	—	note	default values	—	—
value	—	—	also known as	conforms to	—	—	—	—
list	∼	—	—	messages, parameters	*terminals*	—	properties	∼
section	∼	—	—	—	∼	∼	—	∼
subtopic	—	types	participants	—	fields	parameters, methods	operations	functions
Coverage of LDF								

Zaytsev, Lämmel, *A Unified Format for Language Documents*, SLE 2010, LNCS 6563, January 2011.

Bibliography

- Ralf Lämmel, Vadim Zaytsev, *An Introduction to Grammar Convergence*, IFM 2009, LNCS 5423, February 2009.

- Ralf Lämmel, Vadim Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SCAM 2009, September 2009.

- Vadim Zaytsev, *BGF Transformation Operator Suite v.1.0*, July 2010.

- Vadim Zaytsev, Ralf Lämmel, *A Unified Format for Language Documents*, SLE 2010, LNCS 6563, January 2011.

- Ralf Lämmel, Vadim Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SQJ 19:2, March 2011.

- Vadim Zaytsev, *What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*, SAC/PL 2012, March 2012.

- Vadim Zaytsev, *Notation-Parametric Grammar Recovery*, LDTA 2012, April 2012.

- Bernd Fischer, Ralf Lämmel, Vadim Zaytsev, *Comparison of Context-free Grammars Based on Parsing Generated Test Data*, SLE 2011, LNCS 6940, to appear in 2012.

- Vadim Zaytsev, *Language Evolution, Metasyntactically*, BX 2012, pending revision in 2012.

Discussion