Modeling Software Structures with GrammarLab

MoDELS 2013 Tutorial
Vadim Zaytsev, SWAT, CWI

Dr. Vadim Zaytsev

Dr. Vadim Zaytsev

РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

РГУПС РОСТОВ-НА-ДОНУ

Universiteit Twente
de ondernemende universiteit

vrije Universiteit amsterdam

UNIVERSITÄT KOBLENZ · LANDAU

CWI

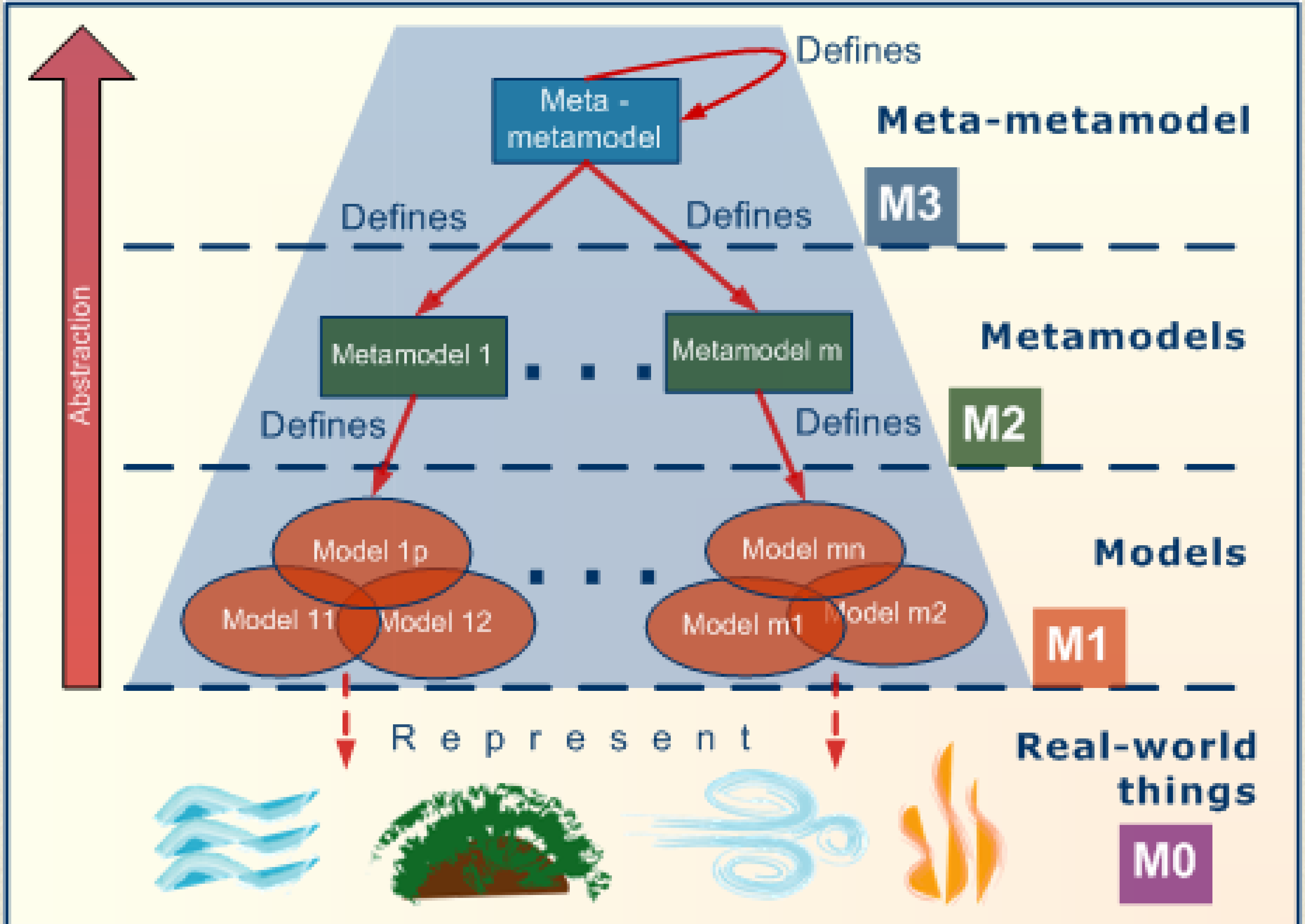
SLE

AMS 2013

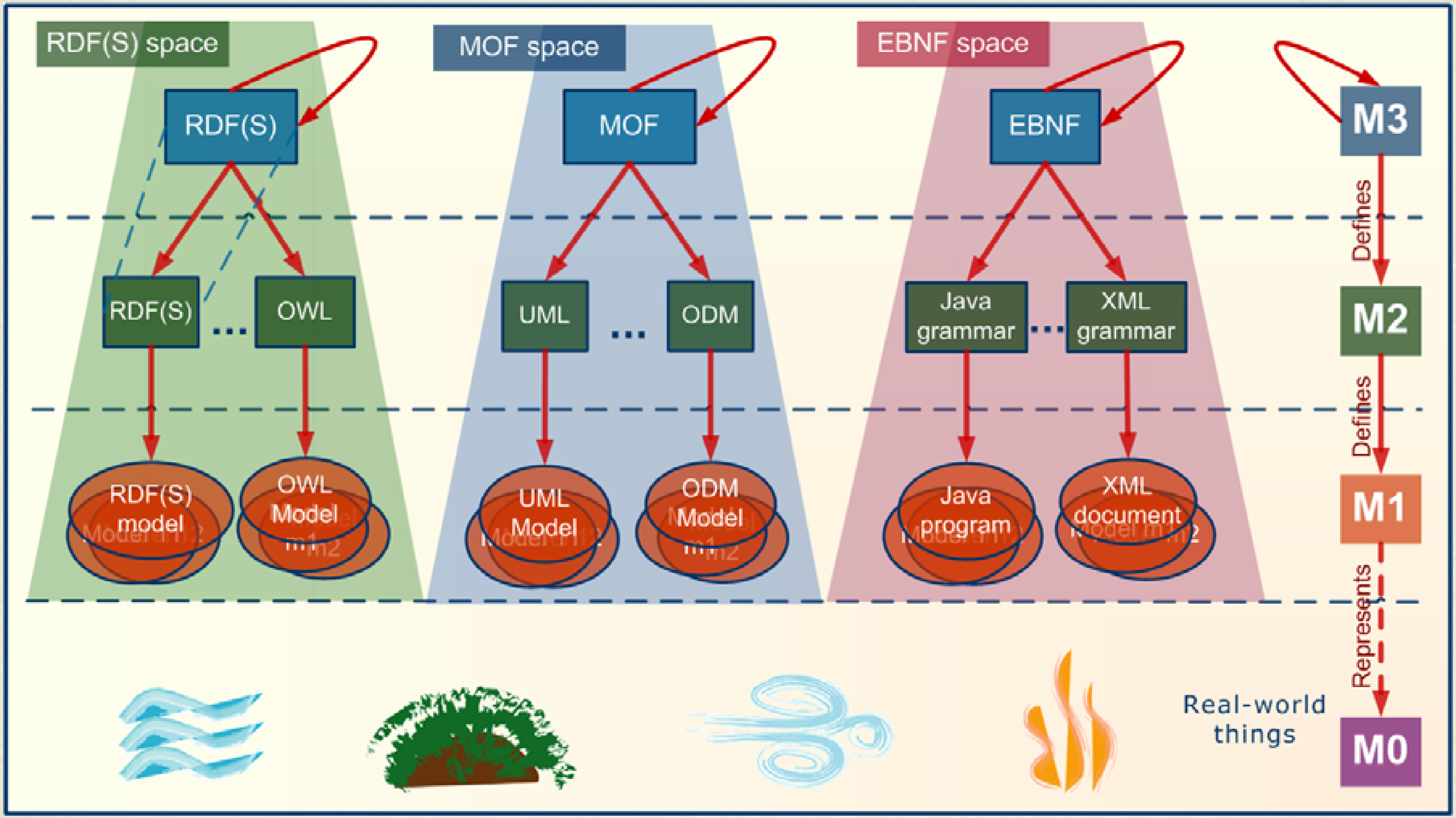
BX BIDIRECTIONAL TRANSFORMATIONS

SCAM 2013

OOP SLE 2013

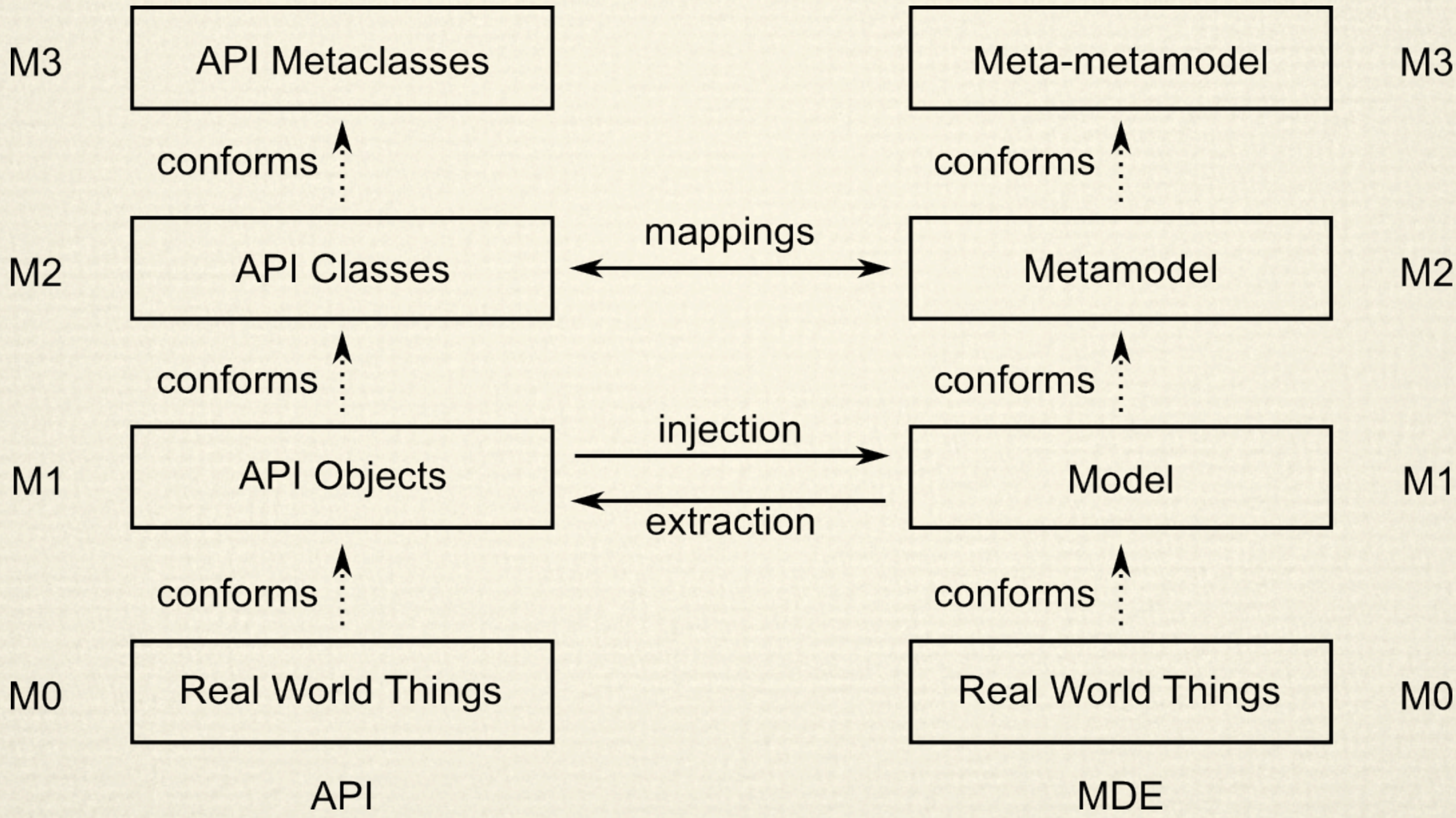


Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.

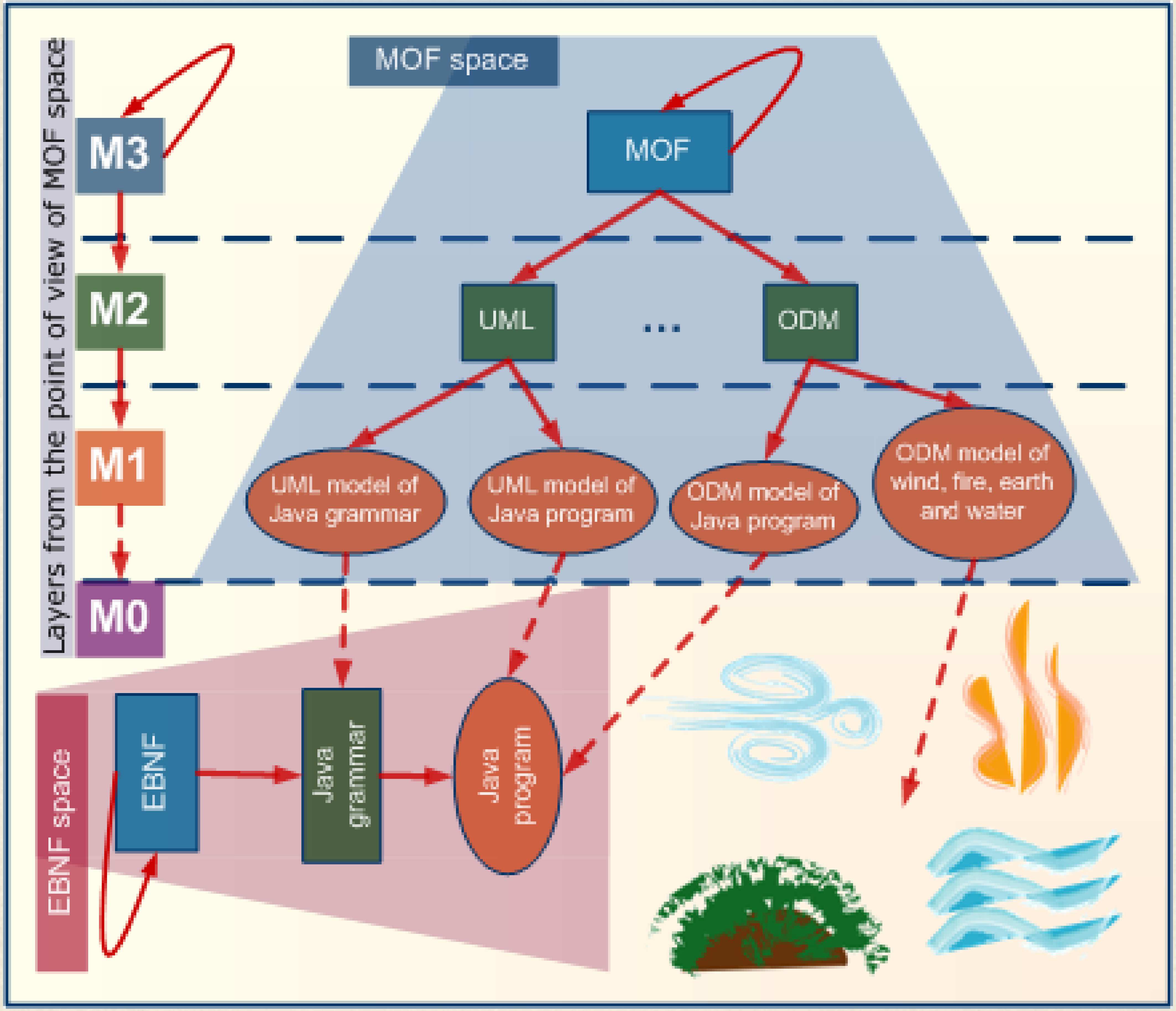


Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.

Technical side of bridging



Javier Cánovas, *Integrating APIs in the modeling realm*,
http://modeling-languages.com/integrating-apis-in-model-driven-engineering/



Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.

