SWAT

Maintenance and Evolution of Grammarware by Grammar Transformation

IPA Spring Days on Model-Driven Software Engineering

Vadim Zaytsev, SWAT, CWI

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

Tweets
Following
Followers
Favorites
Lists
Recent images

Similar to you

EELLEENNAA @EELLEENNAA
Following

Видео эксперимент @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

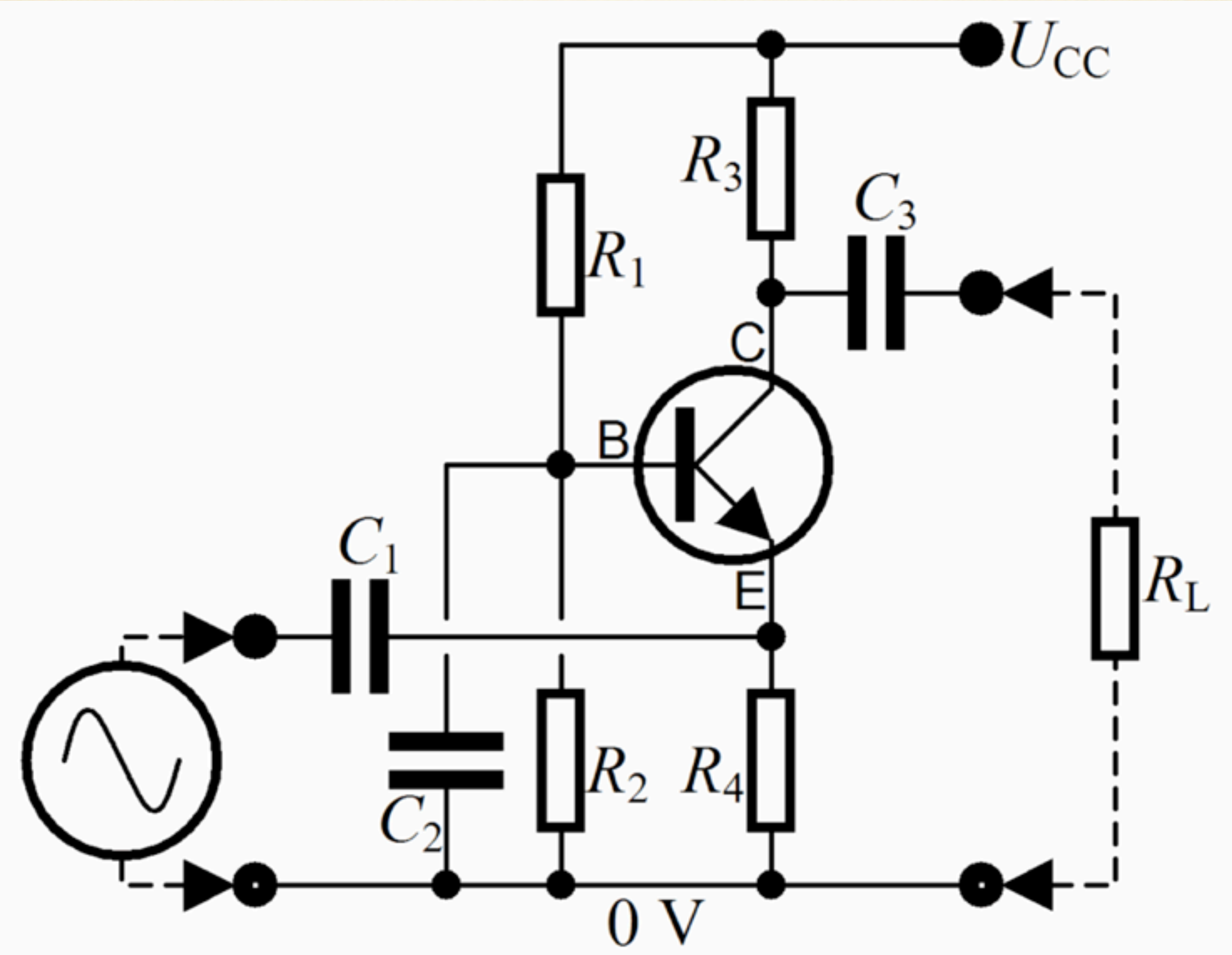
Language: Java

```
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: electric circuit



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

From languages to grammars

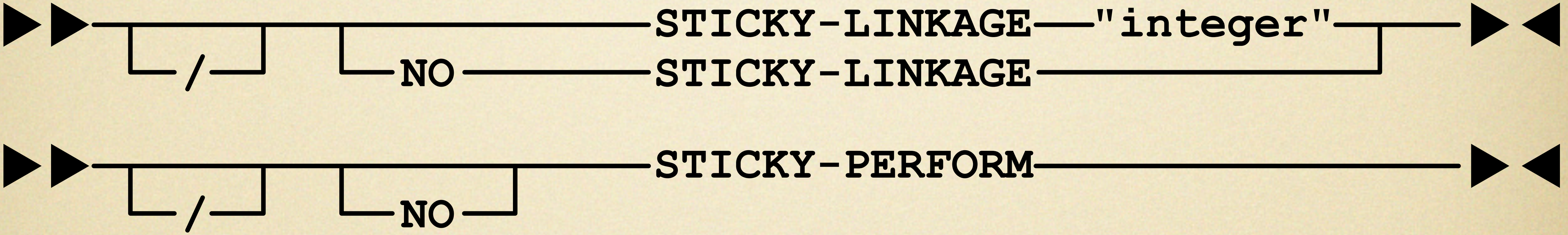
- Grammar
  - finite formal definition of a language
  - defines the structure of allowed language instances
- Classical definition
  - nonterminals, terminals, production rules
  - statement ::= “if” expression “then” statement
- Grammarware
  - grammar-based software

Grammar example (EBNF)

```
compilationUnit ::=
      topLevelDefinition* EOF
topLevelDefinition ::=
      classDefinition
      interfaceDefinition
      functionTypeAlias
      functionSignature functionBody
      returnType? getOrSet identifier formalParameterList functionBody
      “final” type? staticFinalDeclarationList “;”
      variableDeclaration “;”
classDefinition ::=
      “class” identifier typeParameters? superclass? interfaces? “{“ memberDef* “}”
typeParameters ::=
      “<” typeParameter (“,” typeParameter)* “>”
superclass ::=
      “extends” type
interfaces ::=
      “implements” typeList
```

Google Dart v0.05: http://slps.sf.net/zoo/dart/spec-0.05.html (Grammar Zoo)

“Grammar” (syntax diagram)



/ NO STICKY-LINKAGE "integer"

/ NO STICKY-PERFORM

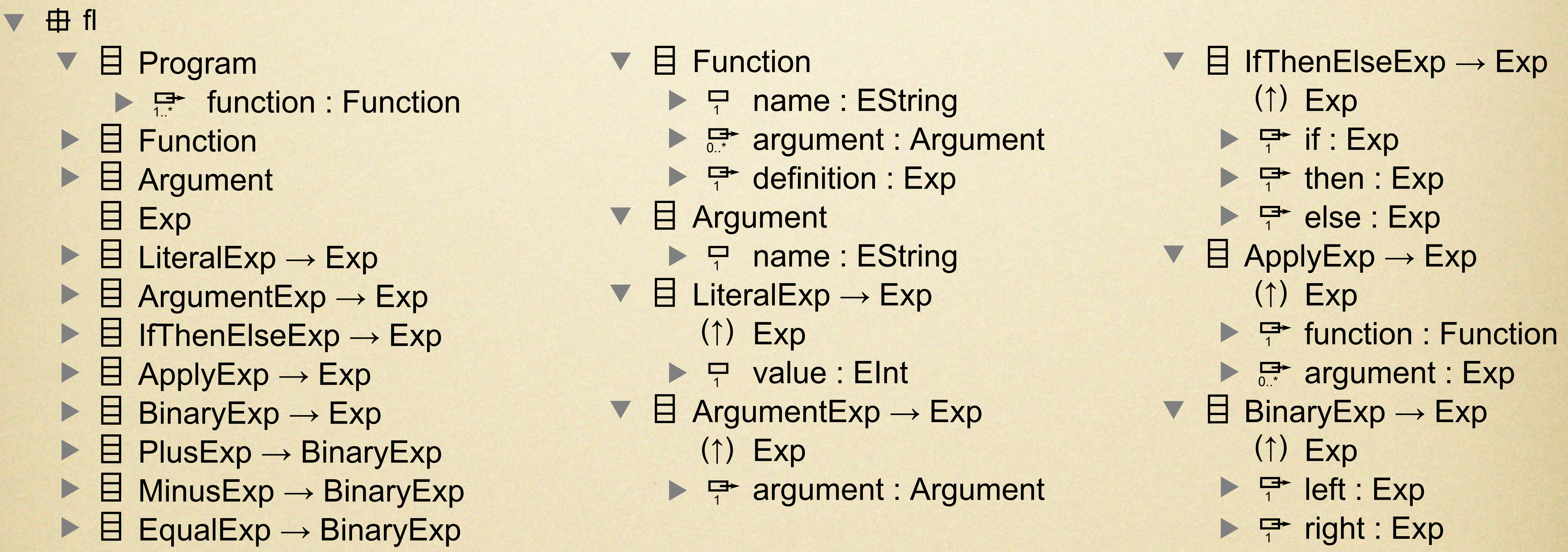
Micro Focus COBOL for UNIX Pocket Guide, Issue 5, 1994, page 3–87.

“Grammar” (parser spec)

```
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)}
```

“Grammar” (metamodel)

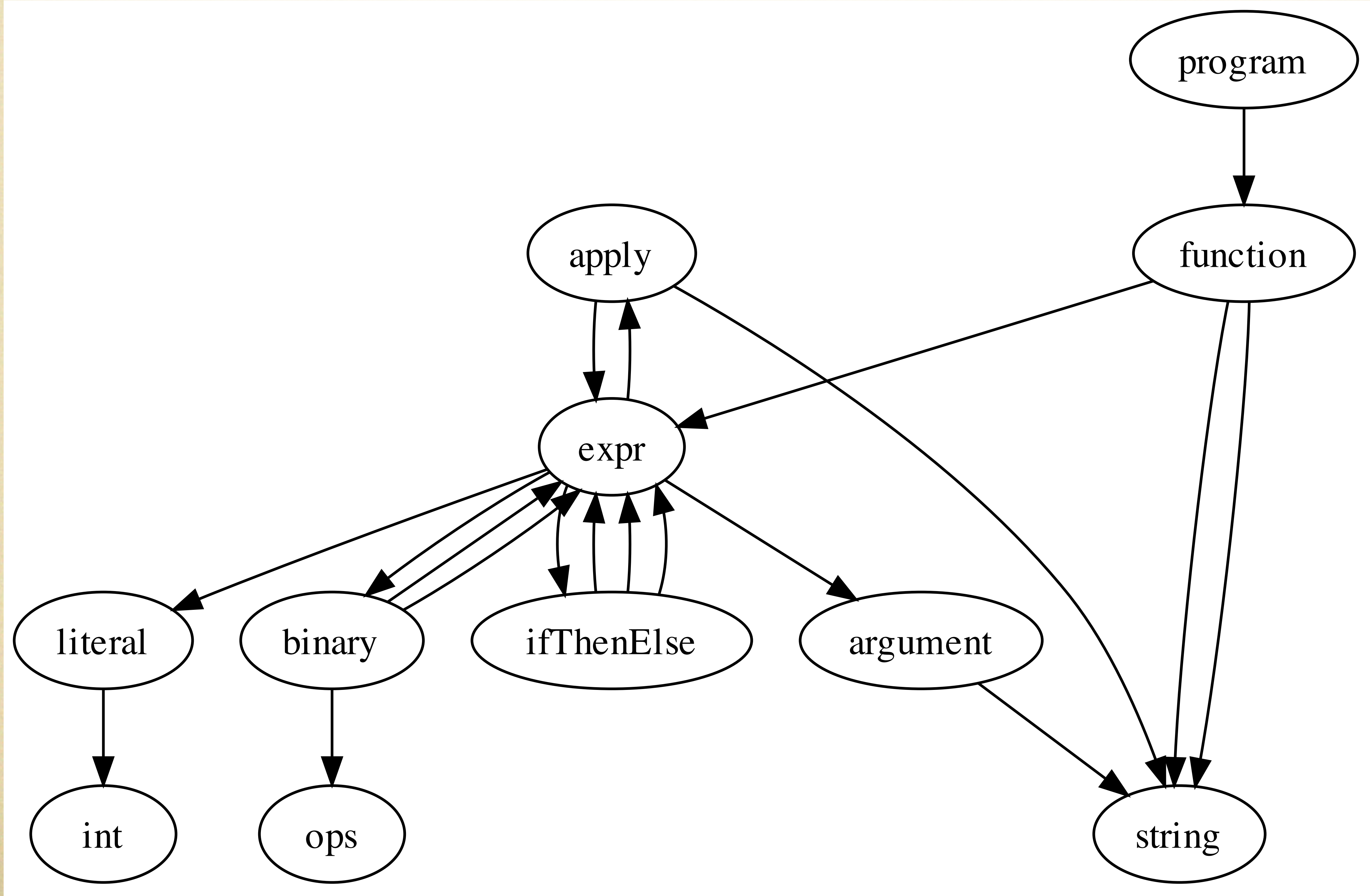


```
▼ fl
    ▼ Program
        ▶ 1..* function : Function
    ▶ Function
    ▶ Argument
      Exp
    ▶ LiteralExp → Exp
    ▶ ArgumentExp → Exp
    ▶ IfThenElseExp → Exp
    ▶ ApplyExp → Exp
    ▶ BinaryExp → Exp
    ▶ PlusExp → BinaryExp
    ▶ MinusExp → BinaryExp
    ▶ EqualExp → BinaryExp

▼ 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
```

“Grammar” (relation diagram)



Grammarware examples

- 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

Grammar Transformations

Motivation

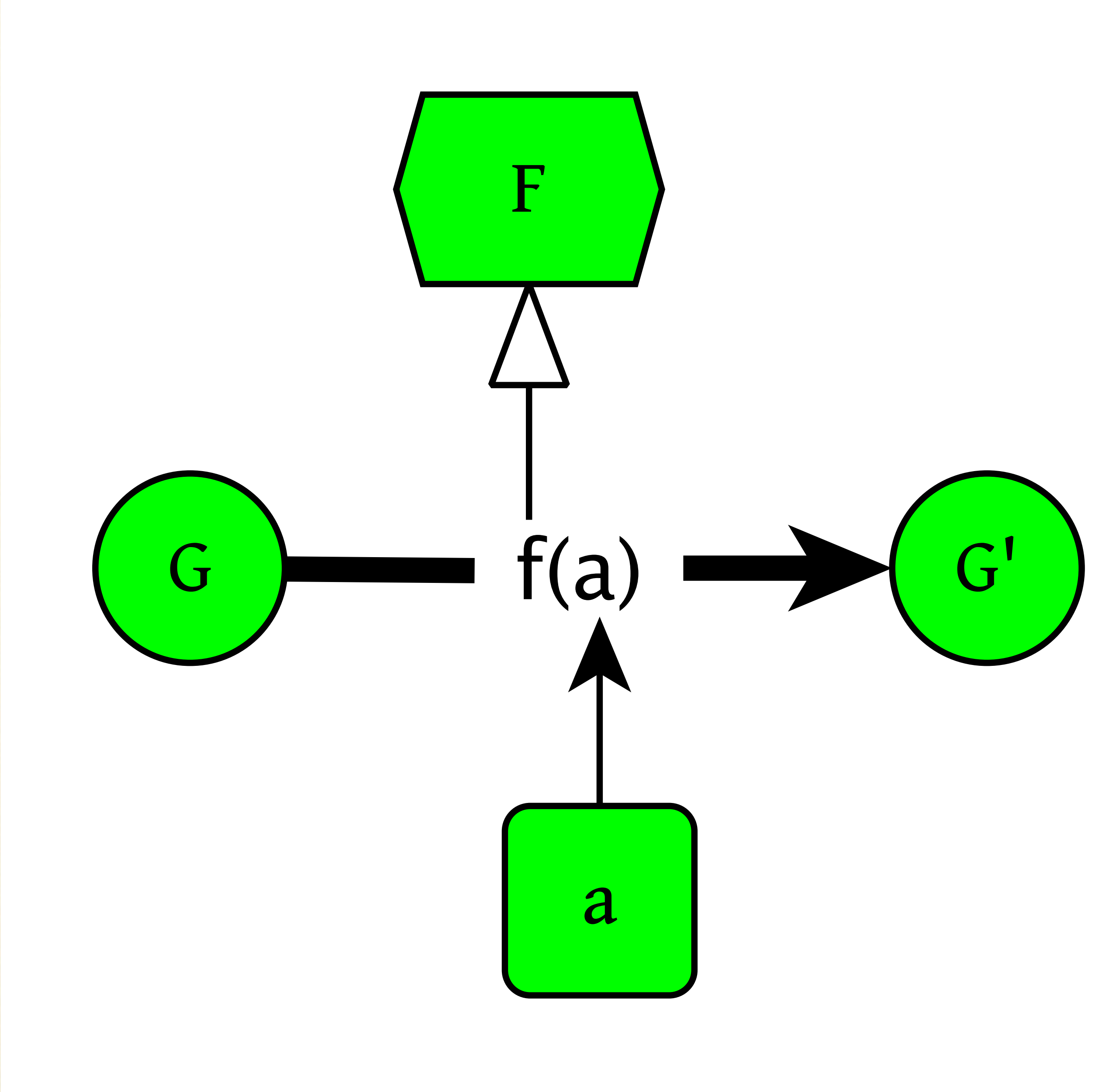
- Why transform?