- M. Wimmer, G. Kramler, *Bridging Grammarware and Modelware*. Workshop on Software Model Engineering at MoDELS, 2005.
- J. Bézivin, V. Devedzic, D. Djuric, J.-M. Favre, D. Gašević, and F. Jouault. *An M3-Neutral infrastructure for bridging model engineering and ontology engineering*. International Conference on Interoperability of Enteprise Software and Applications, 2005.
- D. Djuric, D. Gaševic, V. Devedžic: *The Tao of Modeling Spaces*, Journal of Object Technology, 11(4), 2006.
- …
- J. L. C. Izquierdo, F. Jouault, J. Cabot, J. G. Molina, *Automating the building of bridges between APIs and MDE*, Information & Software Technology, 54(3), 2012.
- G. Guizzardi, V. Zamborlini: *A Common Foundational Theory for Bridging Two Levels in Ontology-Driven Conceptual Modeling*. International Conference on Software Language Engineering, 2012.
- …

To summarise

- Real world is shared
- ~~Models~~ Grammars everywhere
- Reuse across TSs
- Methodology sharing

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

by Actually That Guy, Sep 2013

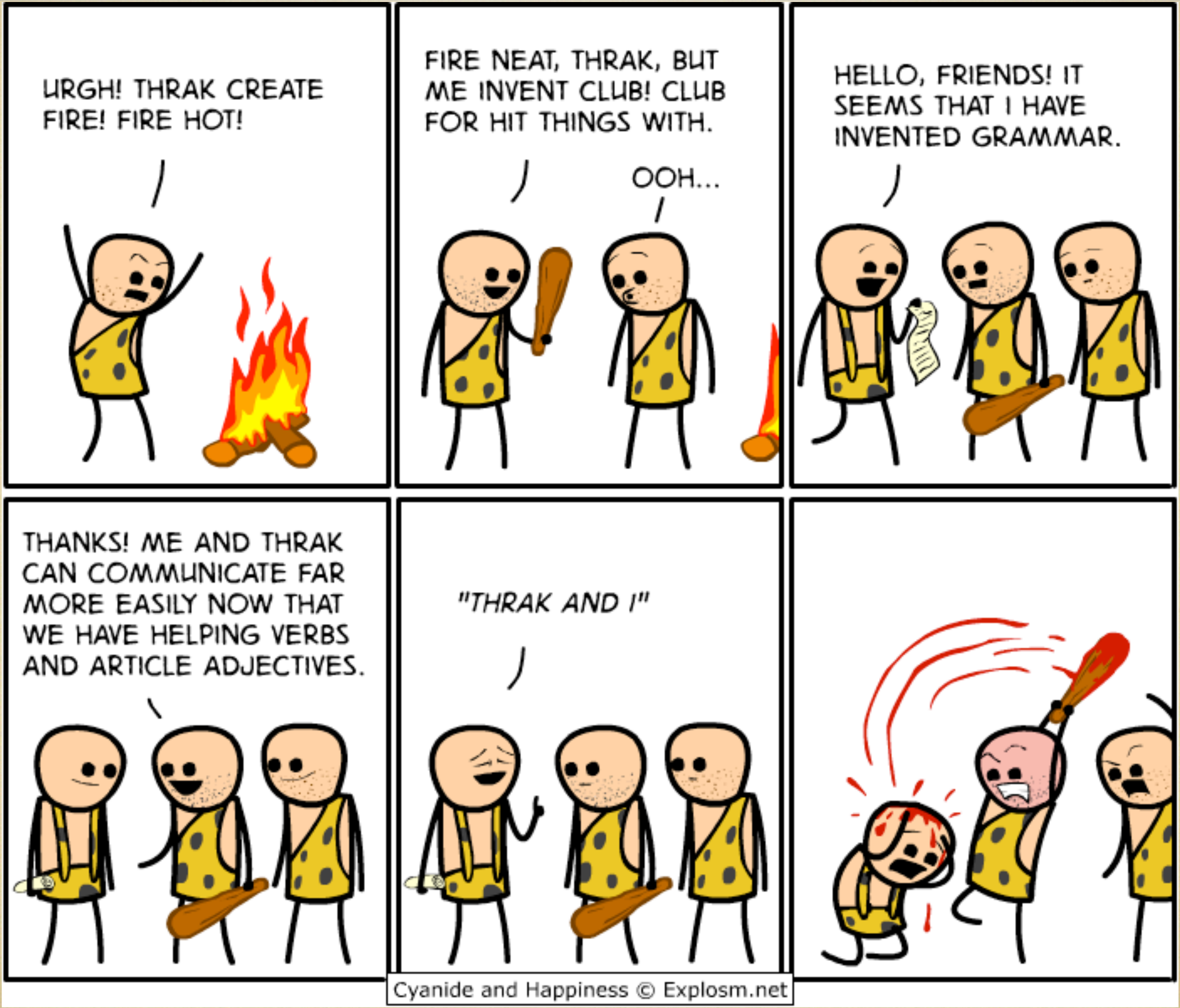
Tutorial ::=
intro
grammars
inside
outside
rascal
grammarlab

Grammars
in a broad sense

Paul Klint, Ralf Lämmel, Chris Verhoef, *Toward an Engineering Discipline for Grammarware*, ACM TOSEM 14(3), 2005.

Grammars in history

- Panini (IV BC)
  - Ashtadhyayi grammar
- Chomsky (1956)
  - hierarchy of grammars
- Backus (1959), Naur (1963), Wirth (1977)
  - notation for grammars


URGH! THRAK CREATE FIRE! FIRE HOT!
FIRE NEAT, THRAK, BUT ME INVENT CLUB! CLUB FOR HIT THINGS WITH.
OOH...
HELLO, FRIENDS! IT SEEMS THAT I HAVE INVENTED GRAMMAR.
THANKS! ME AND THRAK CAN COMMUNICATE FAR MORE EASILY NOW THAT WE HAVE HELPING VERBS AND ARTICLE ADJECTIVES.
"THRAK AND I"
Cyanide and Happiness © Explosm.net

Grammars in SE

- Definitions of languages
  - Finite definitions
  - of infinite languages
- Focus on sets of strings (“words”)
- Also, rewriting systems
- Analytic or generative

Arithmetic Expressions, Boolean Expressions, and Expressions

⟨factor⟩:≡ ⟨number⟩ or ⟨function⟩ or ⟨variable⟩ or ⟨subscr var⟩
or (⟨ar exp⟩) or ⟨factor⟩ ↑⟨ar exp⟩↓

⟨term⟩:≡ ⟨factor⟩ or ⟨term⟩×⟨factor⟩ or ⟨term⟩/⟨factor⟩

⟨ar exp⟩:≡ ⟨term⟩ or +⟨term⟩ or -⟨term⟩ or ⟨ar exp⟩+⟨term⟩
or ⟨ar exp⟩ - ⟨term⟩

⟨ar exp A⟩:≡ ⟨ar exp⟩

⟨relation⟩:≡ < or > or ≤ or ≥ or = or ≠

⟨rel exp⟩:≡ (⟨ar exp⟩ ⟨relation⟩ ⟨ar exp A⟩)

⟨bool term⟩:≡ 0 or 1 or ⟨rel exp⟩ or ⟨function⟩ or
⟨variable⟩ or ⟨subscr var⟩ or (⟨bool exp⟩)
or ¬⟨bool term⟩

⟨bool exp⟩:≡ ⟨bool term⟩ or ⟨bool exp⟩ ⋁ ⟨bool term⟩
or ⟨bool exp⟩ ⋀ ⟨bool term⟩ or
⟨bool exp⟩ ≡ ⟨bool term⟩

⟨exp⟩:≡ ⟨ar exp⟩ or ⟨bool exp⟩

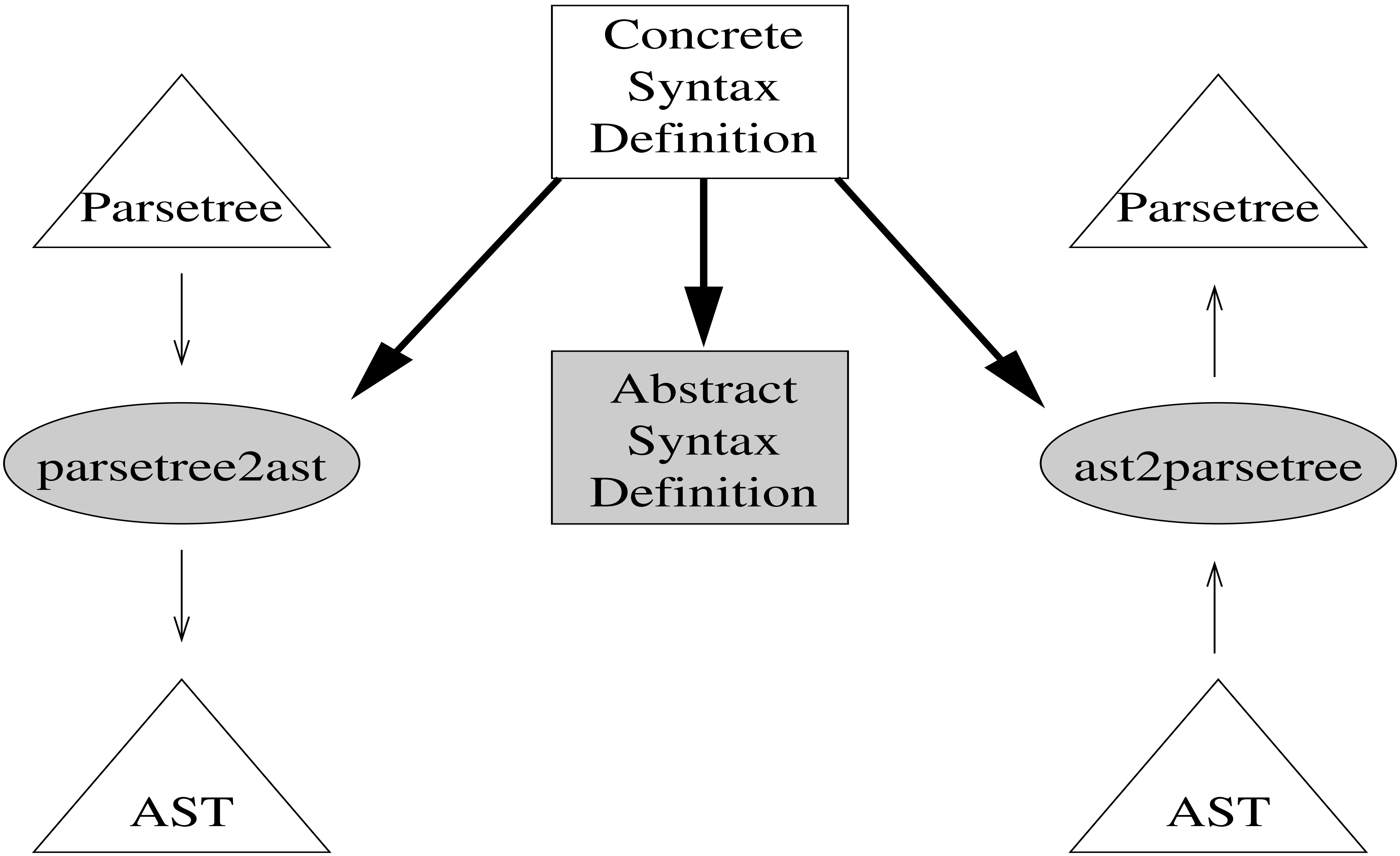
John Backus, The Syntax and Semantics of the Proposed IAL. ICIP, 1960.

$$
\begin{aligned}
S &\to aSB\&AB \mid b \\
A &\to aA \mid \varepsilon \\
B &\to B_1 \mid B_2 \\
B_1 &\to B_1B_3\&B_2B_2 \mid b \\
B_2 &\to B_1B_1\&B_2B_6 \mid bb \\
B_3 &\to B_1B_2\&B_6B_6 \mid bbb \\
B_6 &\to B_1B_2\&B_3B_3
\end{aligned}
$$

Grammars as contracts

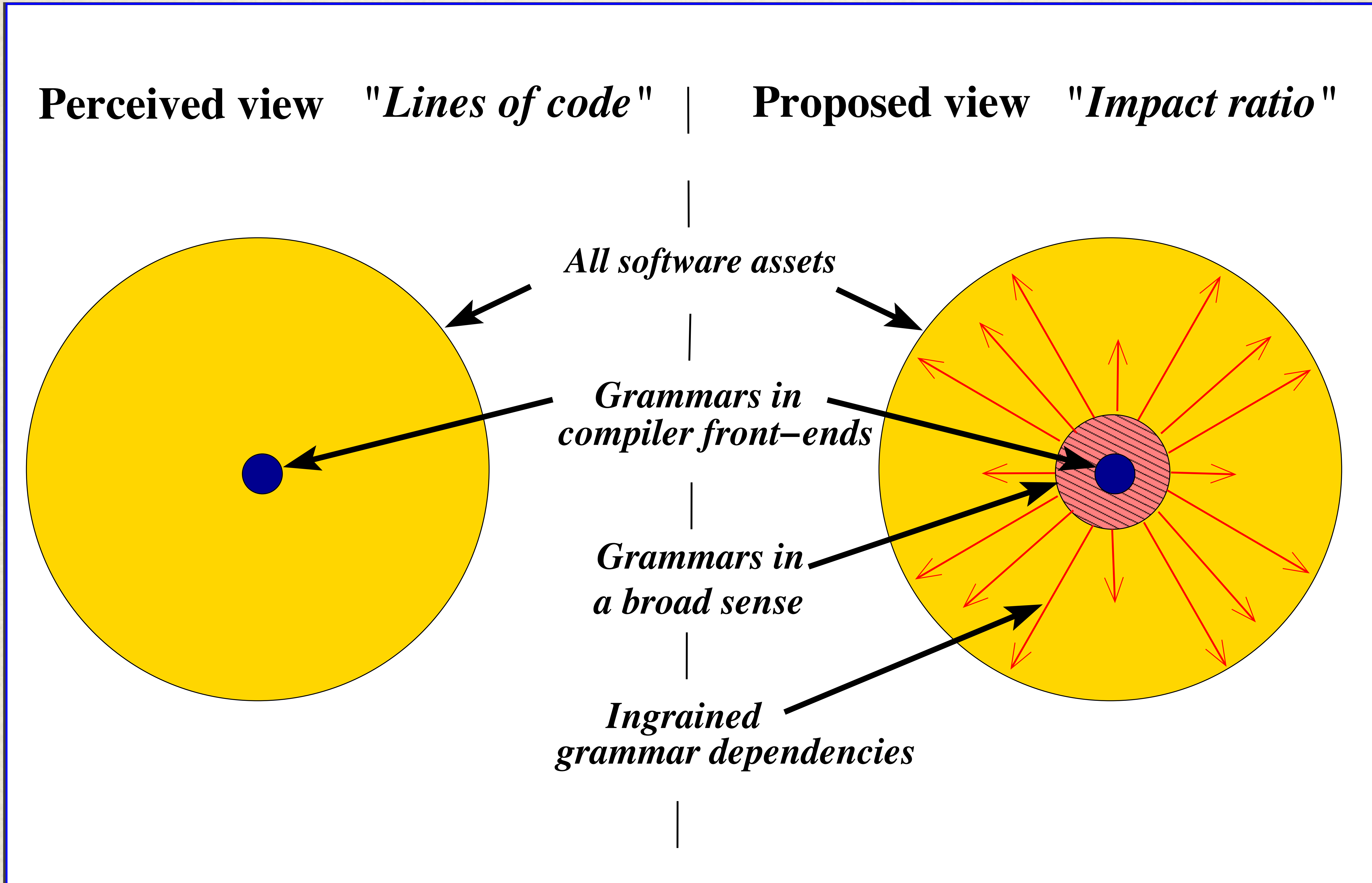
- Many metatools
  - each implies its own language
- Interoperability
  - depends on conforming to the same language

de Jonge, Visser. Grammars as Contracts, GCBSE 2001.


Concrete Syntax Definition
Parsetree
Parsetree
parsetree2ast
Abstract Syntax Definition
ast2parsetree
AST
AST

Grammars as commitments

- Intercommunication purposes
- Bridging-in-the-small
  - Assume two independent collaborators
  - Each side has its own infrastructure
  - E.g., XML and ANTLR
  - Information exchange is crucial
  - The system is evolving

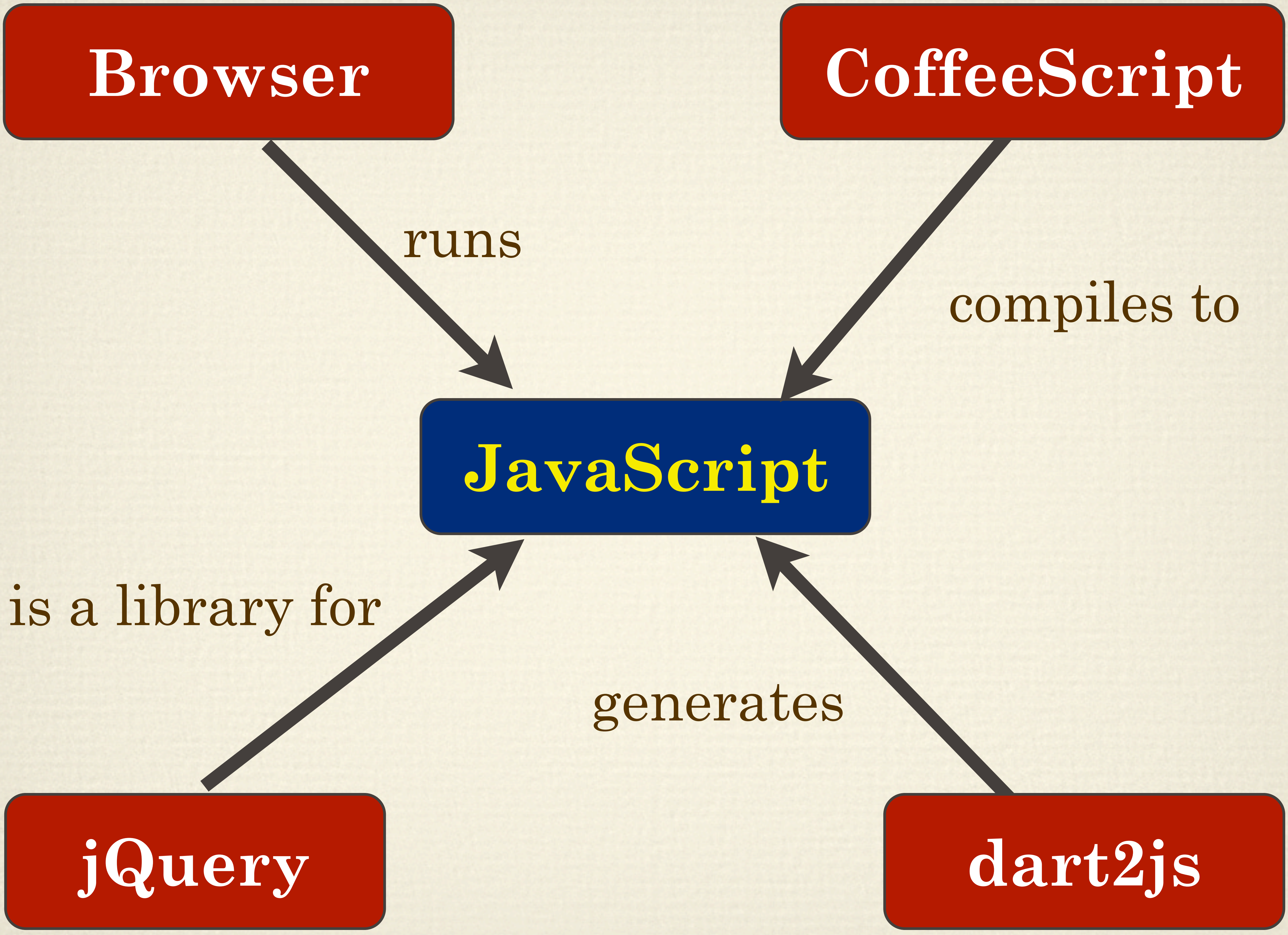


Klint, Lämmel, Verhoef, *Toward an Engineering Discipline for Grammarware*, ACM TOSEM 14(3), 2005.