  - Grammar adaptation
  - Grammar beautification
  - Inconsistency management
  - Version control
- Documented, well-understood, compositional change
- Any difference can be a transformation
- Good for representing relationships

Transformations

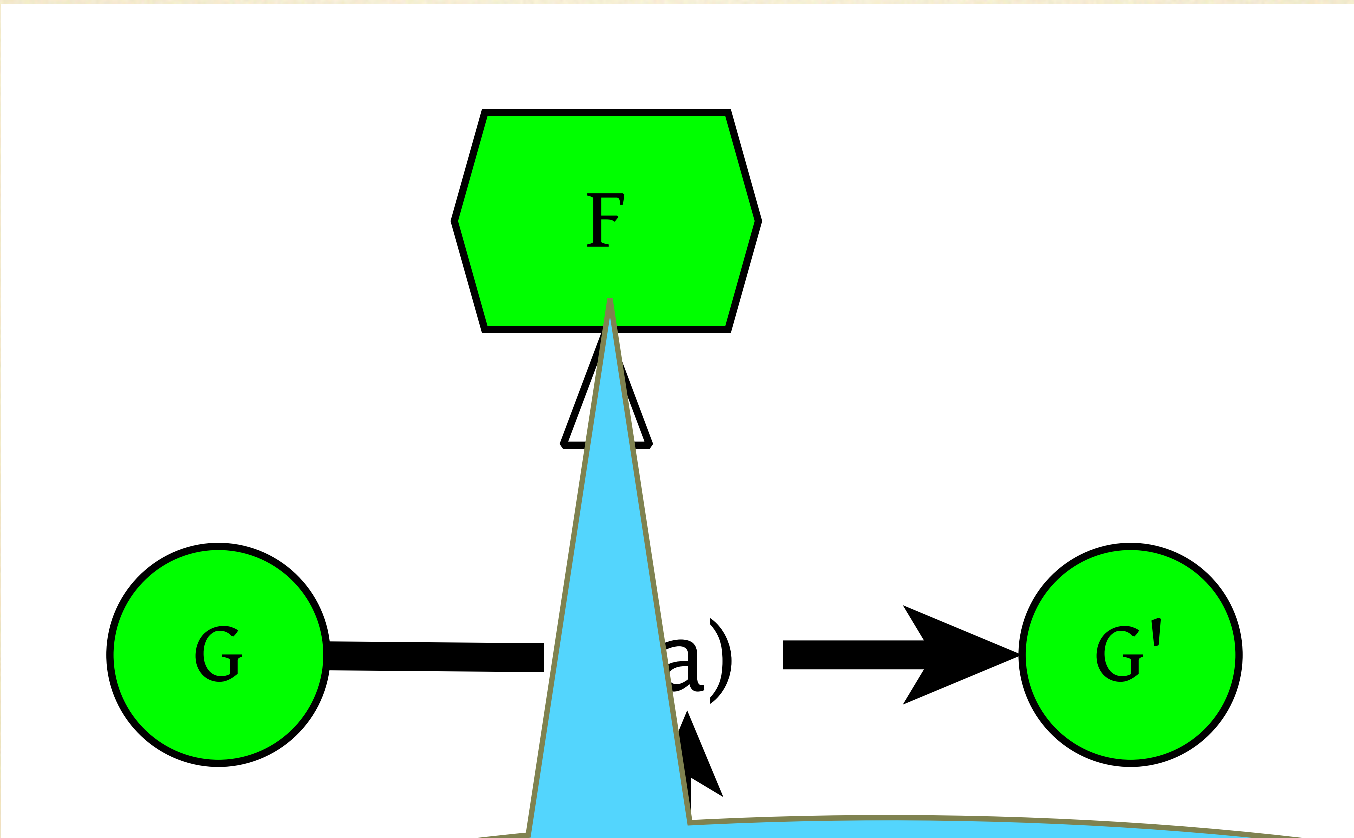
- **Transparent**
  - Full automation
  - Happening behind the scenes
  - Usually optimisations

- **Programmable**
  - Full control
  - Manually programmed
  - Generated from other artefacts

Transformation components



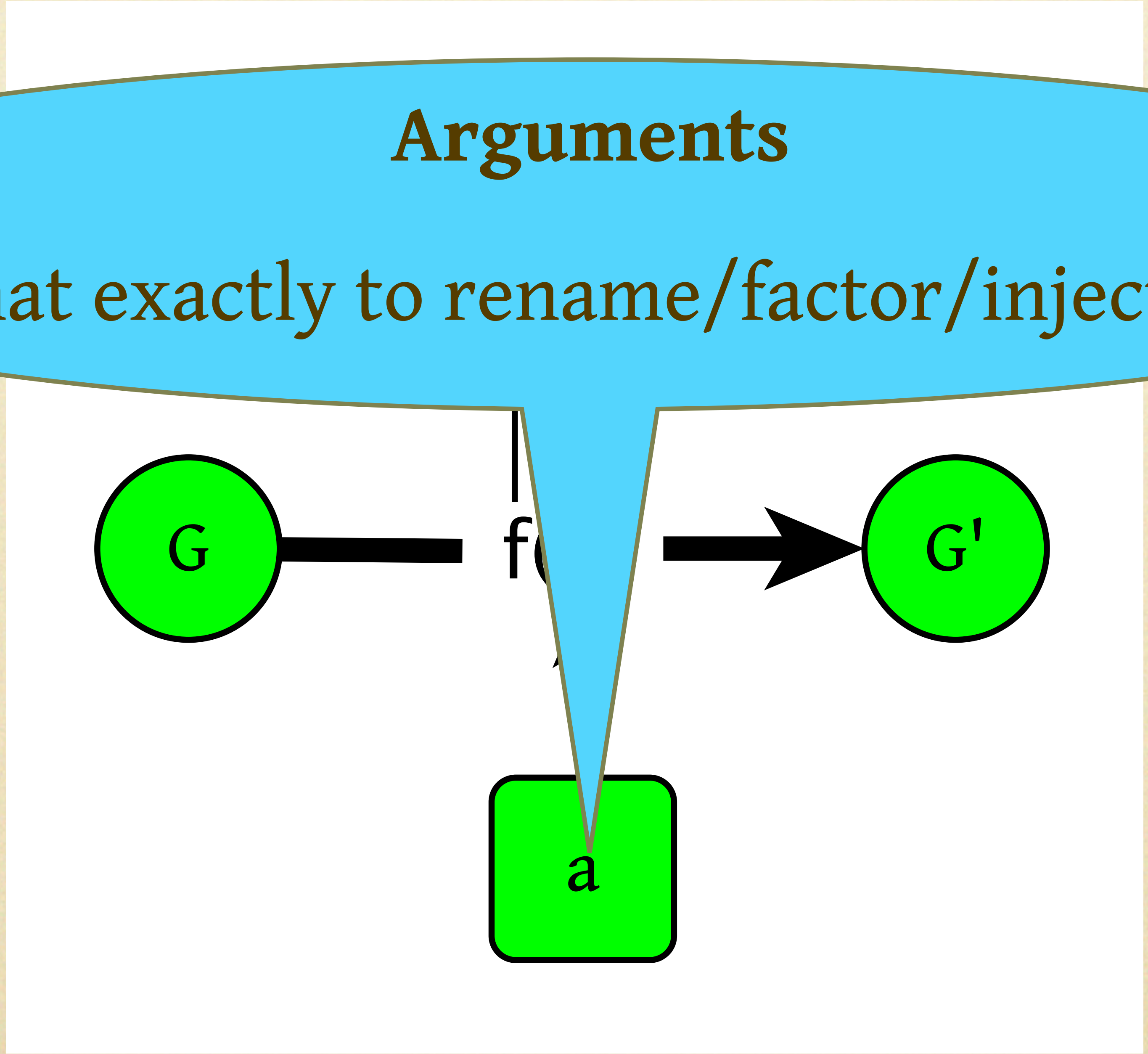
Transformation components



Operator

- known semantics, well-defined algorithm
  - rename, fold, factor, inject, remove, ...

Transformation components



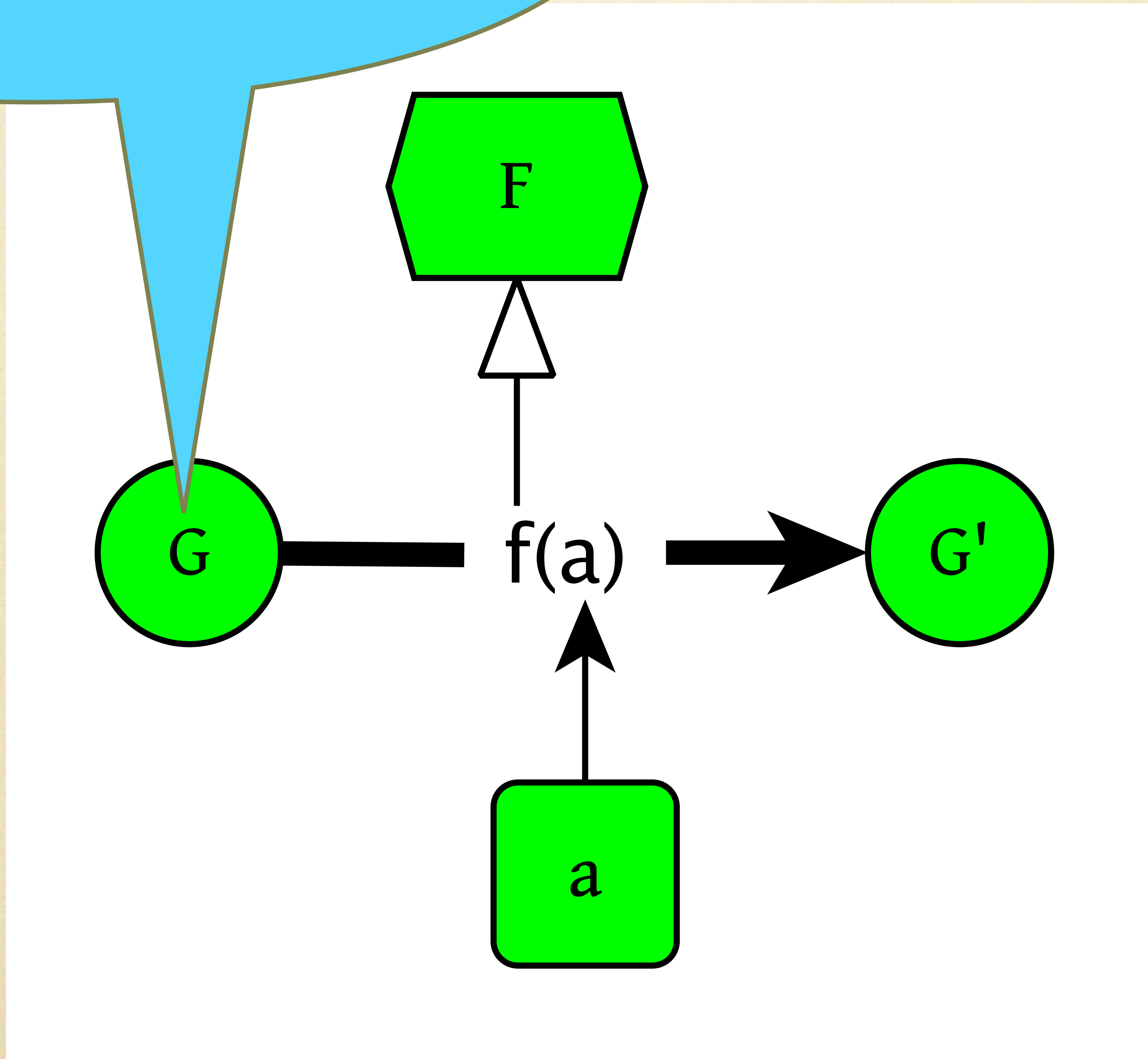
Arguments

- what exactly to rename/factor/inject/...?

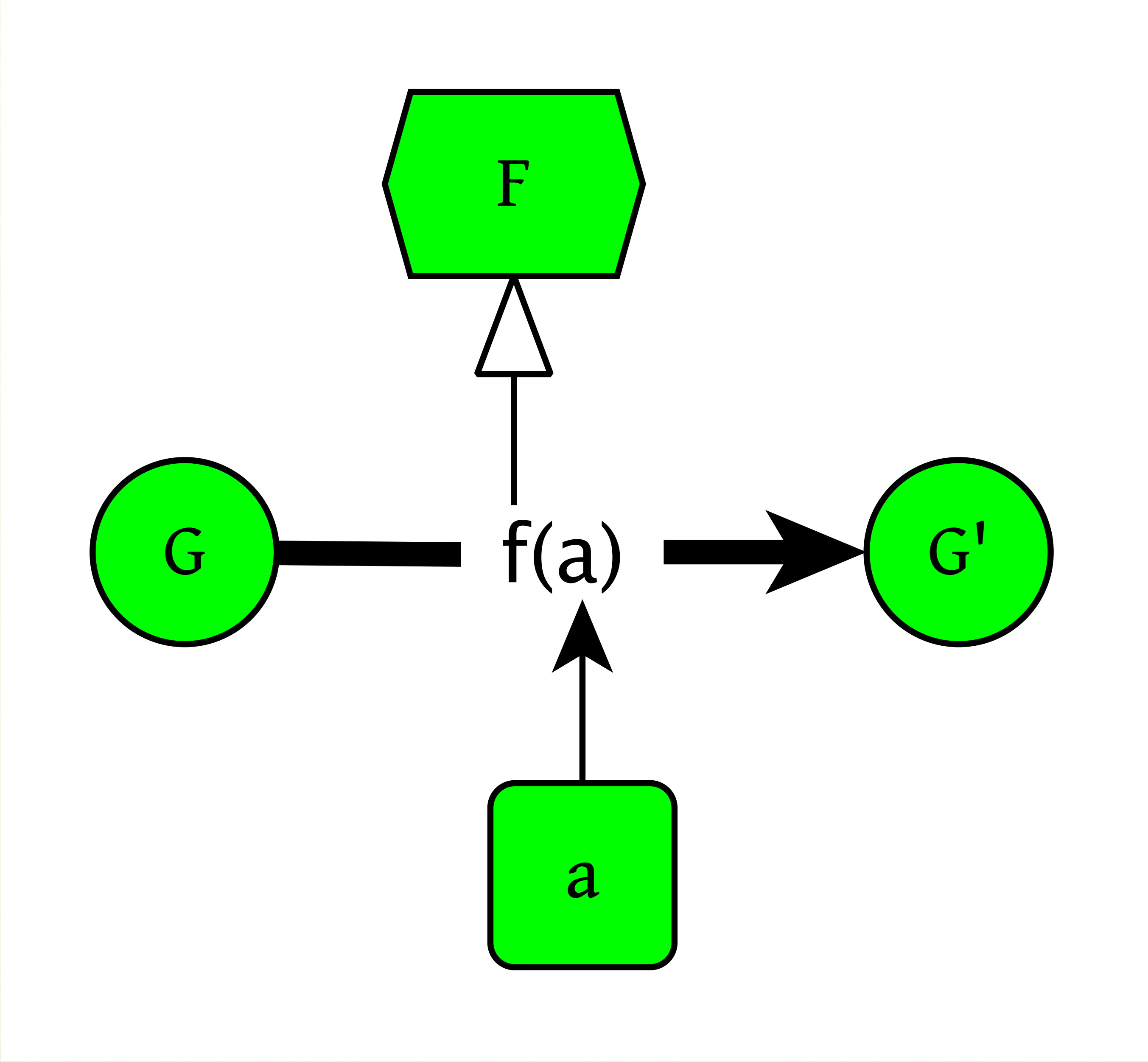
T tion components

Input grammar

- determines applicability



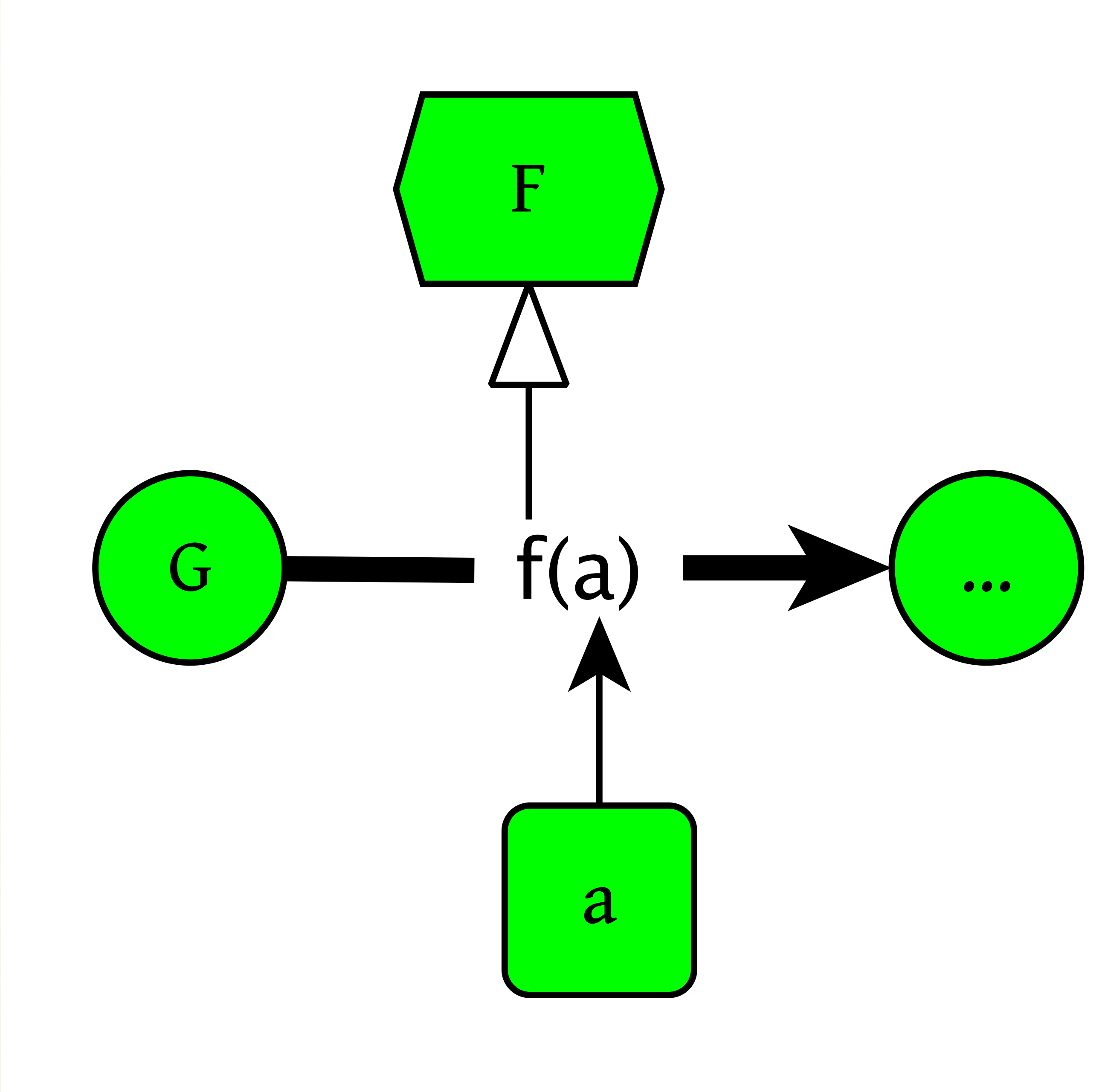
Transformation components



Transformation components

- **Operator**
  - known semantics, well-defined algorithm
  - rename, fold, factor, inject, remove, ...
- **Arguments**
  - what exactly to rename/factor/inject/...?