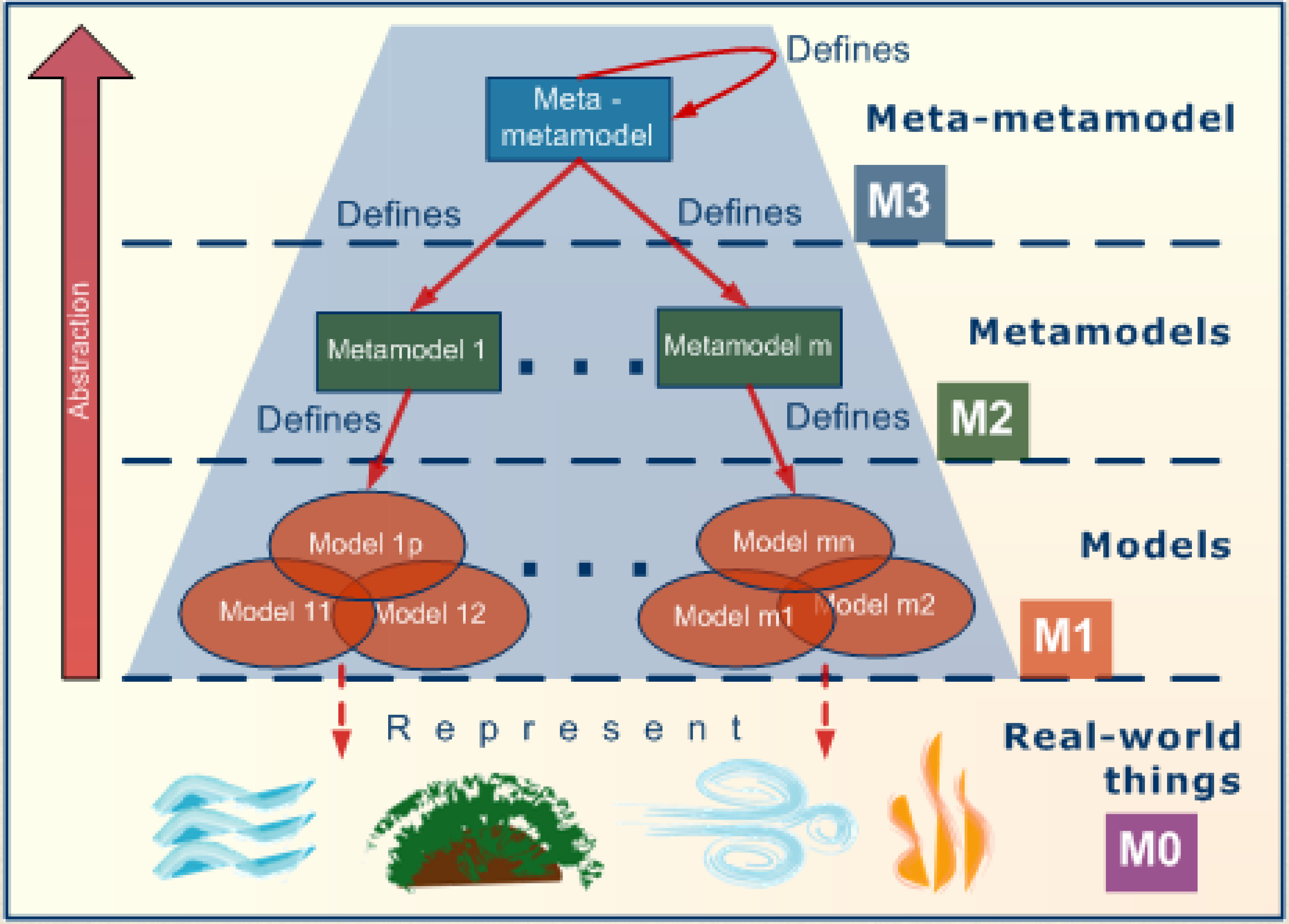
Grammars as interfaces

- Algebraic data types
- Domain models
- Class diagrams
- Libraries
- API
- Questionnaires

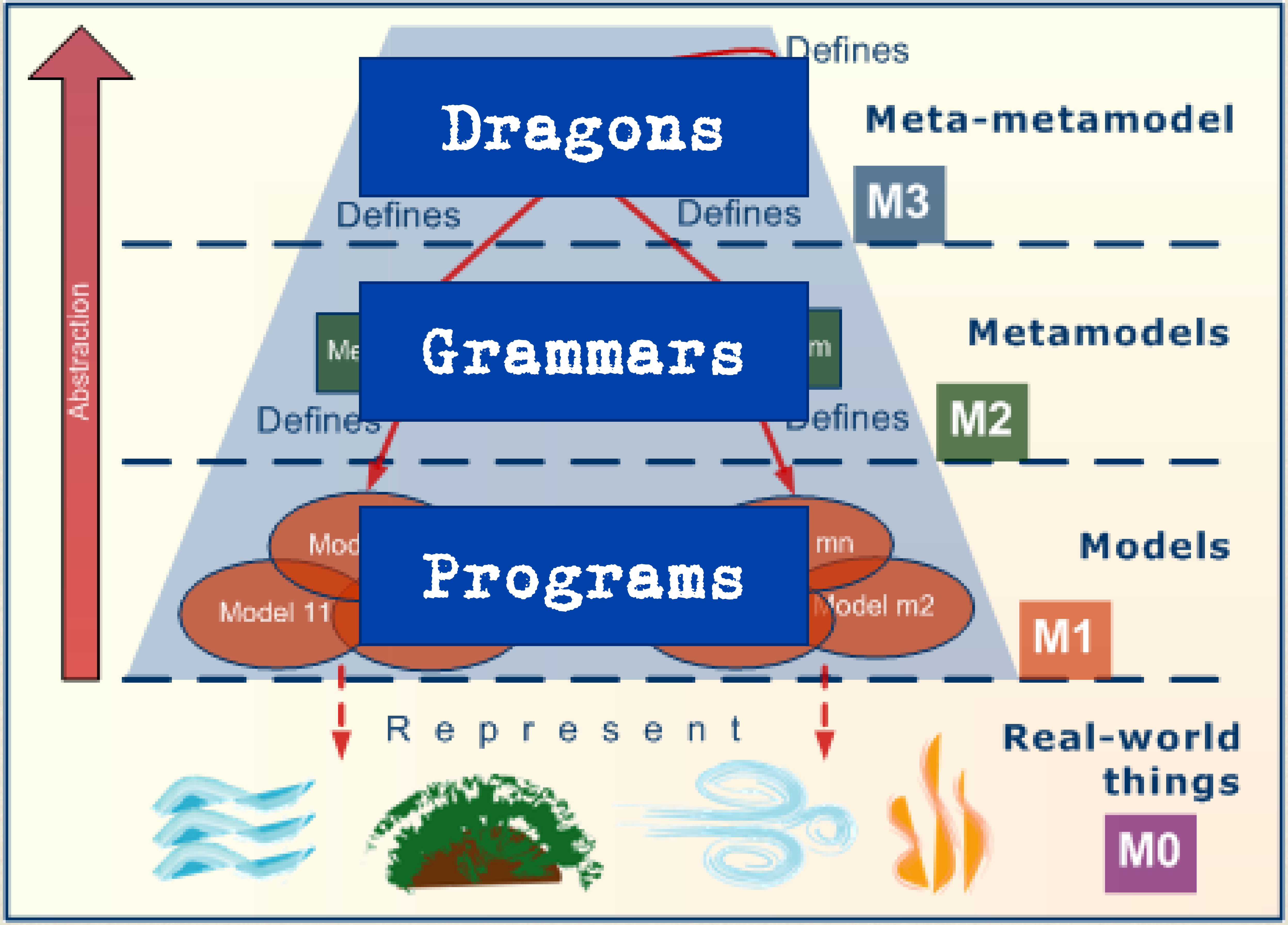
JavaScript


Browser
CoffeeScript
runs
compiles to
JavaScript
is a library for
generates
jQuery
dart2js

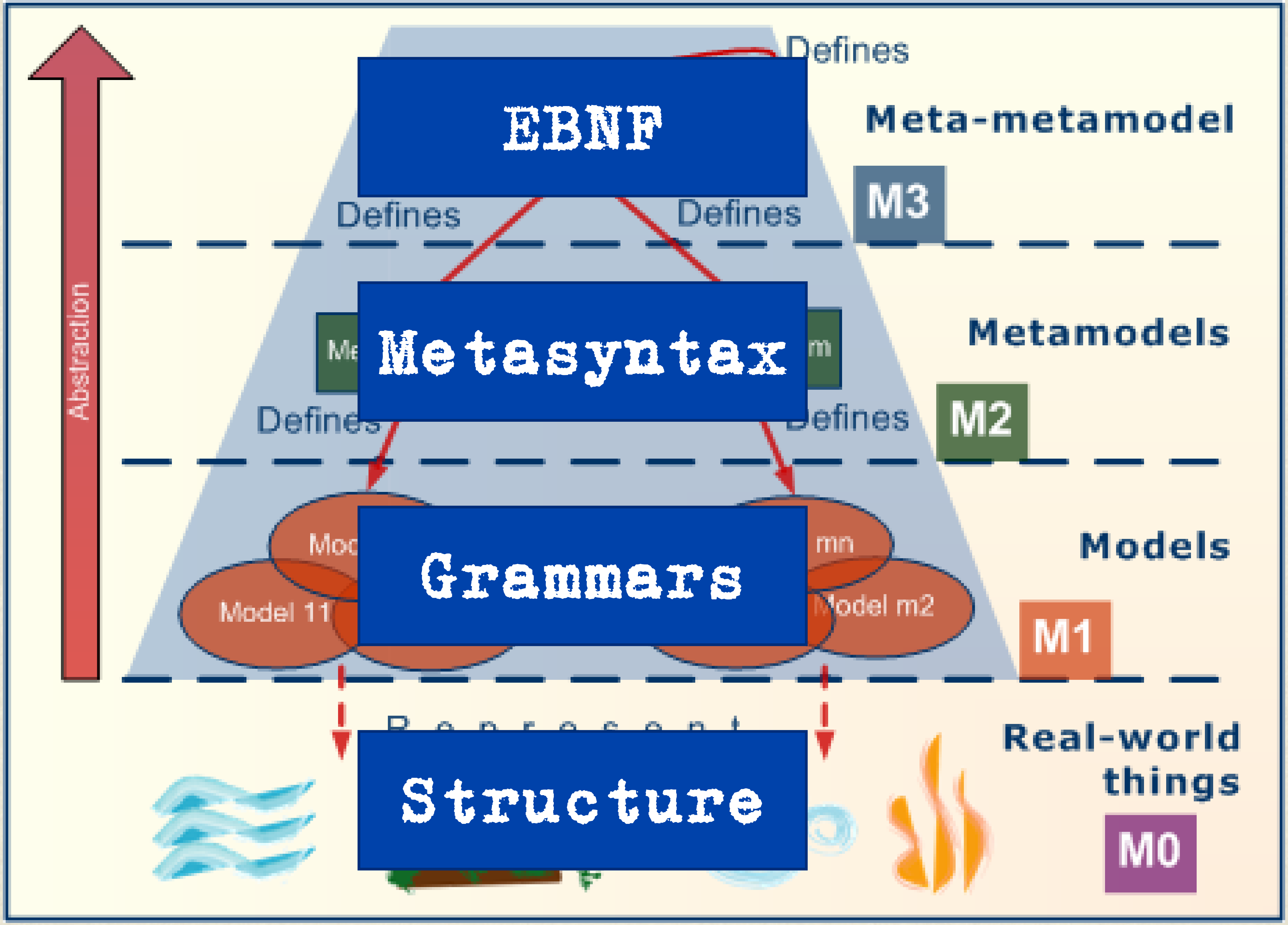

SONY



Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.



Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.

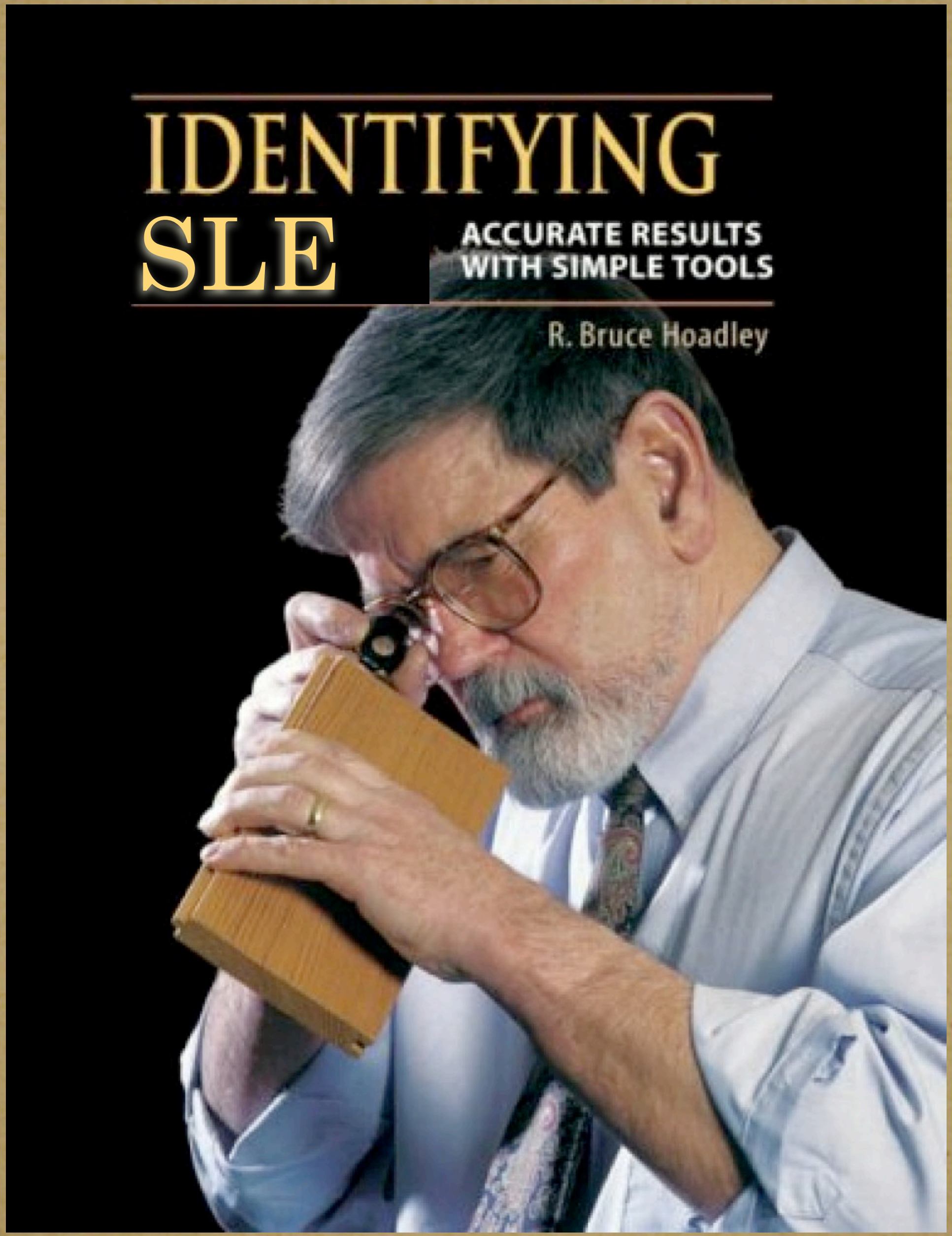


Dragan Djuric, Dragan Gaševic, Vladan Devedžic: *The Tao of Modeling Spaces*, JOT 2006.

To summarise

- Grammars define languages
- Represent structural commitments
- Contracts / interfaces
- Bridgebuilding material
- *Grammars in a broad sense model software languages*

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

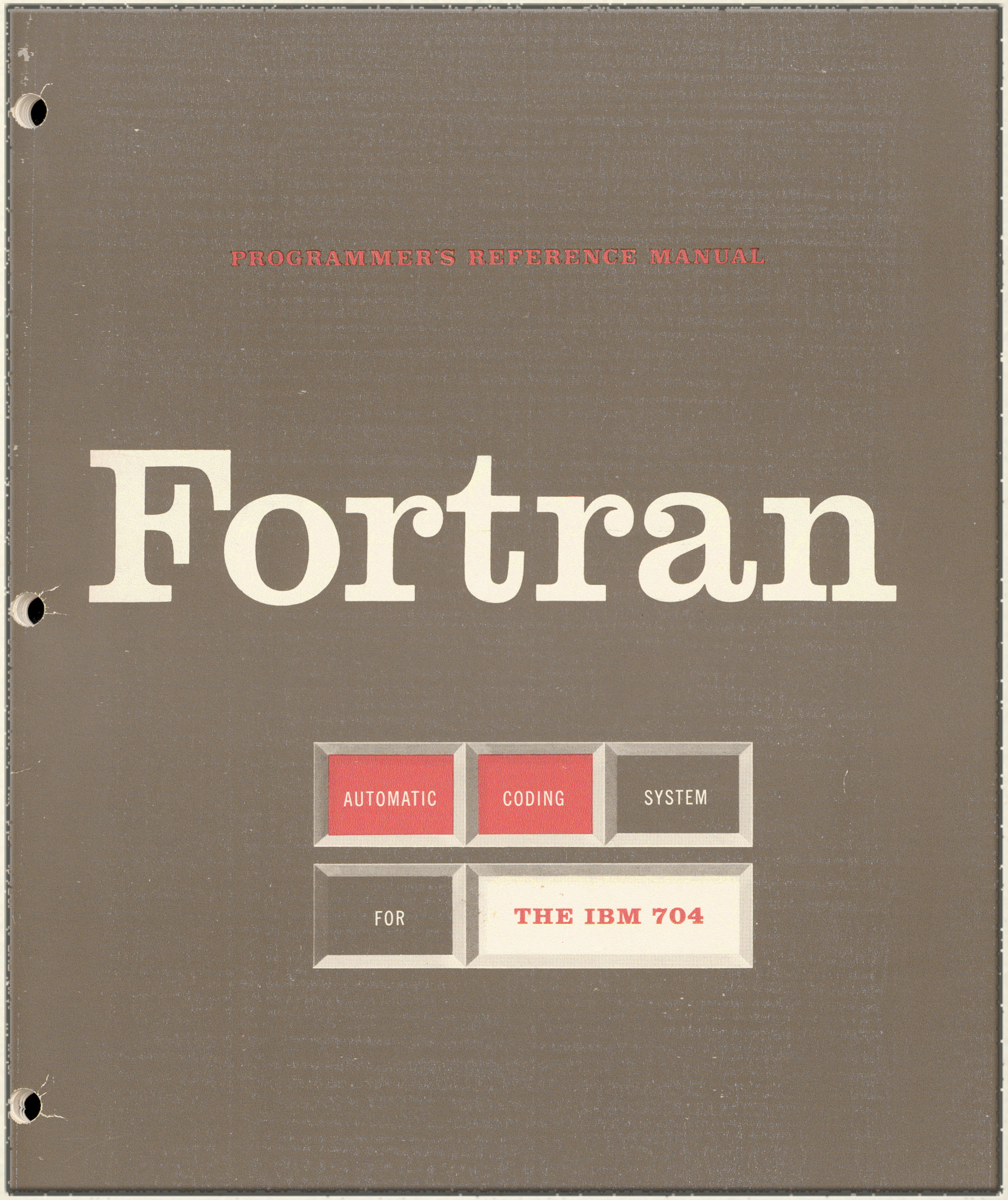


What is a software language?