- Input **grammar**
  - determines applicability

Example 1:
all three components



Example 1: all three components

- Suppose we know the operator(s), the argument(s), the input
- We can execute the transformation
  - obtain the transformed grammar automatically
- We can verify applicability
- We can coevolve language instances
  - transform both the grammar and trees conforming to it
- We can test transformations with constraints
  - change impact analysis

Grammar refactoring

BGF (*read2*)

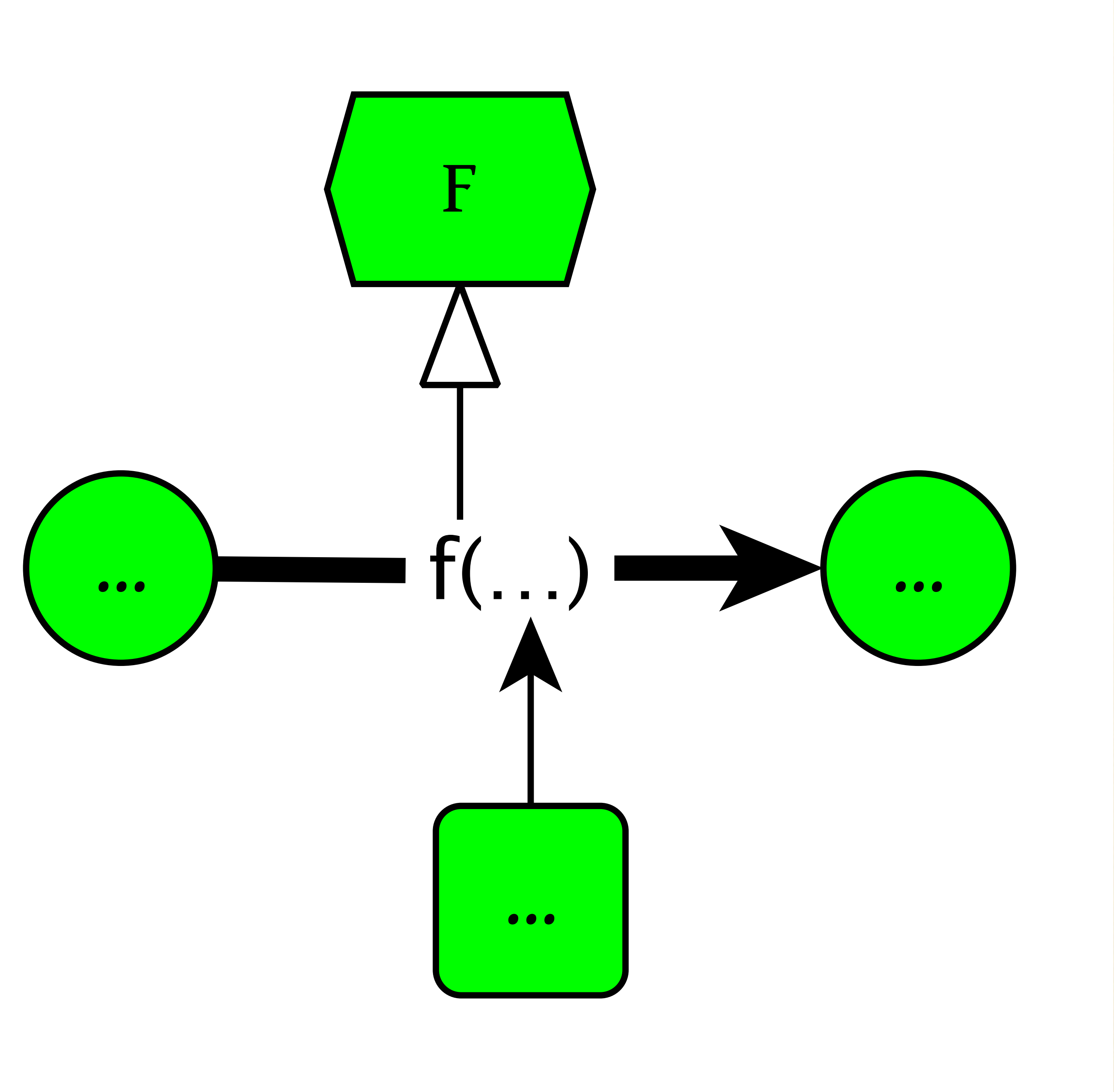
```
ClassBodyDeclarations:
        ClassBodyDeclaration
ClassBodyDeclarations:
        ClassBodyDeclarations ClassBodyDeclaration
ClassBody:
        "{" ClassBodyDeclarations? "}"
```

```
ClassBody:
        "{" ClassBodyDeclaration* "}"
```

XBGF (*grammar refactoring*)

deyaccify(ClassBodyDeclarations);
inline(ClassBodyDeclarations);
massage(
 ClassBodyDeclaration+? ,
 ClassBodyDeclaration*);

Example 2: just operators



Example 2:
just operators

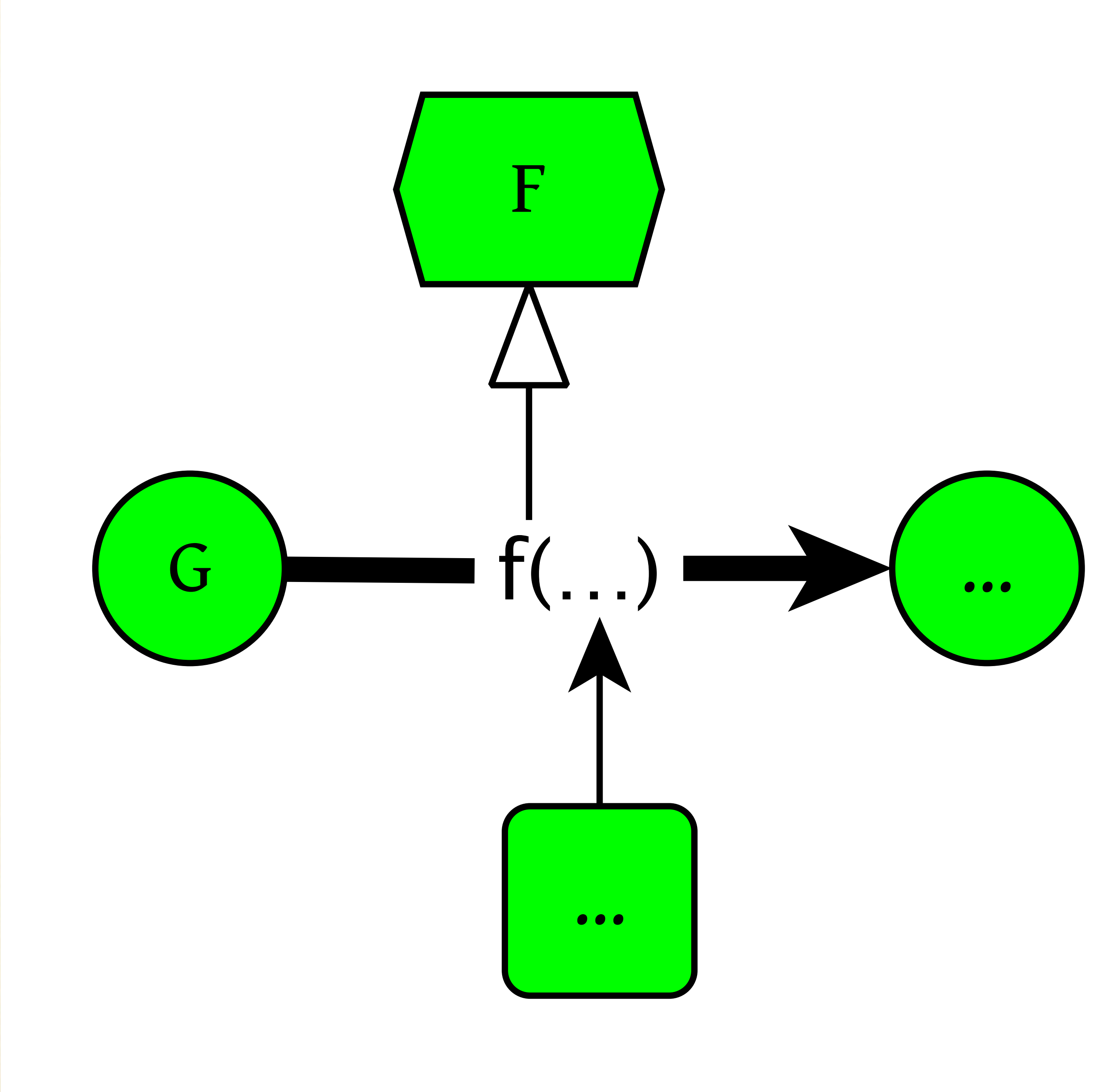
- Suppose we know the operator(s) used in the script
- We do not know/care about their arguments
- We do not know/care about the input grammar
- We still know the semantics
    - ⇒ we know certain properties of the transformation
    - ⇒ we know the relationship between input & output

Java grammar convergence

	jls1	**jls12**	**jls123**	**jls2**	**jls3**	**read12**	**read123**	**Total**
Number of lines	682	5114	2847	6774	10721	1639	3082	30859
Number of transformations	67	290	111	387	544	77	135	1611
○ Semantics-preserving (§4.2.2)	45	231	80	275	381	31	78	1121
○ Semantics-increasing/-decreasing	22	58	31	102	150	39	53	455
○ Semantics-revising	—	1	—	10	13	7	4	35
Preparation phase (§4.2.1)	1	—	—	15	24	11	14	65
○ Known bugs	—	—	—	1	11	—	4	16
○ Post-extraction	—	—	—	7	8	7	5	27
○ Initial correction	1	—	—	7	5	4	5	22
Resolution phase	21	59	31	97	139	35	43	425
○ Extension (§4.2.3)	—	17	26	—	—	31	38	112
○ Relaxation (§4.2.4)	18	39	5	75	112	—	2	251
○ Correction (§4.2.5)	3	3	—	22	27	4	3	62

	jls1	**jls12**	**jls123**	**jls2**	**jls3**	**read12**	**read123**	**Total**
○ *rename*	9	4	2	9	10	—	2	36
○ *reroot*	2	—	—	2	2	2	1	9
○ *unfold*	1	10	8	11	13	2	3	48
○ *fold*	4	11	4	11	13	2	5	50
○ *inline*	3	67	8	71	100	—	1	250
○ *extract*	—	17	5	18	30	—	5	75
○ *chain*	1	—	2	—	—	1	4	8
○ *massage*	2	13	—	15	32	5	3	70
○ *distribute*	3	4	2	3	6	—	—	18
○ *factor*	1	7	3	5	24	3	1	44
○ *deyaccify*	2	20	—	25	33	4	3	87
○ *yaccify*	—	—	—	—	1	—	1	2
○ *eliminate*	1	8	1	14	22	—	—	46
○ *introduce*	—	1	30	4	13	3	34	85
○ *import*	—	—	2	—	—	—	1	3
○ *vertical*	5	7	7	8	22	5	8	62
○ *horizontal*	4	19	5	17	31	4	4	84
○ *add*	1	14	13	7	20	28	20	103
○ *appear*	—	8	11	8	25	2	17	71
○ *widen*	1	3	—	1	8	1	3	17
○ *upgrade*	—	8	—	14	20	2	2	46
○ *unite*	18	2	—	18	21	5	4	68
○ *remove*	—	10	1	11	18	—	1	41
○ *disappear*	—	7	4	11	11	—	—	33
○ *narrow*	—	—	1	—	4	—	—	5
○ *downgrade*	—	2	—	8	3	—	—	13
○ *define*	—	6	—	4	9	1	6	26
○ *undefine*	—	3	—	5	3	—	—	11
○ *redefine*	—	3	—	8	7	6	2	26
○ *inject*	—	—	—	2	4	—	1	7
○ *project*	—	1	—	1	2	—	—	4
○ *replace*	3	1	2	3	6	1	1	17
○ *unlabel*	—	—	—	—	—	—	2	2

Example 3: operators & input



Example 3: operators & input

- We can derive arguments after seeing the grammar
- Grammar **mutation**
    - Disciplined rename (switch naming convention)
    - Remove all terminal symbols (minimalistic implode)
    - Reroot to top (if starting symbol is undefined/wrong)
    - Eliminate top (remove unconnected components)
    - Extract subgrammar (isolate one component)
    - Remove lazy nonterminals (inline or unchain)
    - Deyaccify all yaccified production rules (A:B; A:AB;)

Tough stuff

TS1: Grammar recovery

- Extraction by abstraction
- Notation-parametric automation
- Many bugs are fixed automatically, but not all
- Documentation is incomplete, incorrect, inconsistent
- Existing grammars smell bad

Grammar revision

BGF (*impl2, impl3*)

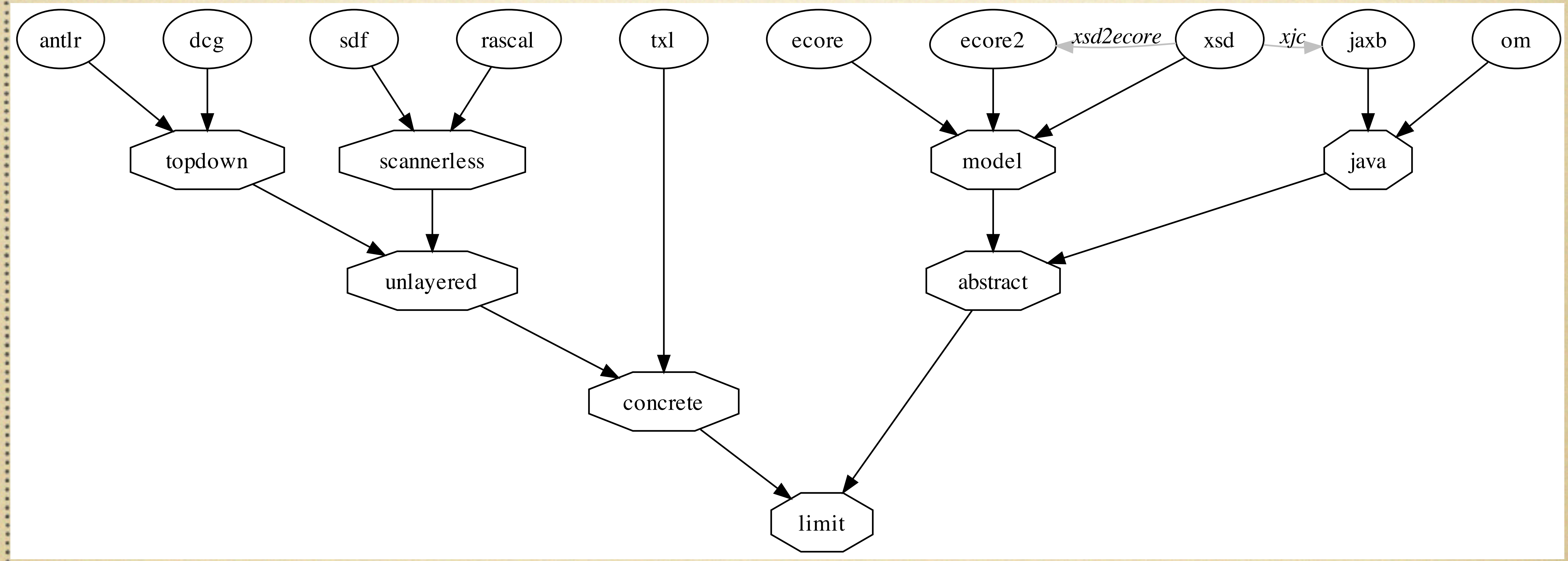
Expression2:
    Expression3 Expression2Rest ?
Expression2Rest:
    (Infixop Expression3) *
Expression2Rest:
    ~~Expression3~~ "instanceof" Type

XBGF (*grammar correction*)

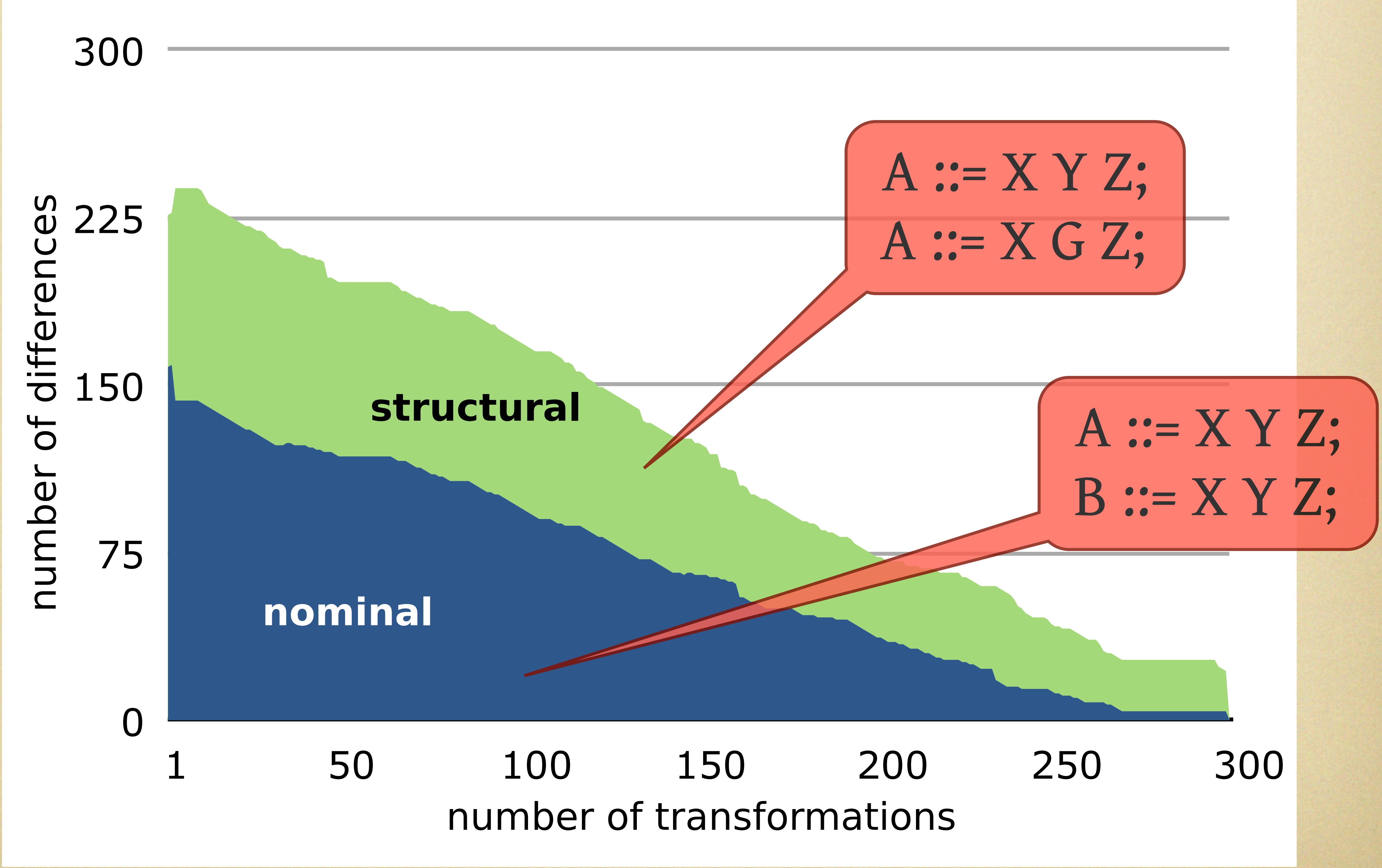
project(
 Expression2Rest:
    < Expression3 > "instanceof" Type
);

TS2: Grammar convergence

Different implementations of the same language
(parsers, data models, etc.)