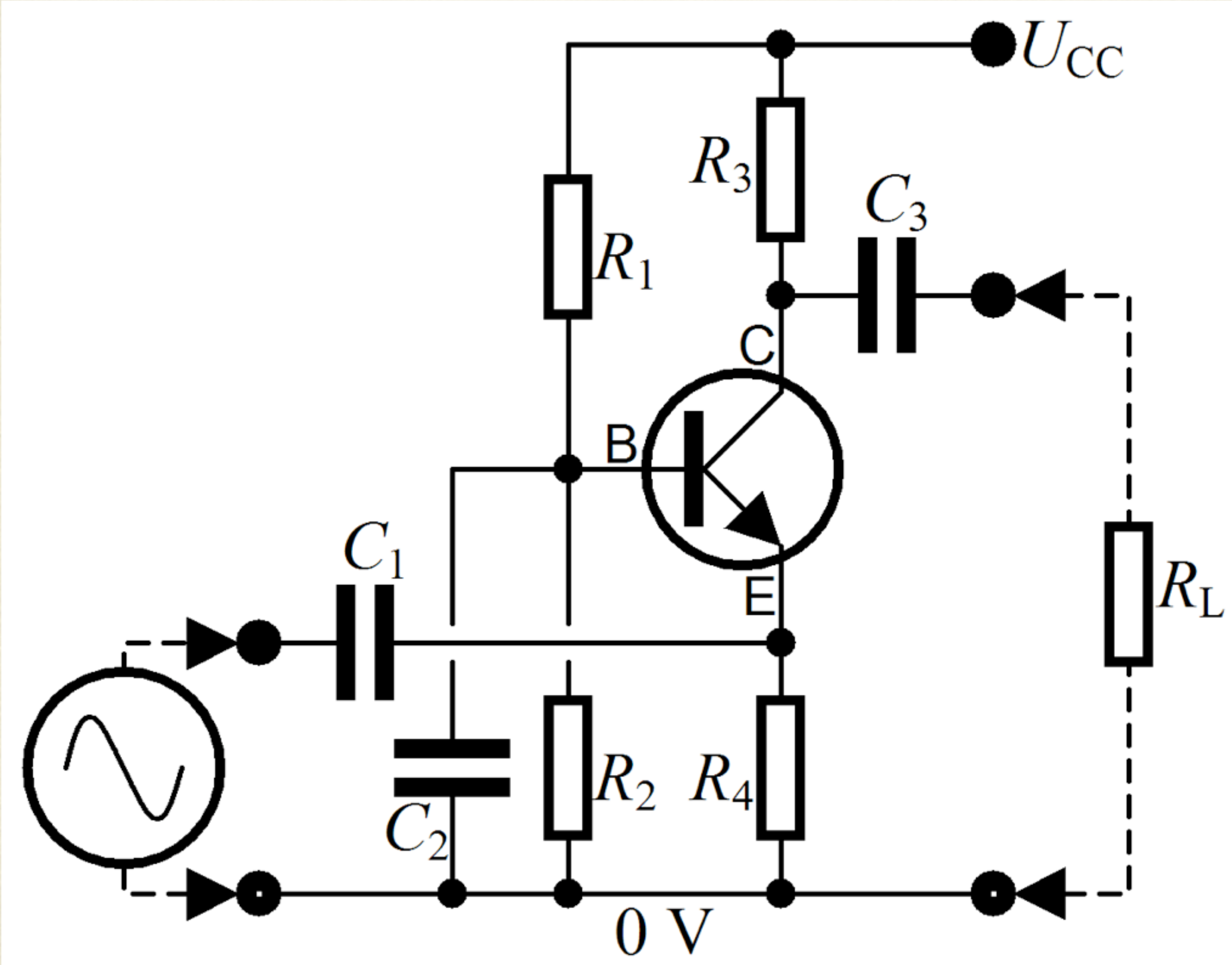
nhbs, Identifying Wood: Accurate Results with Simple Tools.

SL is GPL

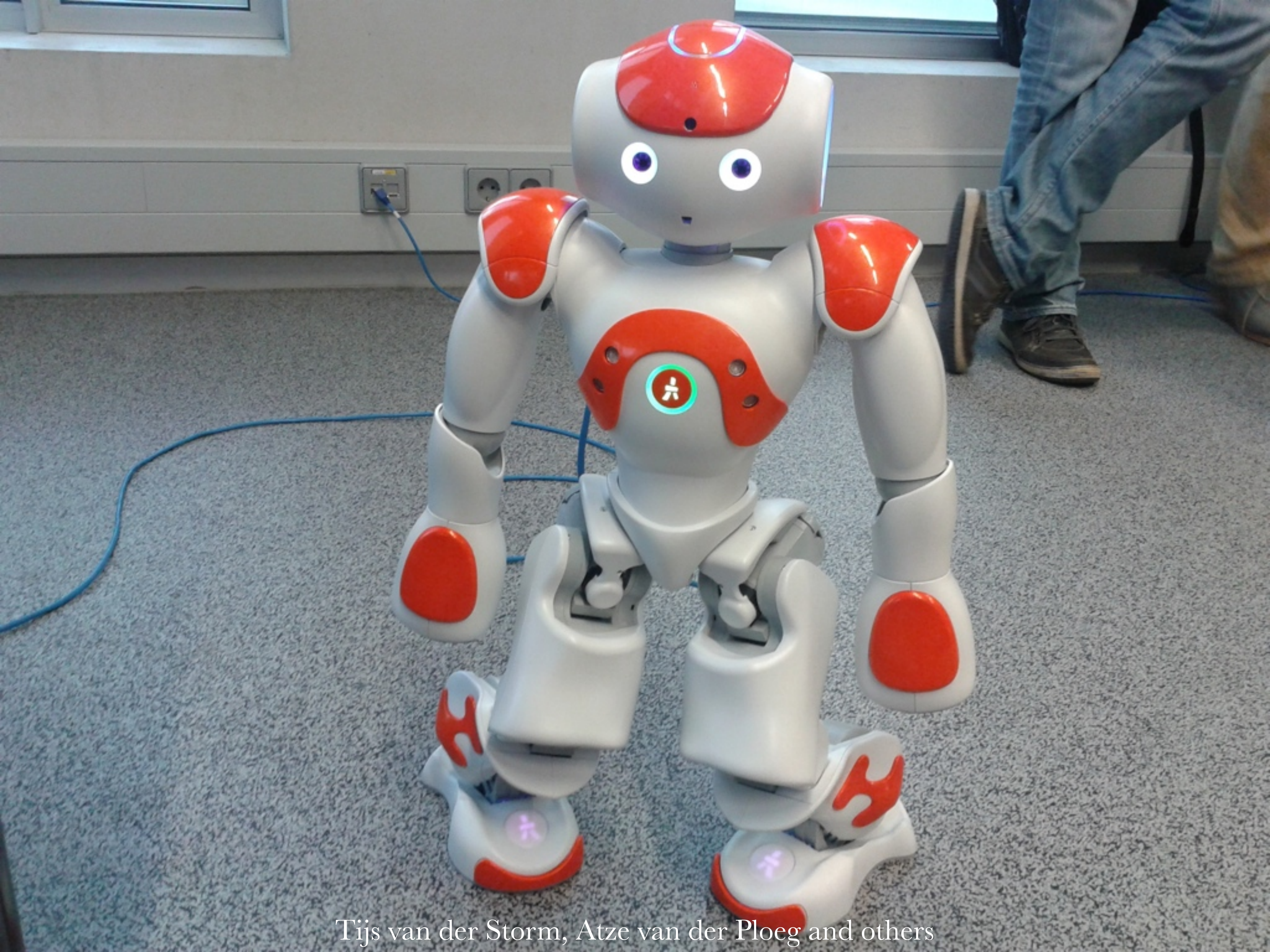
- Programming language
- General purpose
- High level constructs
- Syntactic sugar
- Can be compiled/interpreted/computed/evaluated/…



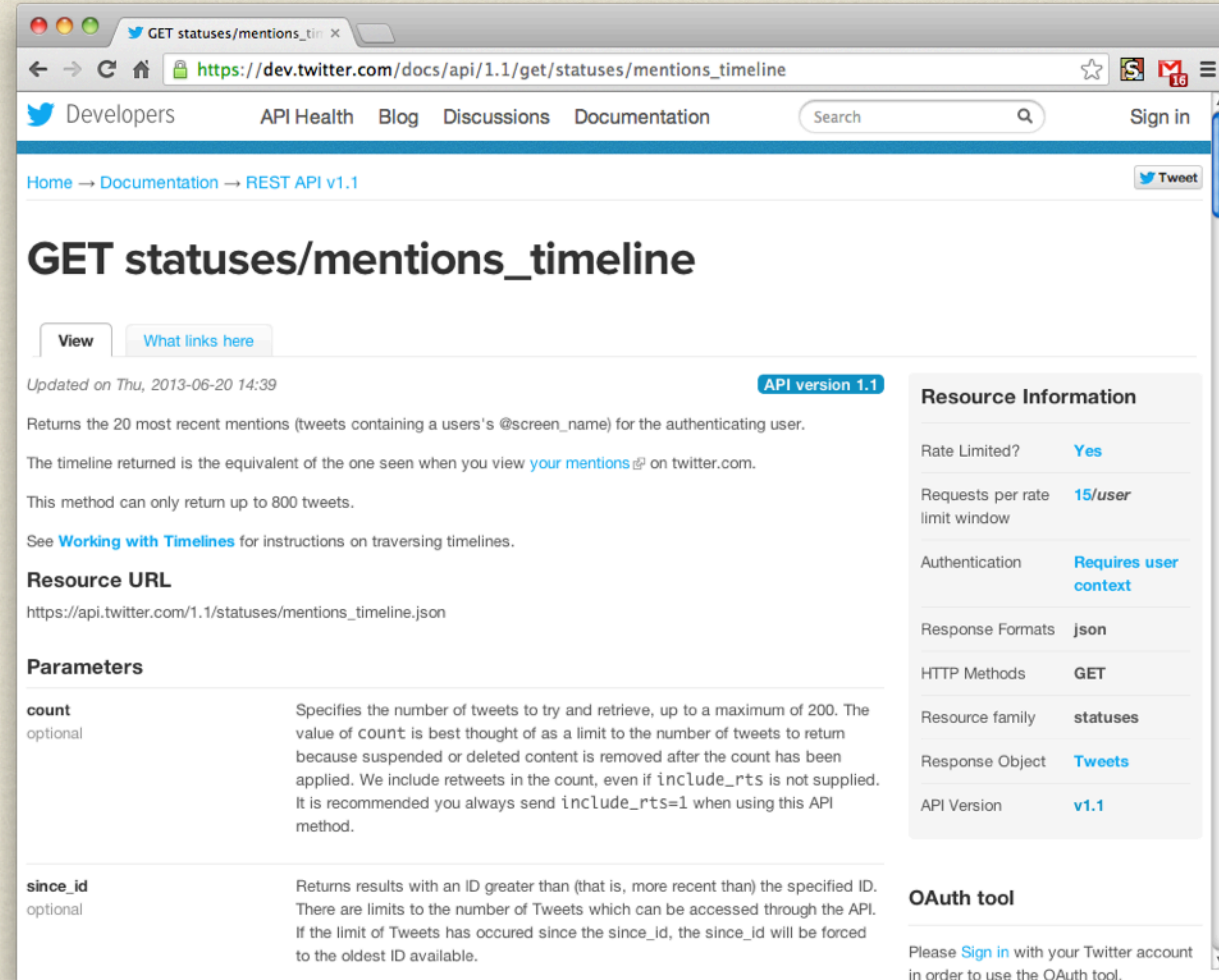
Hackers & Designers



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



Tijs van der Storm, Atze van der Ploeg and others



GET statuses/mentions_tin ×

https://dev.twitter.com/docs/api/1.1/get/statuses/mentions_timeline

Developers API Health Blog Discussions Documentation Search Sign in

Home → Documentation → REST API v1.1

Tweet

GET statuses/mentions_timeline

View | What links here

Updated on Thu, 2013-06-20 14:39

API version 1.1

Returns the 20 most recent mentions (tweets containing a users's @screen_name) for the authenticating user.

The timeline returned is the equivalent of the one seen when you view your mentions on twitter.com.

This method can only return up to 800 tweets.

See **Working with Timelines** for instructions on traversing timelines.

Resource URL

https://api.twitter.com/1.1/statuses/mentions_timeline.json

Parameters

count optional	Specifies the number of tweets to try and retrieve, up to a maximum of 200. The value of `count` is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the count has been applied. We include retweets in the count, even if `include_rts` is not supplied. It is recommended you always send `include_rts=1` when using this API method.
since_id optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.

Resource Information

Rate Limited?	Yes
Requests per rate limit window	15/*user*
Authentication	Requires user context
Response Formats	json
HTTP Methods	GET
Resource family	statuses
Response Object	Tweets
API Version	v1.1

OAuth tool

Please Sign in with your Twitter account in order to use the OAuth tool.

https://dev.twitter.com/docs/api/1.1/get/statuses/mentions_timeline

OHM 2013: Observe. Hack. Make. Dany Qumsiyeh.

B10 | f_x | 0.1156

	A	B	C	D	E	F
7						
8	**Variance/Covariance Matrix:**					
9		***A***	***B***	***C***	***D***	
10	***A***	0.1156	0.0320	0.0384	0.0448	
11	***B***	0.0320	0.1300	0.0480	0.0560	
12	***C***	0.0384	0.0480	0.1476	0.0672	
13	***D***	0.0448	0.0560	0.0672	0.1684	
14						

Clone

Clone cluster

D25 | f_x | =B4*D$20*$B25

	A	B	C	D	E	F	G	H
22								
23				***Covariance Matrix***				
24		***stdev***		**1**	**2**	**3**	**4**	
25		0.34	***1***	0.1156	0.032	0.0384	0.0448	
26		0.3606	***2***	0.032	0.13	0.048	0.056	
27		0.3842	***3***	0.0384	0.048	0.1476	0.0672	
28		0.4104	***4***	0.0448	0.056	0.0672	0.1684	
29								

Hermans, Sedee, Pinzger, van Deursen. Data Clone Detection and Visualization in Spreadsheets. ICSE 2013.

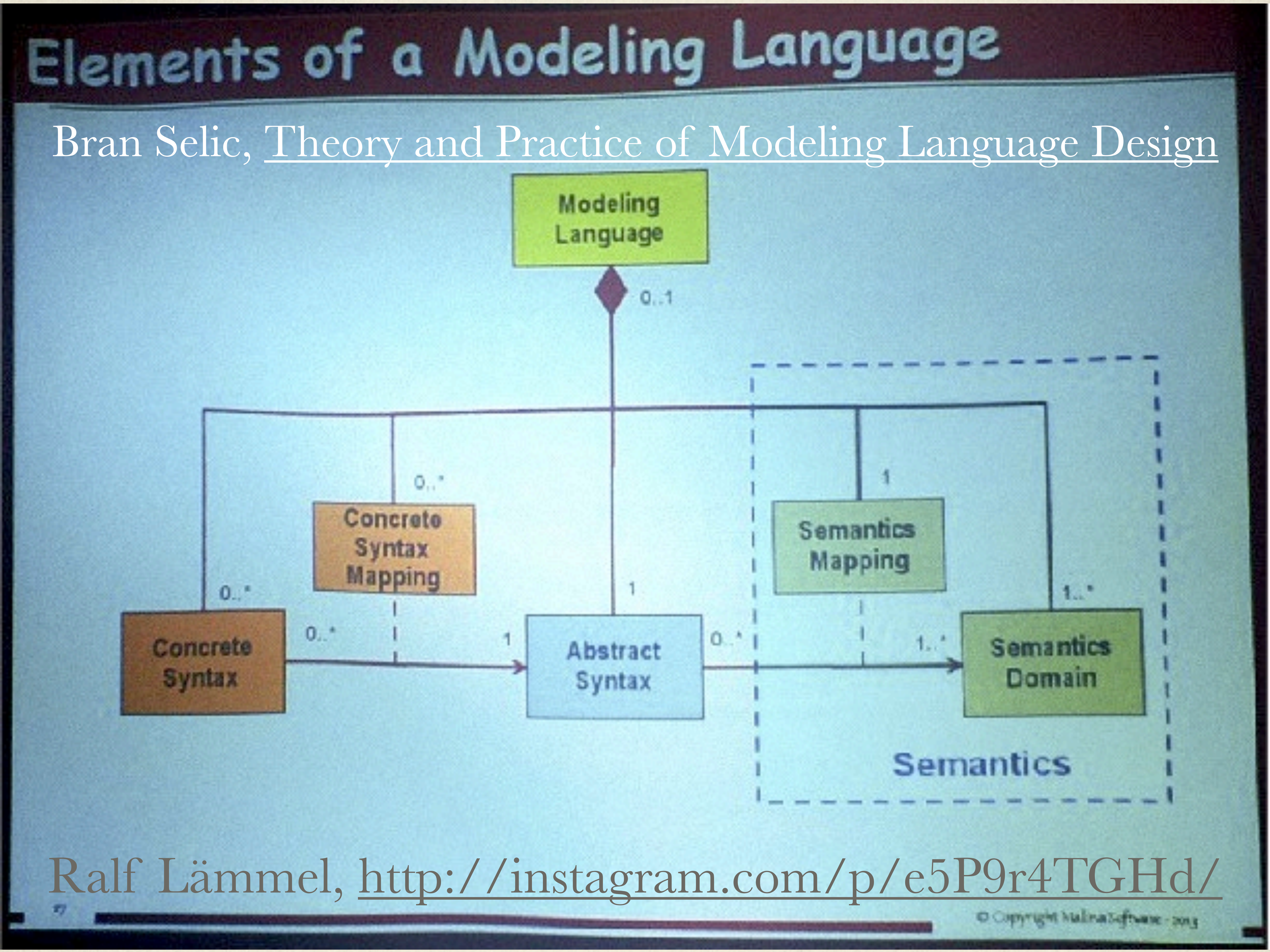
To summarise

- Language is a set of instances
- Language is modelled by a grammar
- Languages are everywhere
- Not necessarily used by experts

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

Abstract
&
concrete

Bran Selic, Theory and Practice of Modeling Language Design



Ralf Lämmel, http://instagram.com/p/e5P9r4TGHd/

Both are grammars!