Transform until equal



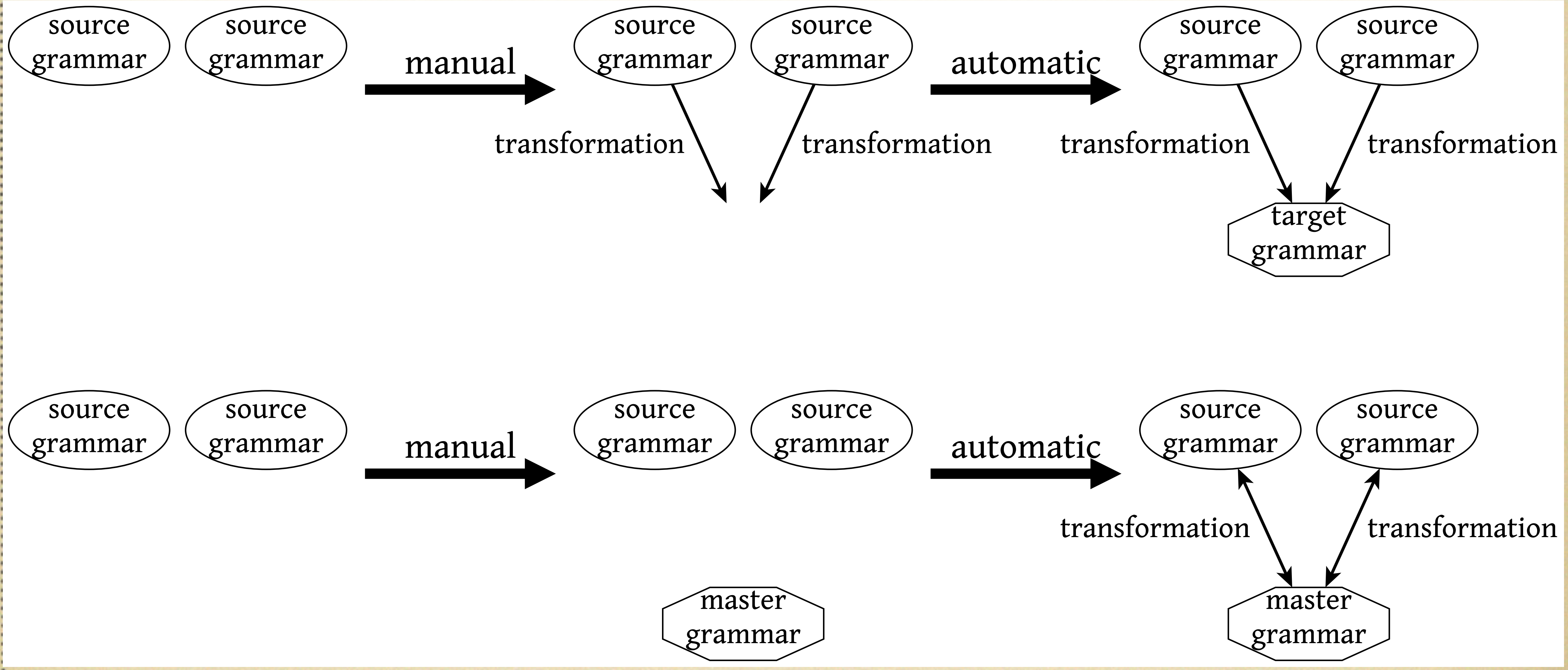
A ::= X Y Z;
A ::= X G Z;

A ::= X Y Z;
B ::= X Y Z;

TS3: Grammar product lines

- Usual framework:
  - baseline grammar
  - transformation scripts to derive other grammars
- If the baseline grammar changes
  - reapply transformations (modulo applicability fixes)
- If a derived grammar changes
  - reestablish relationships with guided convergence

Guided grammar convergence



source grammar
source grammar
manual
source grammar
source grammar
transformation
transformation
automatic
source grammar
source grammar
transformation
transformation
target grammar

source grammar
source grammar
manual
source grammar
source grammar
master grammar
automatic
source grammar
source grammar
transformation
transformation
master grammar

Guided convergence of FL

	$antlr$	dcg	sdf	$rascal$	txl	$ecore$	$ecore_2$	xsd	$jaxb$	om
One to many nonterminals	−	−	−	−	−	+	−	+	−	−
Nominal mismatches	+	+	+	+	+	+	+	+	+	+
More liberal definitions	−	−	−	−	−	−	−	−	+	+
Superfluous nonterminals	+	+	+	+	+	−	−	−	−	−
Disconnected nonterminals	−	−	−	−	−	−	−	+	+	+
Maximum number of versions	1	1	1	2	2	4	1	1	1	1
Chain production rules	+	−	−	−	−	+	+	+	+	+
Permutations	−	−	−	−	−	±	+	+	+	+
Reflexive chain rules	+	+	+	+	+	+	−	−	−	−
Undefined matched as...	ε	ε	ε	ε	ε	φ	ε	ε	ε	ε
Aggregation	−	−	−	−	−	+	−	−	−	−
Layered definitions	+	+	−	−	−	−	−	−	−	−
Meaningful chain rules	−	−	−	−	−	+	−	−	−	−

To summarise

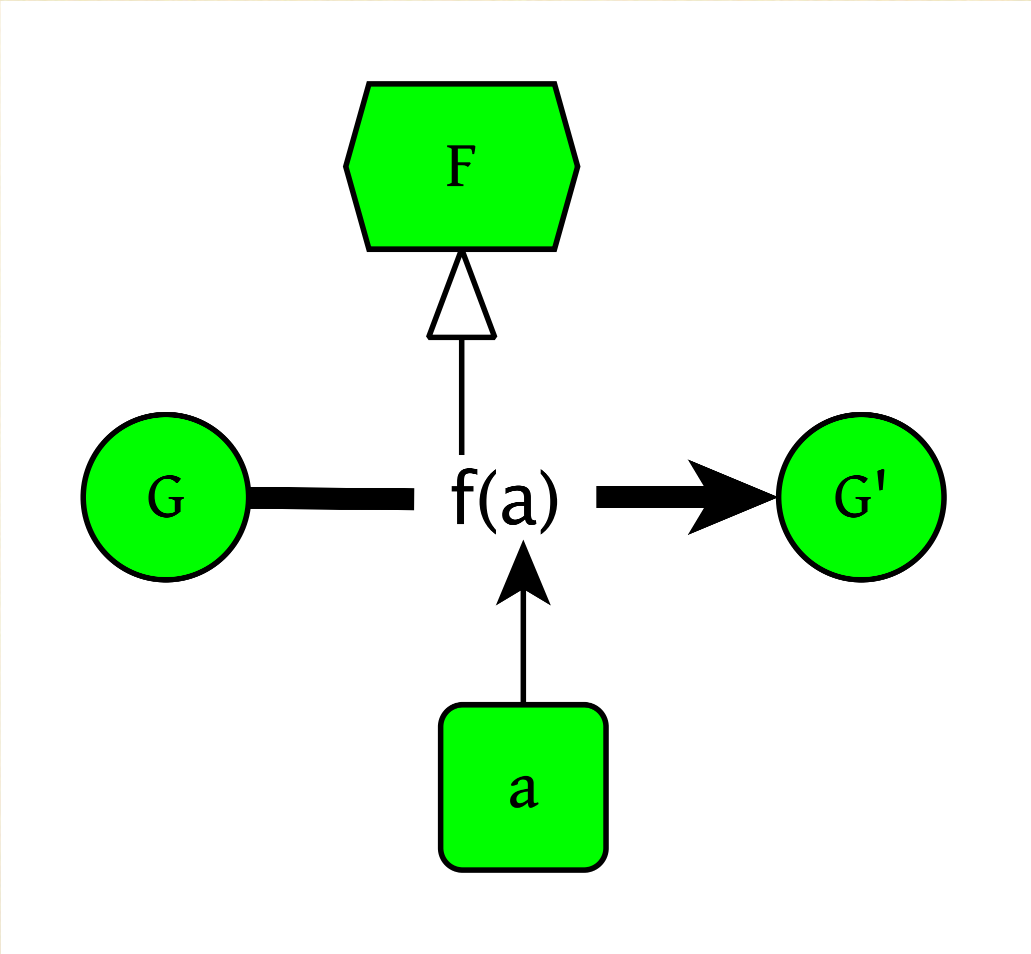
http://commons.wikimedia.org/wiki/File:Torii_kiyoshige_bando_hikosaburo_ii.jpg