- Concrete syntax
  - elements for visualisation & programming
- Abstract syntax
  - conceptual structure
- Beyond dychotomy!

DSL concrete syntax in C#

```
public void Configure(Reader target) {
      target.AddStrategy(ConfigureServiceCall());
      target.AddStrategy(ConfigureUsage());
    }
    private ReaderStrategy ConfigureServiceCall() {
      ReaderStrategy result = new ReaderStrategy("SVCL", typeof (ServiceCall));
      result.AddFieldExtractor(4, 18, "CustomerName");
      result.AddFieldExtractor(19, 23, "CustomerID");
      result.AddFieldExtractor(24, 27, "CallTypeCode");
      result.AddFieldExtractor(28, 35, "DateOfCallString");
      return result;
    }
    private ReaderStrategy ConfigureUsage() {
      ReaderStrategy result = new ReaderStrategy("USGE", typeof (Usage));
      result.AddFieldExtractor(4, 8, "CustomerID");
      result.AddFieldExtractor(9, 22, "CustomerName");
      result.AddFieldExtractor(30, 30, "Cycle");
      result.AddFieldExtractor(31, 36, "ReadDate");
      return result;
    }
```

Martin Fowler, Language Workbenches: The Killer-App for Domain Specific Languages? 2005

DSL concrete syntax in XML

```
<ReaderConfiguration>
  <Mapping Code = "SVCL" TargetClass = "dsl.ServiceCall">
    <Field name = "CustomerName" start = "4" end = "18"/>
    <Field name = "CustomerID" start = "19" end = "23"/>
    <Field name = "CallTypeCode" start = "24" end = "27"/>
    <Field name = "DateOfCallString" start = "28" end = "35"/>
  </Mapping>
  <Mapping Code = "USGE" TargetClass = "dsl.Usage">
    <Field name = "CustomerID" start = "4" end = "8"/>
    <Field name = "CustomerName" start = "9" end = "22"/>
    <Field name = "Cycle" start = "30" end = "30"/>
    <Field name = "ReadDate" start = "31" end = "36"/>
  </Mapping>
</ReaderConfiguration>
```

Martin Fowler, Language Workbenches: The Killer-App for Domain Specific Languages? 2005

DSL concrete syntax in text

```
mapping SVCL dsl.ServiceCall
  4-18: CustomerName
  19-23: CustomerID
  24-27 : CallTypeCode
  28-35 : DateOfCallString

mapping  USGE dsl.Usage
  4-8 : CustomerID
  9-22: CustomerName
  30-30: Cycle
  31-36: ReadDate
```

Martin Fowler, Language Workbenches: The Killer-App for Domain Specific Languages? 2005

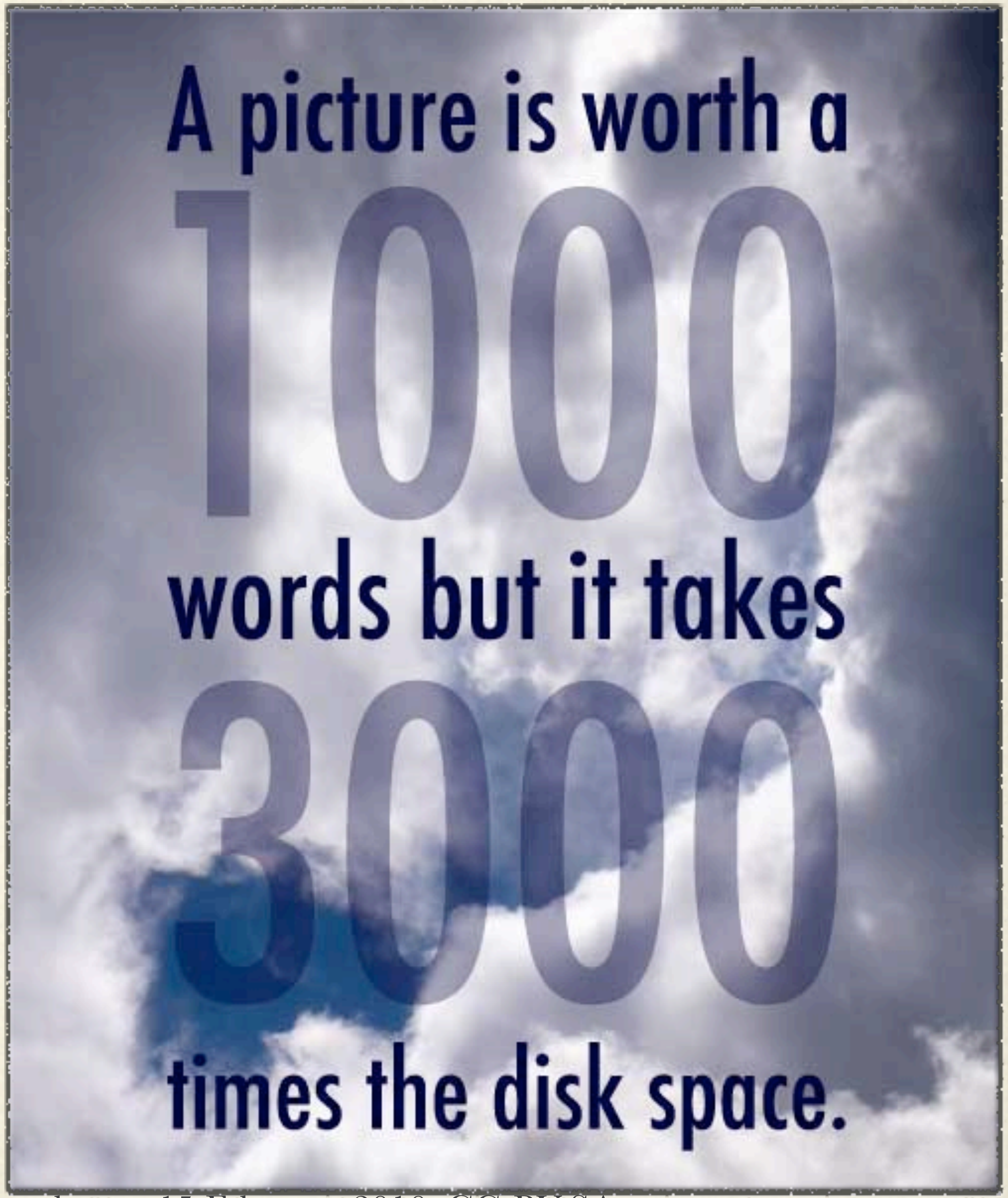
Grammars
on the inside

Backus Naur Form

- Historically useful
- Extended over and over
- “Backus-Wirth Notation”
  - (not a normal form)
-

Backus Naur Form

- Historically useful
- Extended over and over
- "Backus-Wirth Notation"
  - (not a normal form)
- Textual



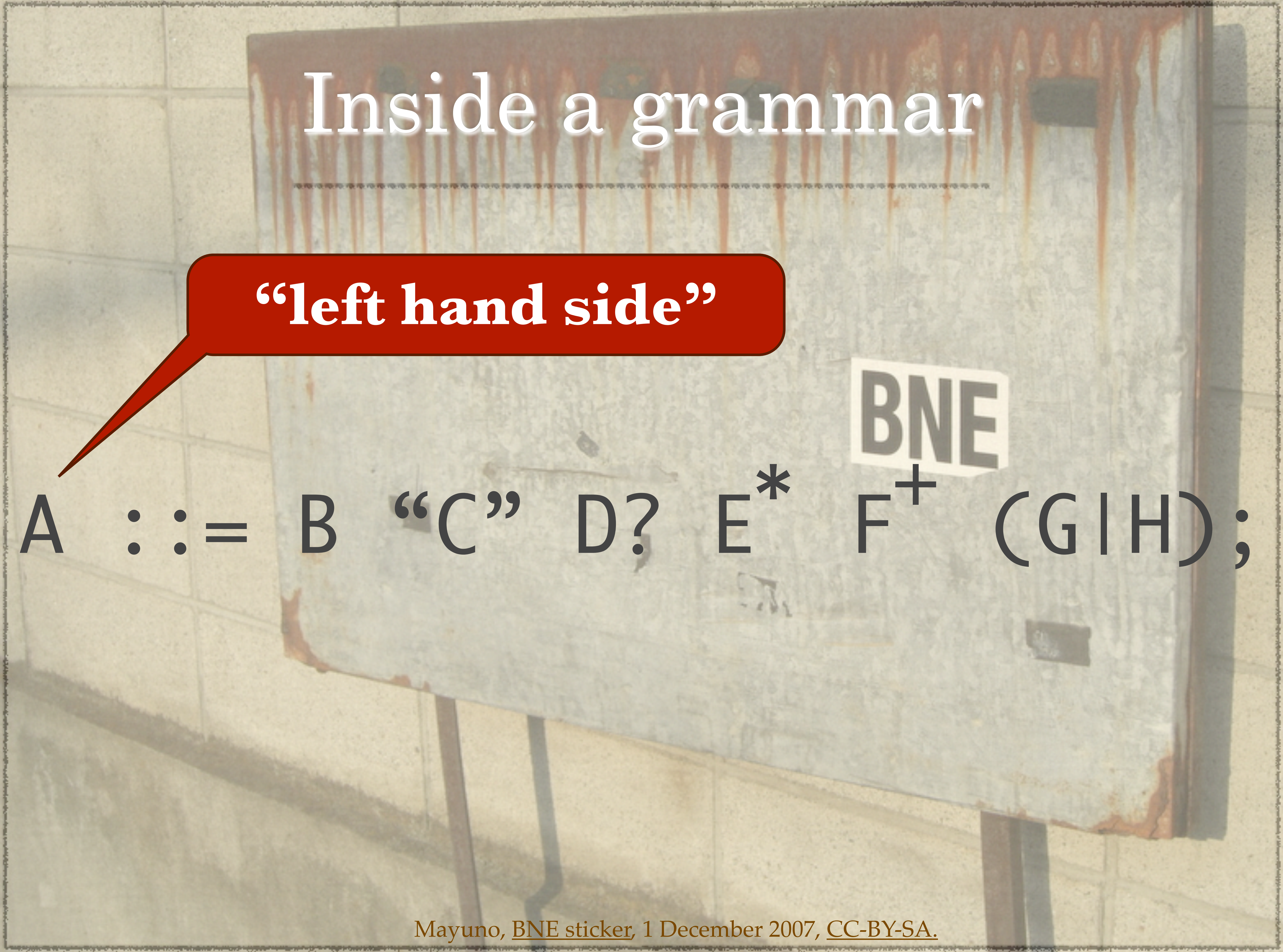
The Next Web Photos, A picture is worth a…, 15 February 2010, CC-BY-SA.

Inside a grammar

production rule

```
A ::= B "C" D? E* F+ (G|H);
```

Inside a grammar



“left hand side”

```
A ::= B “C” D? E* F+ (G|H);
```

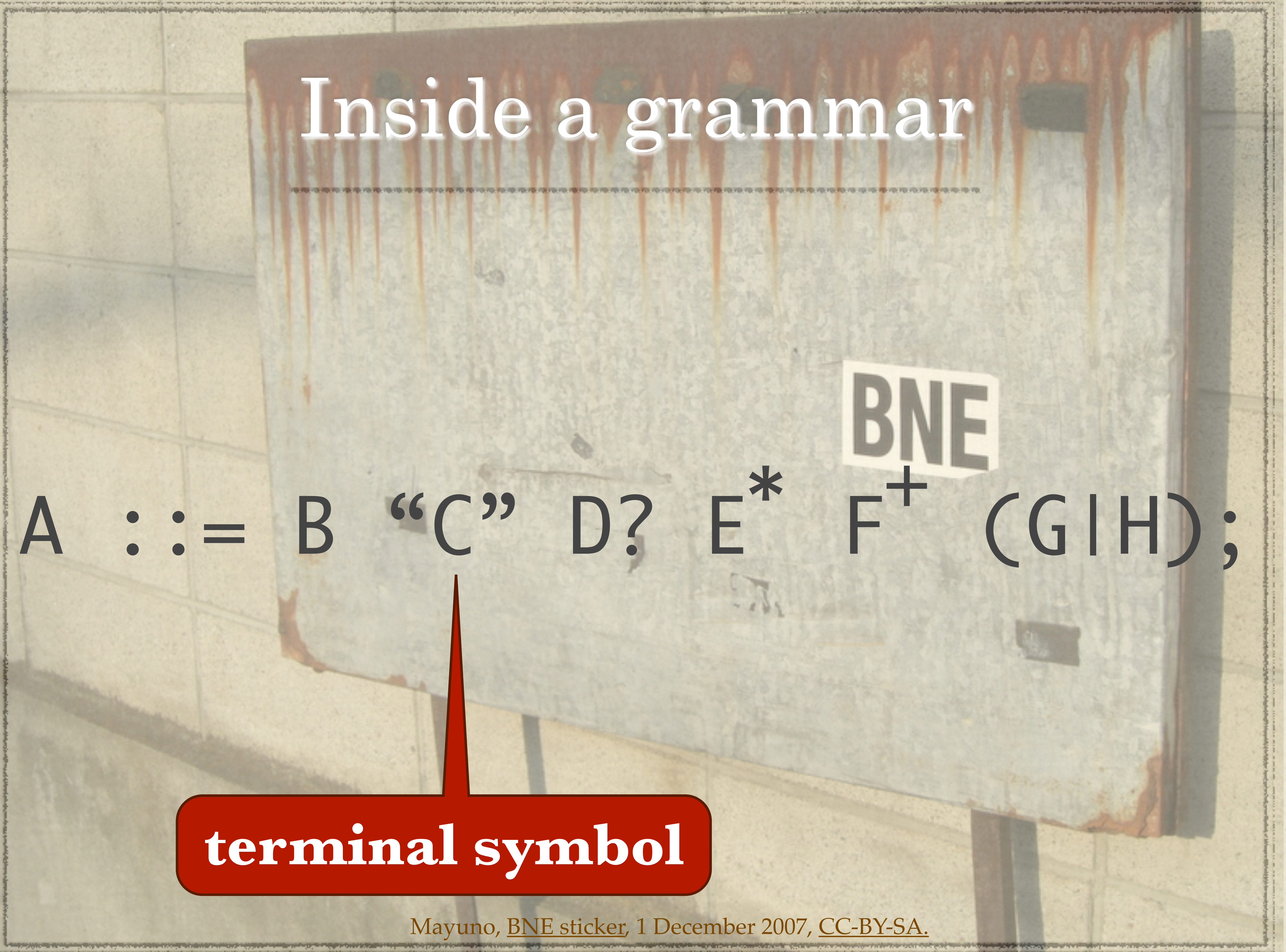
Mayuno, BNE sticker, 1 December 2007, CC-BY-SA.

Inside a grammar

```
A ::= B “C” D? E* F+ (G|H);
```

nonterminal symbol

Inside a grammar



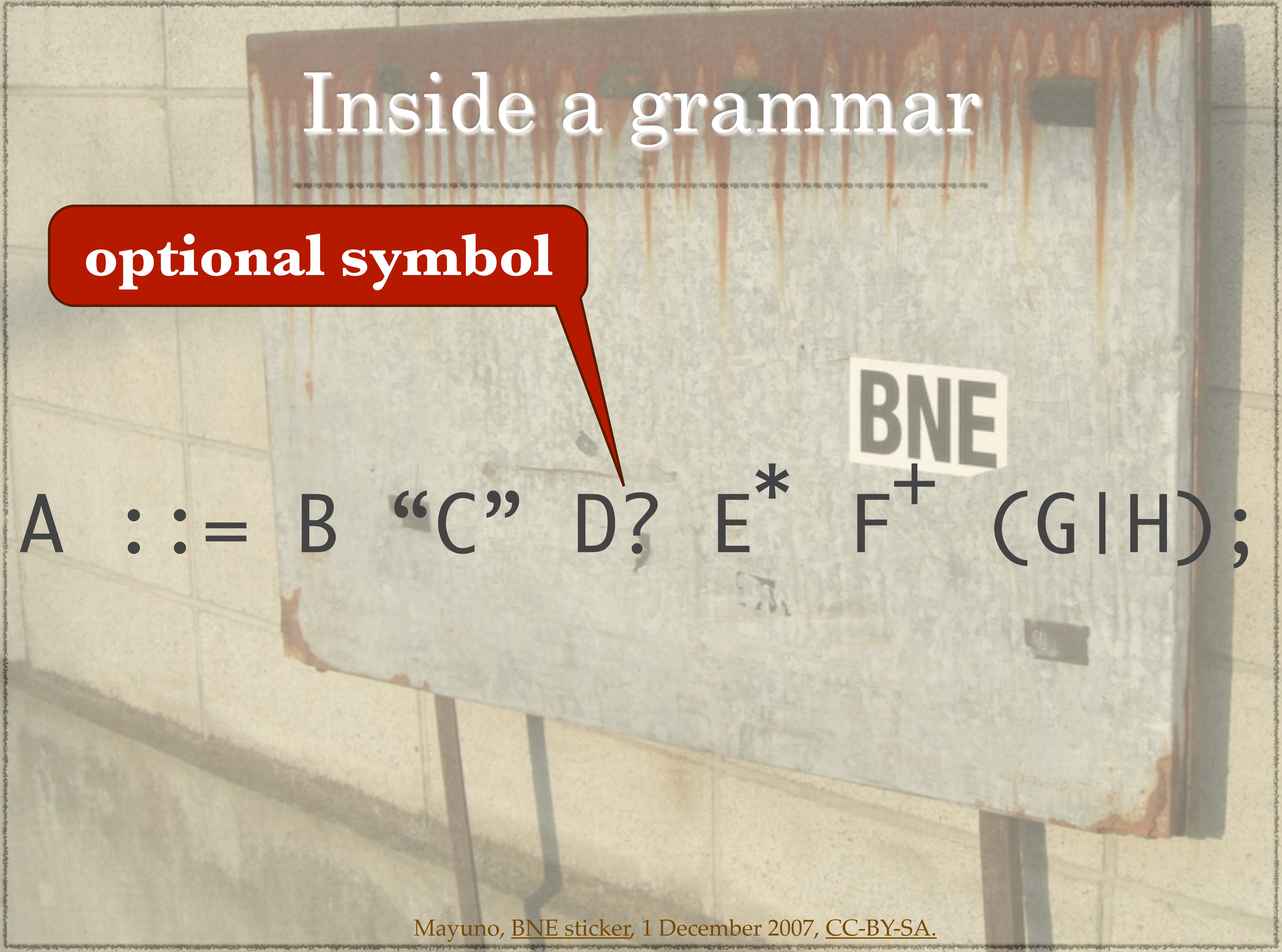
A ::= B "C" D? E* F+ (G|H);

terminal symbol

Inside a grammar

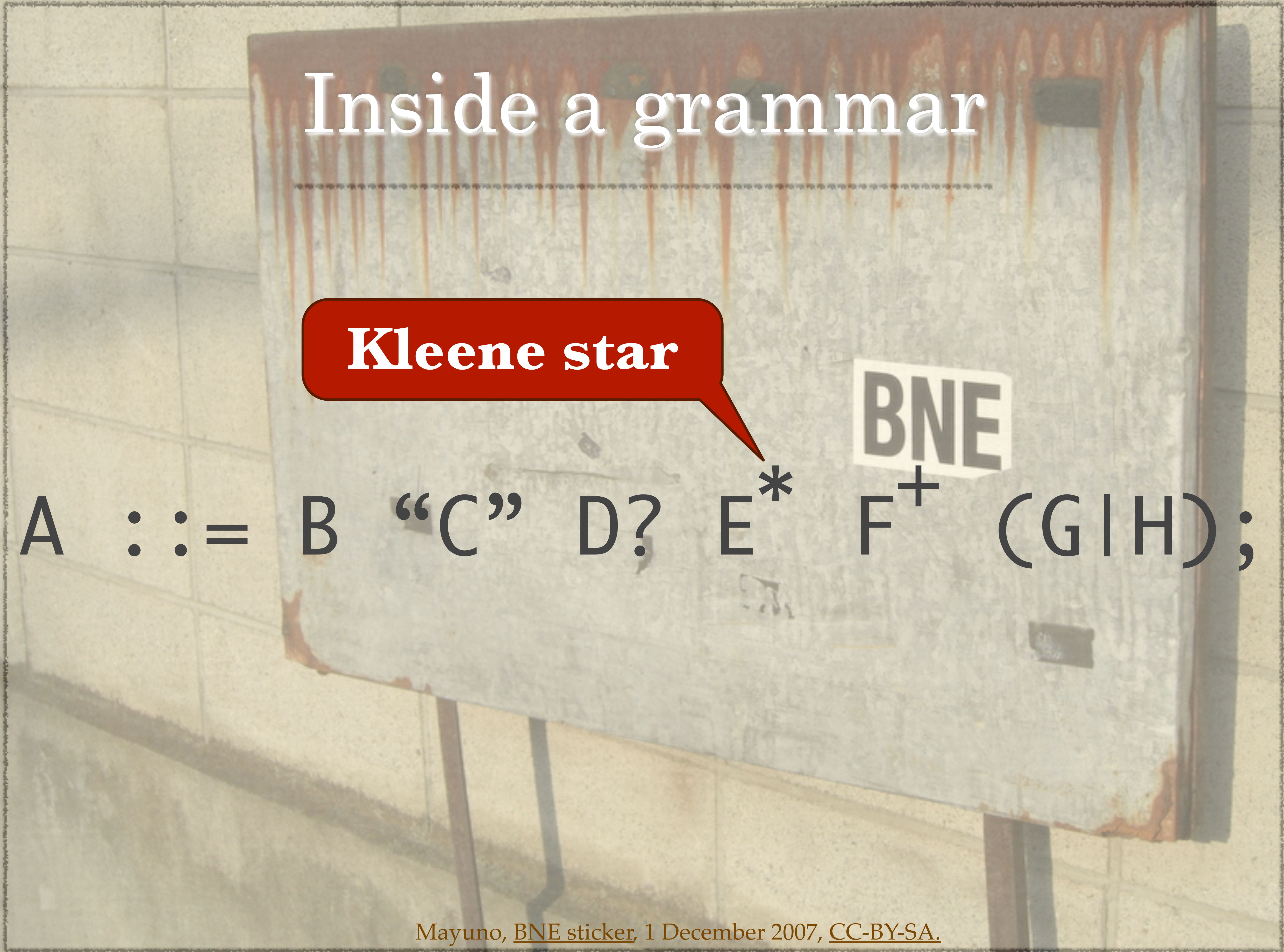
optional symbol

A ::= B “C” D? E* F+ (G|H);



Mayuno, BNE sticker, 1 December 2007, CC-BY-SA.

Inside a grammar



Kleene star

```
A ::= B "C" D? E* F+ (G|H);
```

Mayuno, BNE sticker, 1 December 2007, CC-BY-SA.

Inside a grammar

one or more

$$A ::= B \text{ “C” } D? \; E^{*} \; F^{+} \; (G|H);$$

Inside a grammar

disjunction

```
A ::= B “C” D? E* F+ (G|H);
```

Inside a grammar

markers

```
X ::= <a>:b [c]::d ;
```

labels

Inside a grammar

X ::= ε ;

X ::= φ ;

X ::= α ;

nothing

Inside a grammar

X ::= ε ;

X ::= φ ;

X ::= α ;

error

Inside a grammar

$X ::= \varepsilon\ ;$

$X ::= \varphi\ ;$

$X ::= \alpha\ ;$

anything

Inside a grammar

```
X ::= { Y Z }+ ;

X ::= A & B ;

X ::= ¬N ;
```

Y
Y Z Y
Y Z Y Z Y
...

Inside a grammar

```
X ::= { Y Z }+ ;

X ::= A & B ;

X ::= ¬N ;
```

conjunction

Inside a grammar

```
X ::= { Y Z }+ ;

X ::= A & B ;

X ::= ¬N ;
```

negation

Ada 95 (Magnus Kempe, HTML)

```
compilation:
        compilation_unit*

compilation_unit:
        context_clause library_item
        context_clause subunit

library_item:
        "private"? library_unit_declaration
        library_unit_body
        "private"? library_unit_renaming_declaration

library_unit_declaration:
        subprogram_declaration
        package_declaration
        generic_declaration
        generic_instantiation

library_unit_renaming_declaration:
        package_renaming_declaration
        generic_renaming_declaration
        subprogram_renaming_declaration
```

http://slps.github.io/zoo/ada/kempe.html

BibTeX (Guillaume Hillairet, Ecore)

```
Bibtex:
        entries::Entry+
```

```
Entry:
        Article
```

```
Entry:
        Book
```

```
Article:
        key::String fields::Field+
```

```
Book:
        key::String fields::Field+
```

```
Field:
        Authors
```

```
Field:
        AuthorUrls
```

```
Field:
        Title
```

http://slps.github.io/zoo/bibtex/bibtex-1.html

ANSI C90 (Vinju, Kooiker, van den Brand, SDF)

```
TranslationUnit:
        ExternalDeclaration+

ExternalDeclaration:
        FunctionDefinition

ExternalDeclaration:
        Declaration

FunctionDefinition:
        Specifier* Declarator Declaration* "{" Declaration* Statement* "}"

Declaration:
        Specifier+ (InitDeclarator ("," InitDeclarator)*) ";"

Declaration:
        Specifier+ ";"

InitDeclarator:
        Declarator

InitDeclarator:
        Declarator "=" Initializer
```

http://slps.github.io/zoo/c/ansi-sdf.html

GNU C grammar (TXL, Andrew J. Malton, James R. Cordy)

```
preprocessor:
        "#define" id "(" id+ ")" expression NL
        "#define" id expression NL
       EX "#else" (IN NL)
       EX "#endif" (NL NL)
       NL "#if" expression (IN NL)
       NL "#ifdef" id (IN NL)
       NL "#ifndef" id (IN NL)
        "#ident" stringlit NL
        "#include" stringlit NL
        "#include" "<" SPOFF filepath ">" SPON NL
        "#line" integernumber stringlit? NL
        "#undef" id NL
        "#LINK" stringlit NL
```

```
filepath:
       file_id slash_fileid*
```

```
file_id:
       id
       key
```

http://slps.github.io/zoo/c/cordy-malton.html

OASIS DocBook (RELAX-NG)

```
bookcomponent.content:
        divcomponent.mix+ (sect1* | refentry.class* | simplesect* | section.class*)
        sect1+
        refentry.class+
        simplesect+
        section.class+

local.set.attrib:
        ε

set.role.attrib:
        role.attrib

set:
        set::(set.attlist div.title.content? setinfo? toc? book.class+ setindex?)

set.attlist:
        fpi::string? status.attrib common.attrib set.role.attrib local.set.attrib

local.setinfo.attrib:
        ε
```

http://slps.github.io/zoo/markup/docbook-walsh.html

To summarise

- Nonterminals
- Terminals
- Metaproperties (?,*, +)
- Special cases
- Labels and markers
- Disjunction
- …

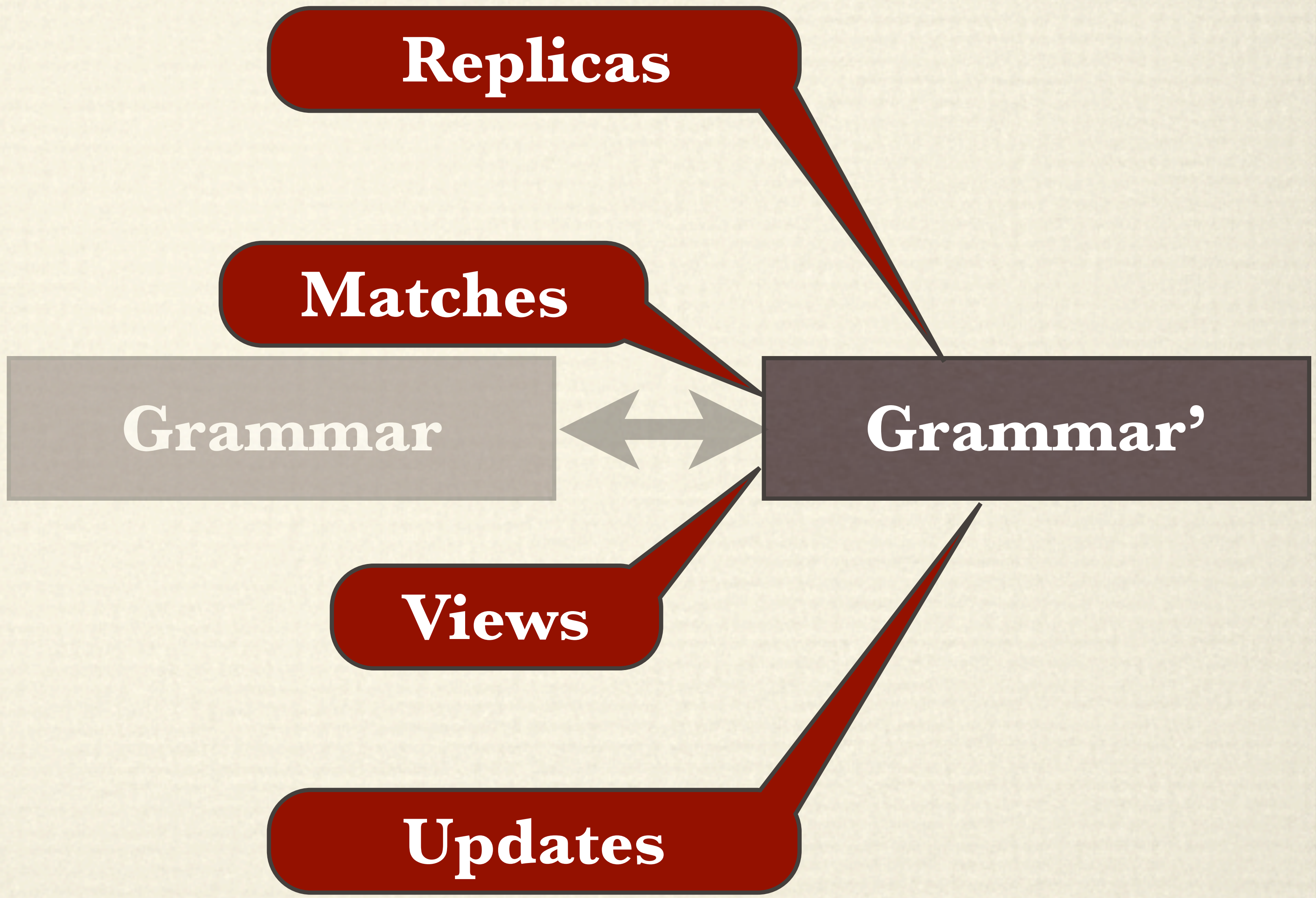
http://commons.wikimedia.org/wiki/File:Torii_kiyoshige_bando_hikosaburo_ii.jpg