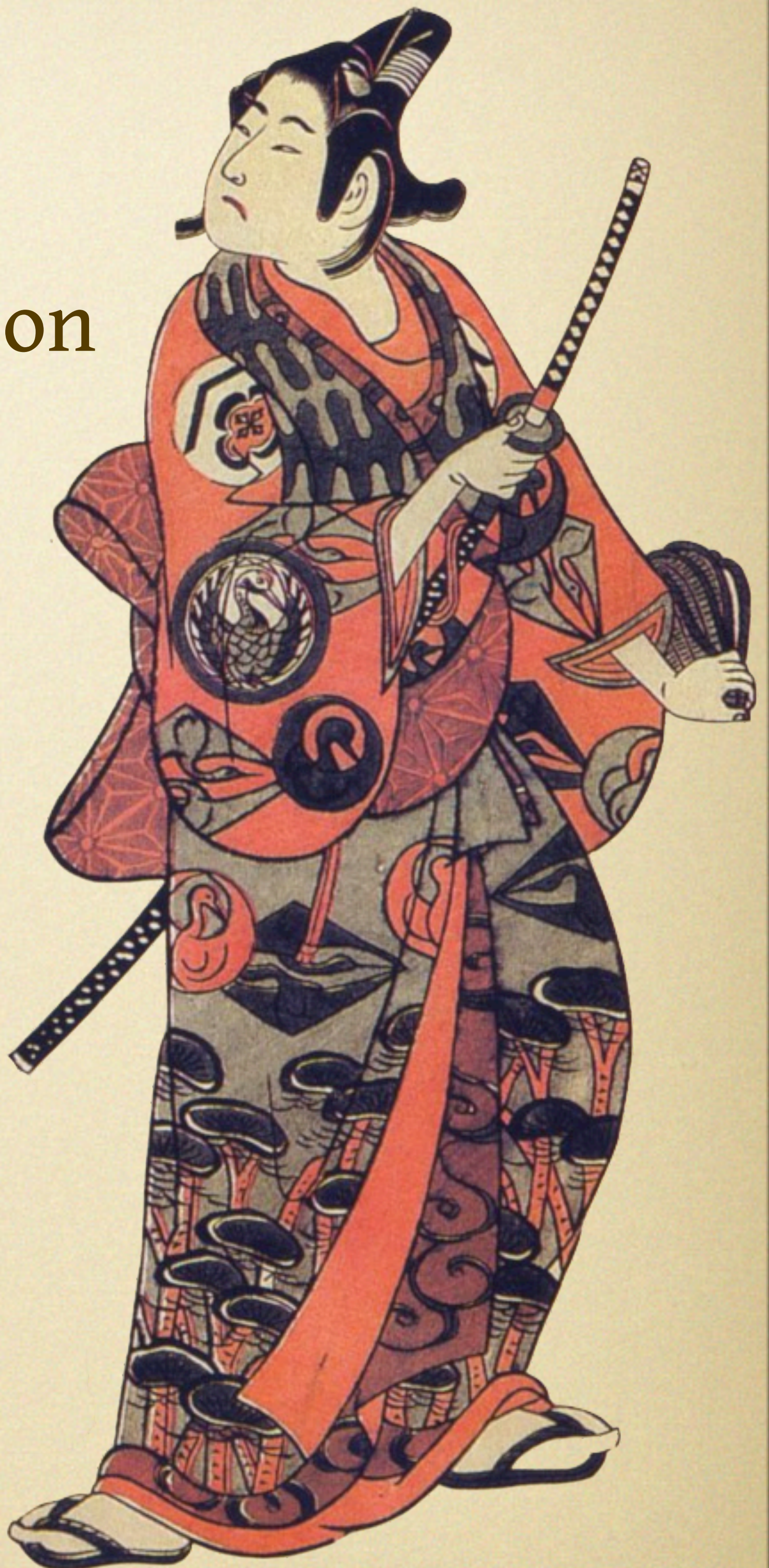
To summarise

- Software languages are everywhere
- Grammars are finite executable descriptions
- Grammarware is software based on a grammar
- Transformations are grammar differences

To summarise

- Programmable transformation
    - operator
    - arguments
    - input grammar





To summarise

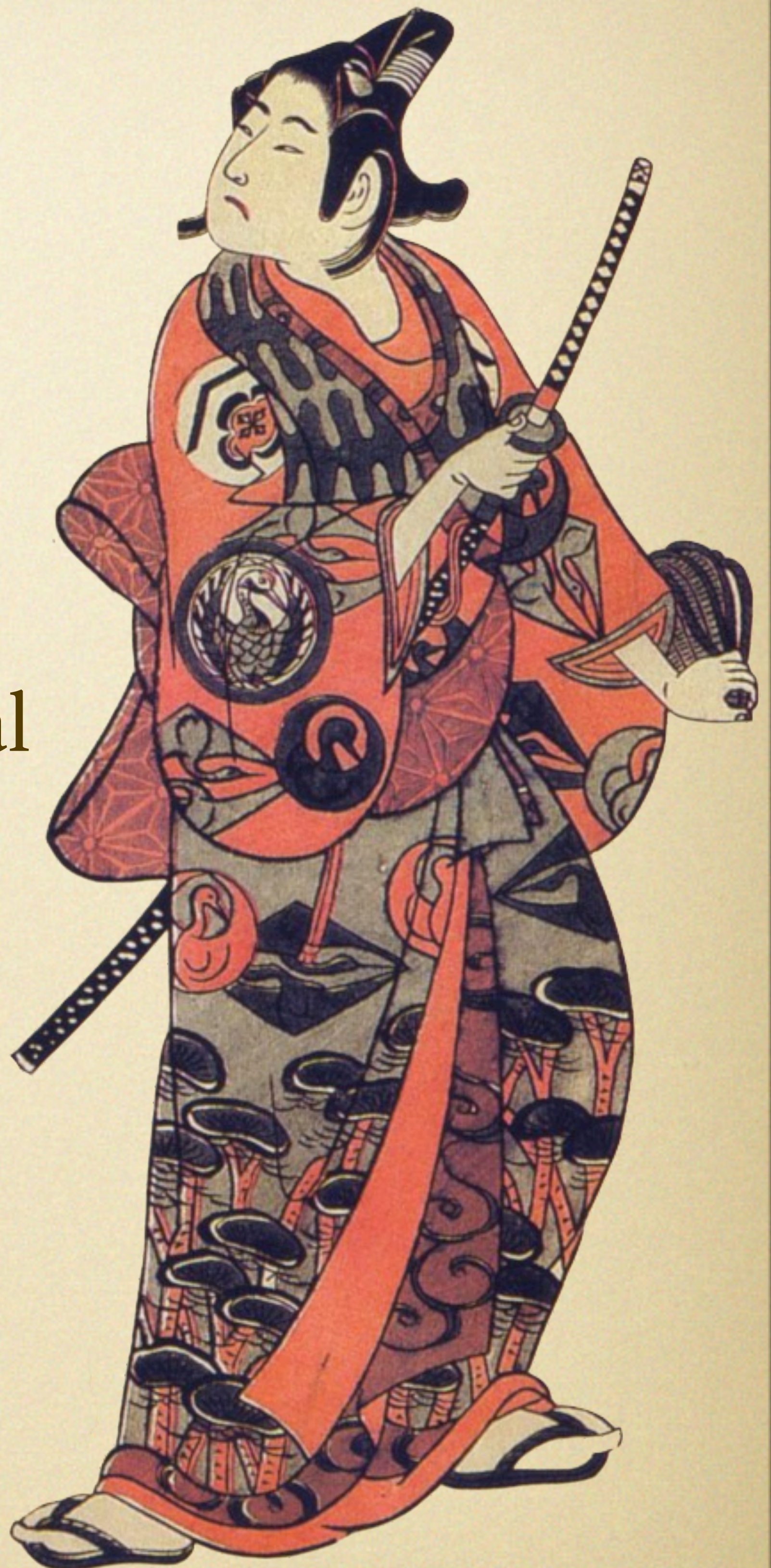
- Operator + arguments + grammar ⇒ verify, execute, coevolve, ...
- Operators ⇒ relationship
- Operators + grammar ⇒ grammar mutation

To summarise

- Grammar recovery:
  - notation-parametric
  - correction + smell removal
- Grammar convergence:
  - take related grammars
  - transform until equal
- Grammar product lines:

  → transformations

  ← guided coevolution



Bibliography

- Alex Sellink, Chris Verhoef, *Development, Assessment, and Reengineering of Language Descriptions*, ASE 1998, CSMR 2000.
- Ralf Lämmel, *Grammar Adaptation*, FME 2001, LNCS 2021, 2001.
- Ralf Lämmel, *Transformations Everywhere*, SCP 52:1–3, August 2004.
- Paul Klint, Ralf Lämmel, Chris Verhoef, *Toward an Engineering Discipline for Grammarware*, ACM ToSEM 14:3, 2005.
- 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*, SQJ 19:2, March 2011.
- Vadim Zaytsev, Ralf Lämmel, Tijs van der Storm, *Software Language Processing Suite*, 2008–2012
  - Vadim Zaytsev, *BGF Transformation Operator Suite v.1.0*, July 2010.
- Vadim Zaytsev, *Notation-Parametric Grammar Recovery*, LDTA 2012, April 2012.
- Vadim Zaytsev, *Guided Grammar Convergence*. Draft, 2012.

Questions

vadim@grammarware.net