Grammars
from the outside

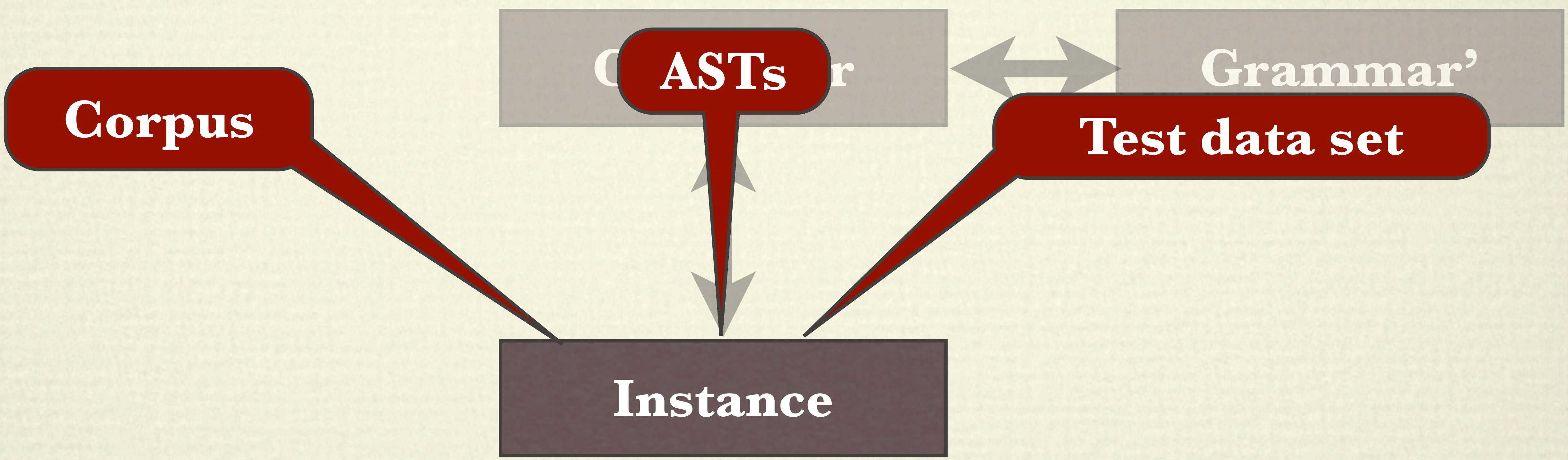
Grammar world

Grammar

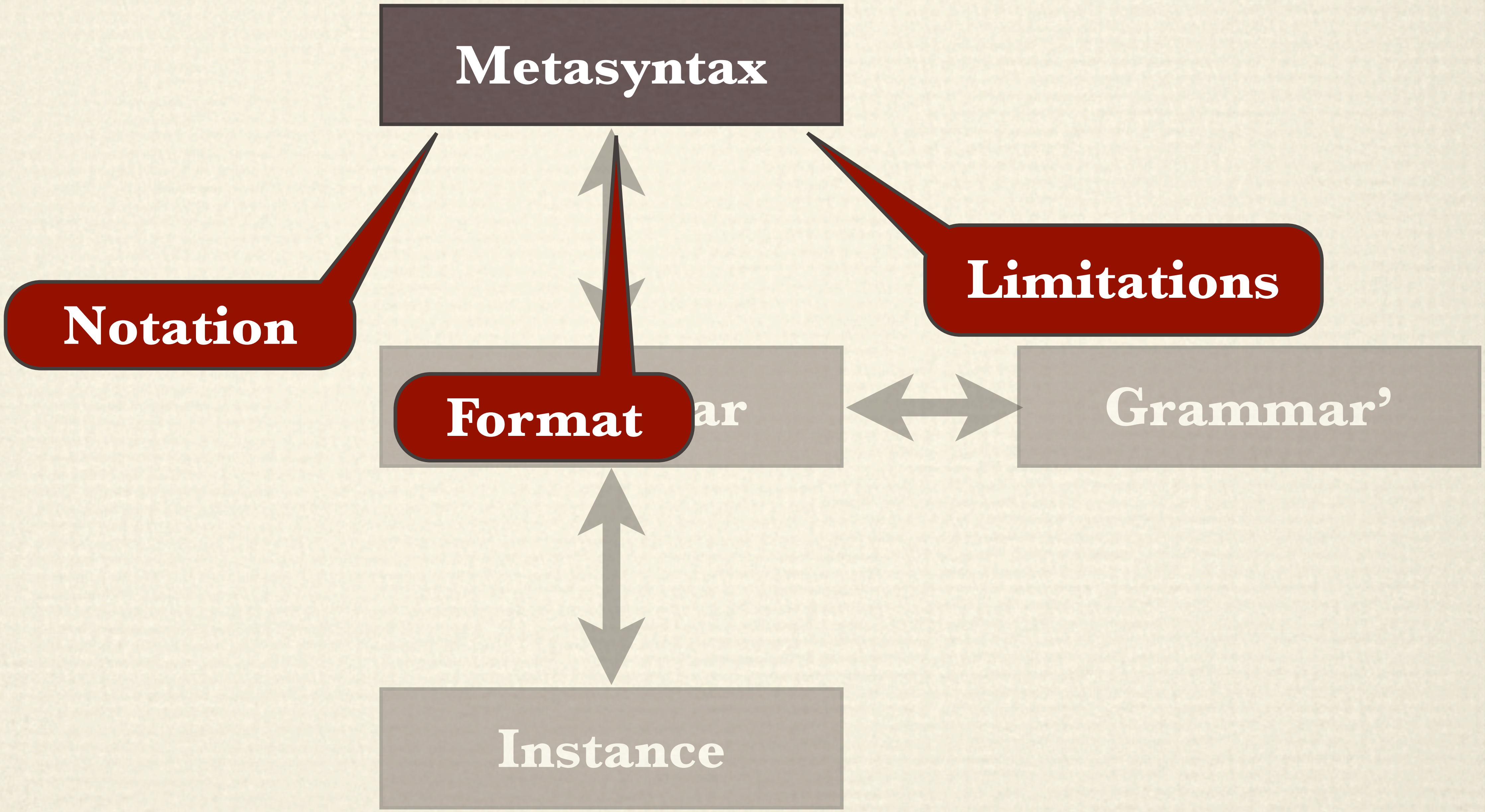
Grammar world: grammars



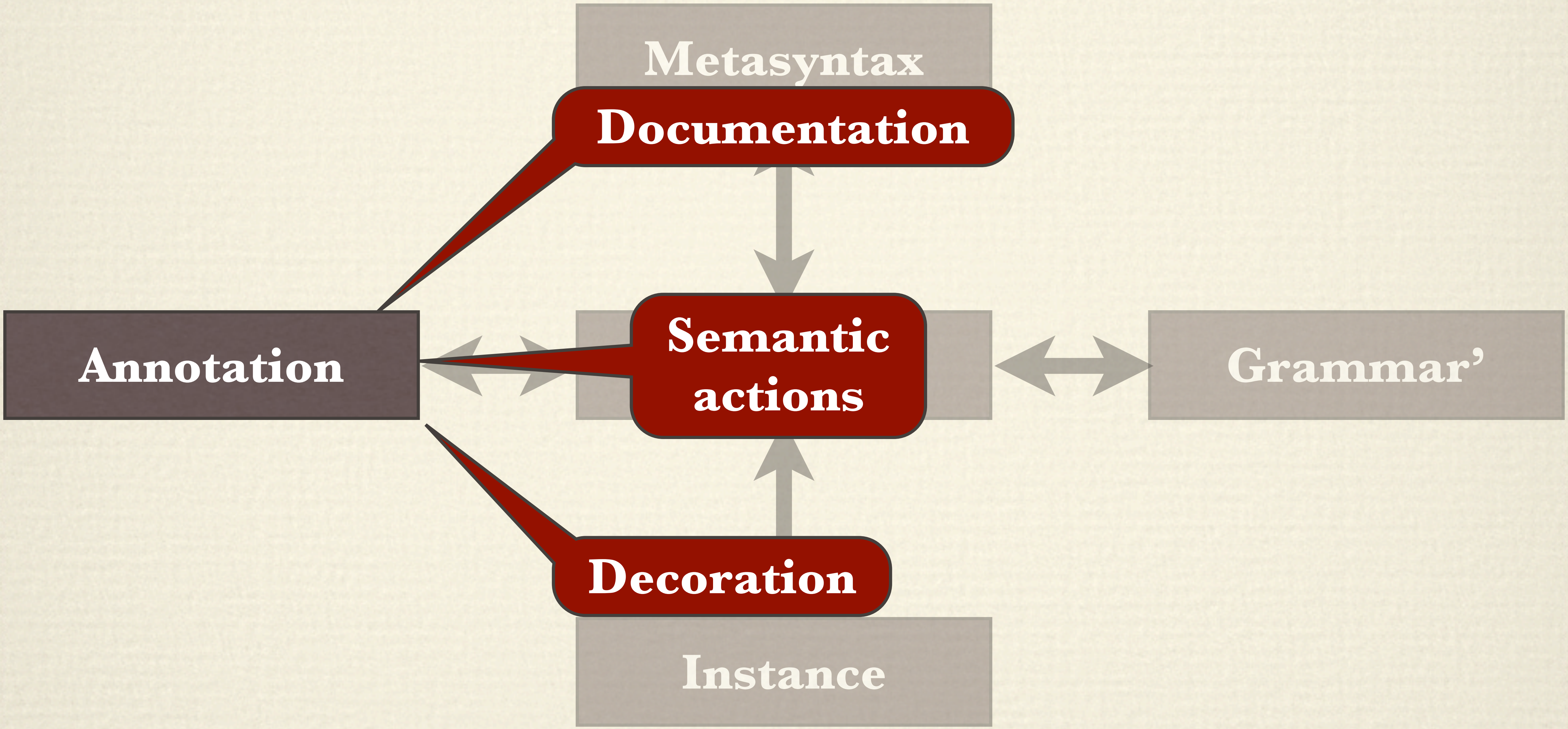
Grammar world: instances



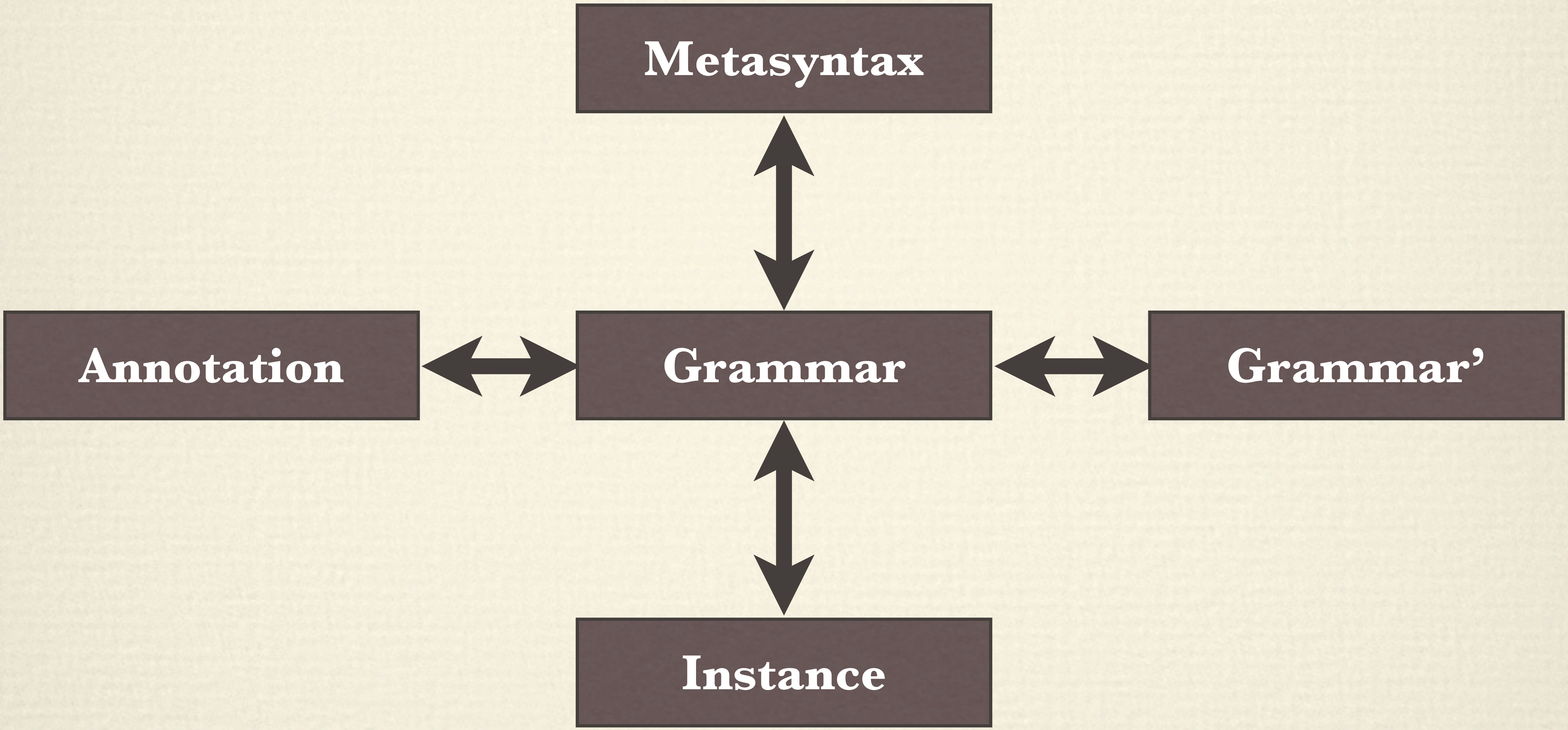
Grammar world: metamodels



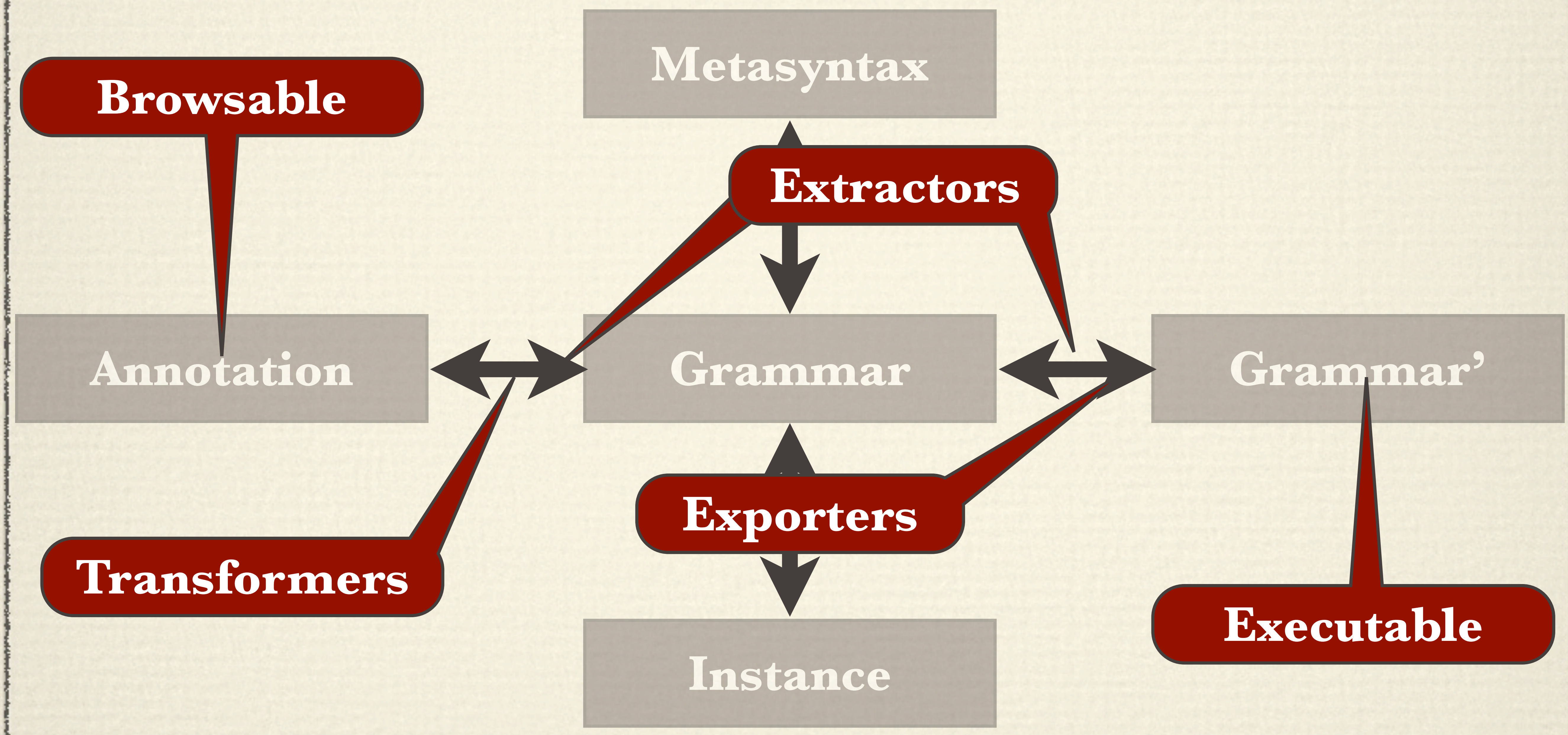
Grammar world: annotations



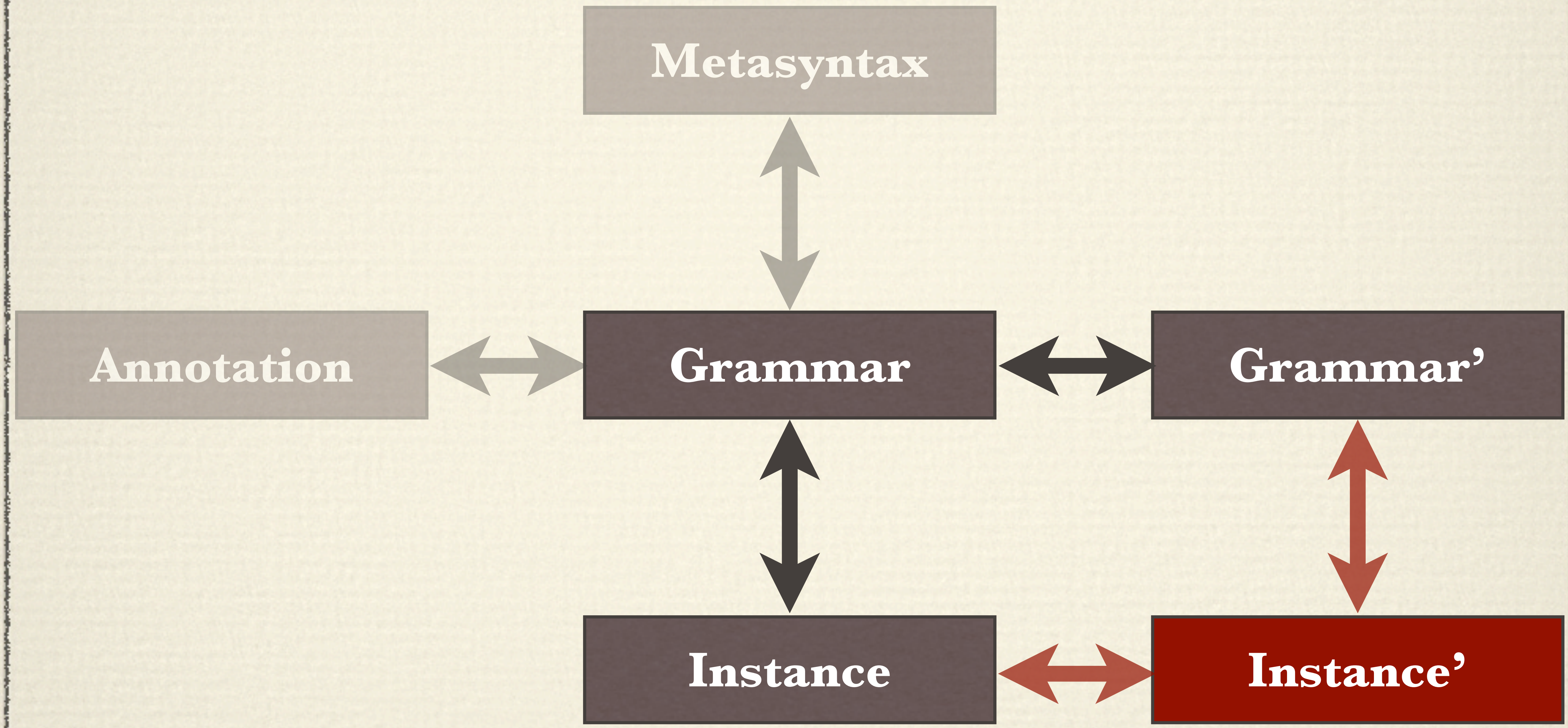
Grammar world



Grammar world: transformations



Grammar world: synchronisations



To summarise

- Diversity
- Evolution
- Extraction
- Documentation

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

Grammarware

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 eng. tool
- Benchmark
- Recommender
- Renovation tool

"Creative Space"

Parser [Aho et al]

- Syntactic analyser
- Traverses text
- Validates conformance to grammar
- Recognises language instances ("words")
- Constructs a parse tree

A. V. Aho, R. Sethi, J. D. Ullman, *Compilers*, 1985.

Parser [Lesk/Schmidt, Johnson]

- Two separate phases
  - lexical analysis (lexer)
  - syntactic analysis (parser)
- Write your grammar in a very specific style
- The parser is generated automatically
  - ...and maybe will work
- Want a good parser?
  - forget about readability and reuse

Lesk, Schmidt, *Lex – A Lexical Analyzer Generator*, 1975.
Johnson, *YACC — Yet Another Compiler Compiler*, 1975.

Parser [Cook et al]

- Recognises syntactic features
- Creates object structure
- Results in a graph
  - object grammars allow crosslinks
- Naturally bidirectional (parsing/rendering)
  - object grammars contain pretty-printing info

van der Storm, Cook, Loh, Object Grammars: *Compositional & Bidirectional Mapping Between Text and Graphs*, SLE 2012.

Parser [Meyers/Vangheluwe]

- A mapping between
  - one of concrete syntaxes
  - the abstract syntax
- Assumes the “spanning tree” is available or easy

Meyers, Vangheluwe, *A Framework for Evolution of Modelling Languages*, SCP, 2011.

Parser [Ophel]

- Parser has three tasks
  - verify that the input is valid
  - provide semantical framework
  - if invalid, explain why

Ophel, *Breadth-First Parsing*, 1997.

To summarise

- Disregard textual parsing?
- Separate lexical from syntactic?
- Link to semantics?

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

Syntactic Structures (1957)

Despite the undeniable interest and importance of *semantic* and statistical studies of language, they appear to have *no direct relevance* to the problem of determining or characterizing the set of grammatical utterances. I think that we are forced to conclude that grammar is *autonomous* and independent of meaning.

Noam Chomsky


Search
Andreas Wortmann
@andwor
Follow
Semantics is a social contract #mpm13 #models2013
Reply
Retweeted
Favorite
More
2
RETWEETS
3:44 PM - 30 Sep 13
Reply to @andwor
© 2013 Twitter
About
Help
Ads

Grammars define structure & can assume different semantics

http://instagram.com/p/e0pF3HlDFb/

rASCAL

Language Workbench

Joint work of:

Ali Afroozeh, Jeroen van den Bos, Magiel Bruntink, Jan van Eijck, Paul Griffioen, Mark Hills, Anastasia Izmaylova, Paul Klint, Wouter Kwakernaak, Dimitrios Kyritsis, Mattijn Lahuis, Davy Landman, Robert van Liere, Bert Lisser, George Marmanidis, Chris Mulder, Atze van der Ploeg, Riemer van Rozen, Alexander Serebrenik, Ashim Shahi, Sunil Simon, Michael Steindorfer, Tijs van der Storm, Ioannis Tzanellis, Jurgen Vinju, Kevin van der Vlist, Vadim Zaytsev (and others)

Language workbench

- Language oriented programming
  - Little languages: grep, sed, awk
  - Homoiconic languages: LISP, REBOL, XSLT
  - Configuration files
- Other workbenches
  - MPS (Völter et al), Intentional Software (Simonyi et al), Epsilon (Paige et al)

Martin Fowler, Language Workbenches: The Killer-App for Domain Specific Languages? 2005

Rascal MPL – Getting start

www.rascal-mpl.org/start/

Home Download Help Developers Blog About

Rascal Metaprogramming Language

Eclipse update site | Commandline REPL Jar File

Disclaimer

We currently only release alpha versions of Rascal, which are subject to frequent changes

Eclipse plugin

The Eclipse update site for Rascal is: **http://update.rascal-mpl.org/stable/**

You need *Eclipse for RCP and RAP Developers (Juno or Kepler)* version of Eclipse available at www.eclipse.org/downloads/ to run Rascal. It has been reported recently that the latest release also works with normal (non RCP/RAP) versions of Eclipse. Not thoroughly tested though!

Please note that:

- Rascal now needs a JDK because it uses the Java compiler, so please download a JDK, not just a JRE.
- You may have to edit the Eclipse init file so that Eclipse can find your Java installation and Eclipse can allocate enough resources.
- For generating parsers, Rascal uses quite a bit of memory. Please use `-vmargs -Xmx1G -Xss32m`

Follow these steps to install the plugin into Eclipse

1. Start Eclipse.
2. Select Help -> Install New Software.
3. Make sure that the tick for "Contact all update sites during install to find required software" is enabled.
4. Type: `http://update.rascal-mpl.org/stable/` in the "Work with" edit box.
5. Select the feature *Rascal*.
6. Select Next (several times) and accept the software license. The process may take a few minutes!
7. Once these features have been installed, restart Eclipse.

Technical challenges

- How to parse source code/data files/models?
- How to extract facts from them?
- How to perform computations on these facts?
- How to generate new source code (transform, refactor, compile)?
- How to synthesize other information?

EASY: Extract-Analyze-SYnthesize Paradigm

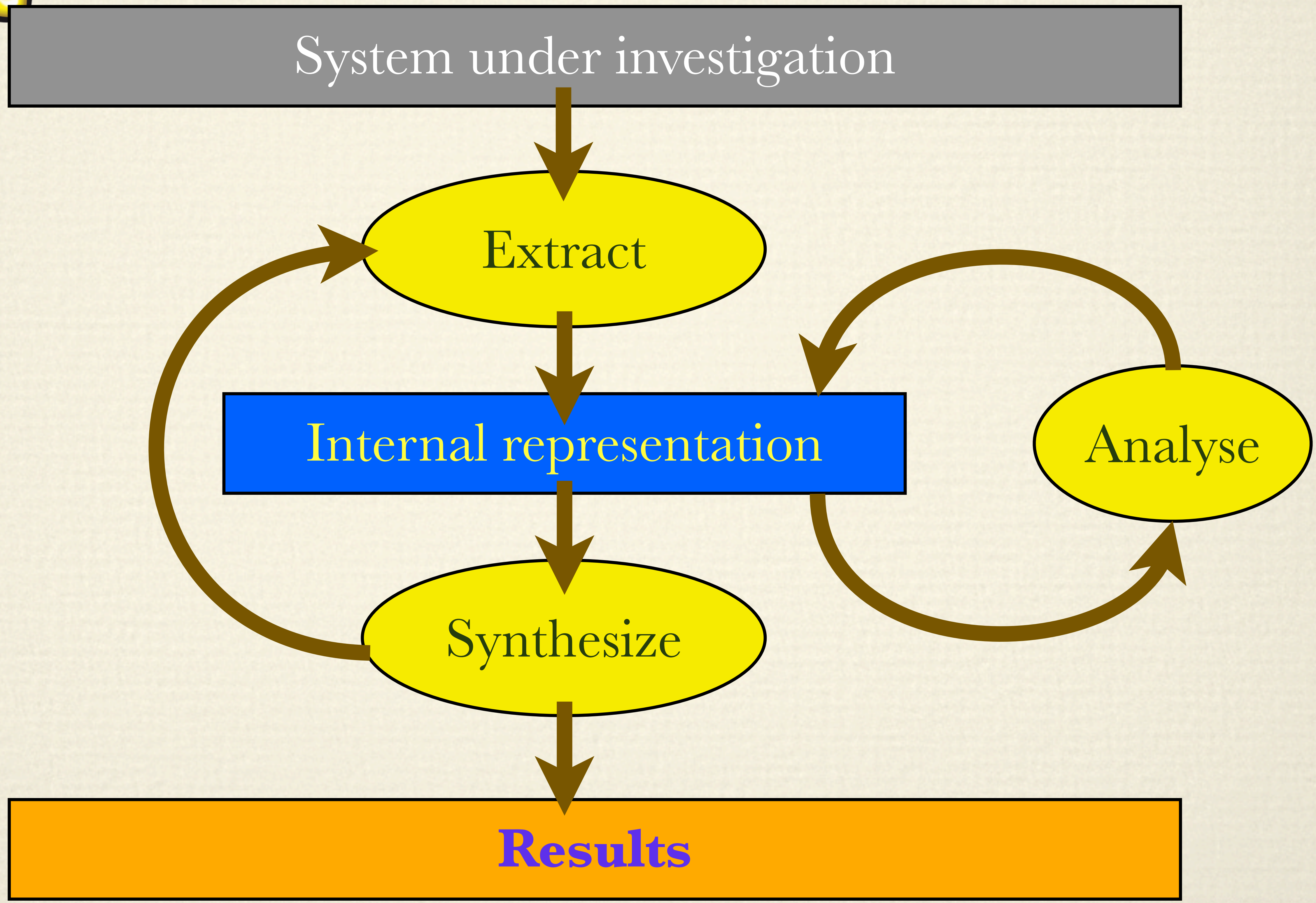
Metaprogramming is EASY

- **Extract**
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- **Analyze**
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...
- **SYnthesize**
  - String templates
  - Concrete syntax
  - Interactive visualization generator

Metaprogramming is EASY

- **<u>E</u>xtract**
  - Fast context-free general top-down parsing
  - Pattern matching & generic traversal
- **<u>A</u>nalyze**
  - Relational queries and comprehensions
  - Backtracking, fixed point computation, ...
- **<u>SY</u>nthesize**
  - String templates
  - Concrete syntax
  - Interactive visualization generator

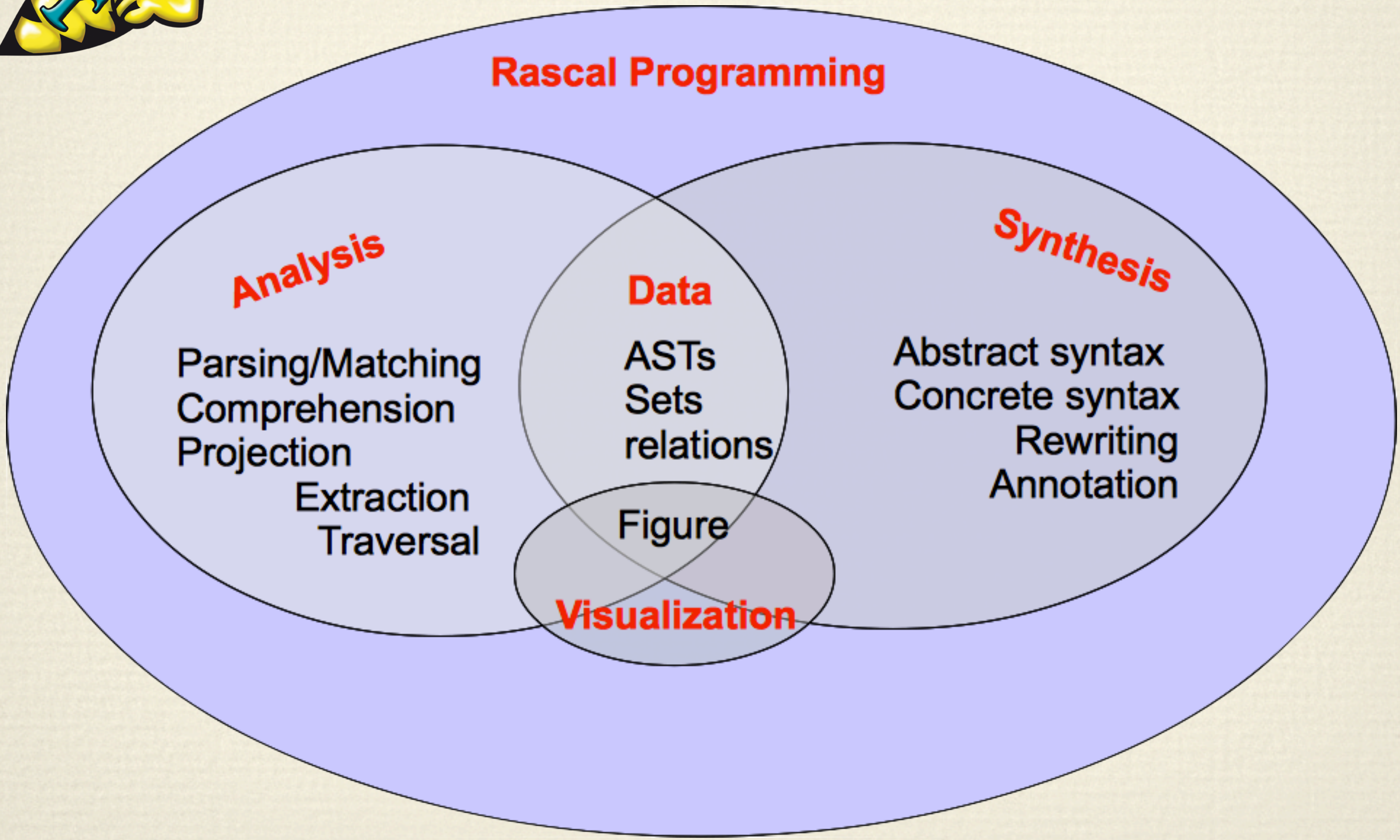
The EASY paradigm



Why a new language?

- No current technology spans the full range of **EASY** steps
- There are many fine technologies but they are
  - highly specialized with steep learning curves
  - hard to learn unintegrated technologies
  - not integrated with a standard IDE
  - hard to extend
- Goal: keep all benefits of ASF+SDF and Rscript
  - in a new, unified, extensible, teachable framework

Bridging the gaps



Rascal keywords

- Complex built-in data types
- Immutable data
- Static safety
- Generic types
- Local type inference
- Pattern matching
- Syntax definitions & parsing
- Concrete syntax
- Visiting/traversal
- Comprehensions
- Higher-order
- Familiar syntax
- Java and Eclipse integration
- Read-Eval-Print (REPL)

Rascal design

- Java-like syntax

presumably familiar

- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

installs as a plugin

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

low barrier to entry,
learn features as you go

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

concrete syntax matching

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting

  traversals, matching, ...

- Relational calculus

Rascal design

- Java-like syntax
- Embedded in Eclipse
- Layered design
- Syntax analysis
- Term rewriting
- Relational calculus

relations for sharing/merging of facts for different languages

To summarise

- Rascal is a language workbench
- EASY paradigm
    - extract
    - analyse
    - synthesise
- Metaprogramming
- http://rascal-mpl.org

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

Rascal features

Rich (immutable) data

- Built-in types:
  - lists
  - sets
  - maps
  - tuples
  - relations
- with comprehensions and many operators

```
rascal> [1..10]
list[int]: [1,2,3,4,5,6,7,8,9,10]
rascal> [x/2 | x <- [1..10]]
list[int]: [0,1,1,2,2,3,3,4,4,5]
rascal> {x/2 | x <- [1..10]} + {4,5,6}
set[int]: {6,5,4,3,2,1,0}
```

Syntax definitions

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/...
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

```
lexical Id = [A-Za-züäöß]+ !>> [A-Za-züäöß];
lexical Num = [0-9]+ !>> [0-9];
```

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/...
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

```
start syntax System = Line+;
syntax Line = Num ":" {Id ","}+ "." ;
```

- Define lexica
- Define context-free syntax
- Define whitespace/layout/...
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

- Define lexical sy

```
layout WS = [\ \t\n\r]* !>> [\ \t\n\r];
```

- Define context-free syntax
- Define whitespace/layout/…
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Syntax definitions

- Define lexical syntax
- Define context-free syntax
- Define whitespace/layout/…
- Get GLL parser for free
- Define an algebraic data type
- Automatically implode parse trees to ASTs

Patterns

- Pattern matching
  - on concrete syntax
  - on lists
  - on sets
  - on trees
  - ...
- Pattern-driven dispatch

```
rascal> {int x, str y} := {2}
bool: false
rascal> {int x, str y} := {2,"3"}
bool: true
rascal> {int x, *y, str z} := {2,2,2,"3",4,"2"}
bool: true
```

Other pattern kinds

- **Regular:** grep/Perl like regular expressions

```
/^<before:\W*><word:\w+><after:.*$>/
```

- **Abstract:** match data types

```
whileStat(Exp, Stats*)
```

- **Concrete:** match parse trees

```
` while <Exp> do <Stats*> od `
```

Pattern-directed invocation

Prolog?

```
bool eqfp(fpnt(), fpnt()) = true;

bool eqfp(fpopt(), fpopt()) = true;

bool eqfp(fpplus(), fpplus()) = true;

bool eqfp(fpstar(), fpstar()) = true;

bool eqfp(fpempty(), fpempty()) = true;

bool eqfp(fpmany(L1), fpmany(L2)) = multiseteq(L1,L2);

default bool eqfp(Footprint pi, Footprint xi) = false;
```

Zaytsev, *Guided Grammar Convergence*. Poster at SLE 2013

Switch/case

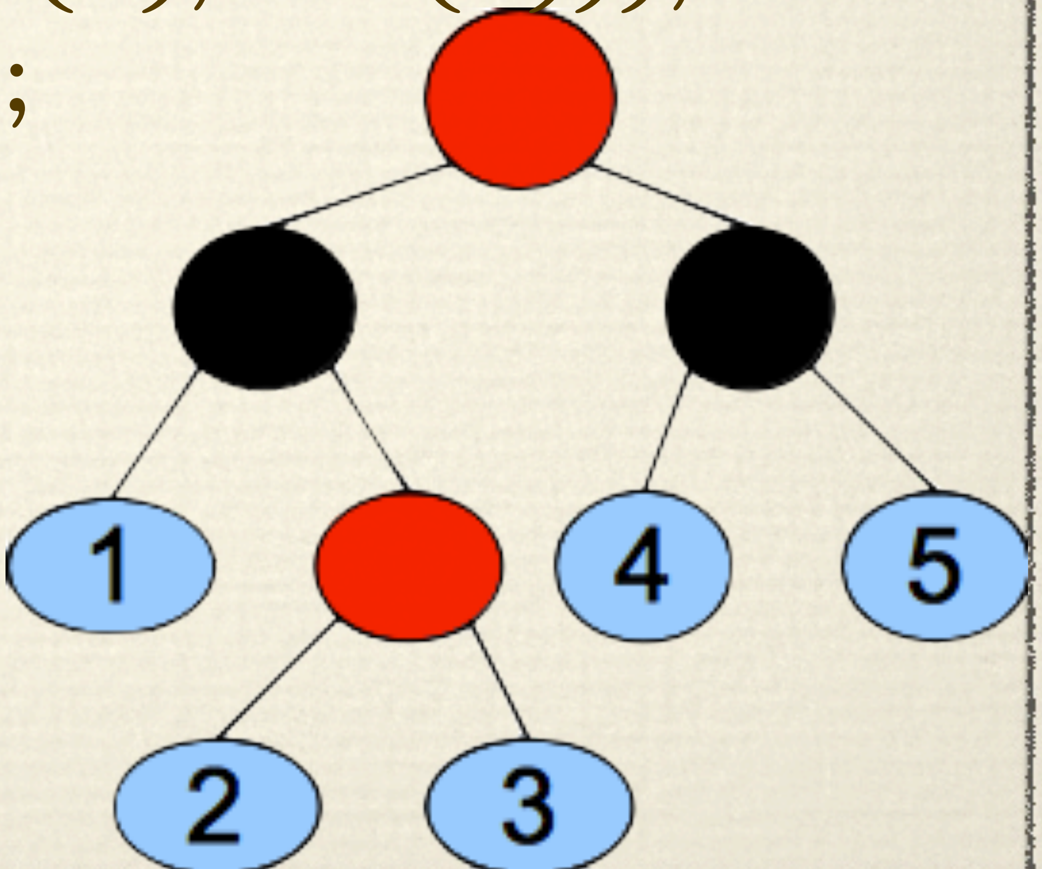
```
switch(p)
{
   case (DCGFun)`[]` => ["ε"];
   case (DCGFun)`<Word n>` =>
      ["<n>" | "<n>"==toLowerCase("<n>")];
   case (DCGFun)`(<{DCGFun ","}* args>)` =>
      [*getTags(a) | a <- args];
   case (DCGFun)`<Word f> (<{DCGFun ","}* as>)` =>
      ["<f>"] + [*getTags(a) | a <- as];
   default …
}
```

Visitor and foldr

```
@contributor{Bas Basten - Bas.Basten@cwi.nl (CWI)}
@contributor{Mark Hills - Mark.Hills@cwi.nl (CWI)}
module Operations

import AST;
import IO;

public Company cut(Company c) {
   return visit (c) {
      case employee(name, [*ep,ip:intProp("salary",salary),*ep2])
                => employee(name, [*ep,ip[intVal=salary/2],*ep2])
}}

public int total(Company c) {
return (0 | it+salary | /employee(name, [*ep,ip:intProp
("salary",salary),*ep2]) <- c);
}
```

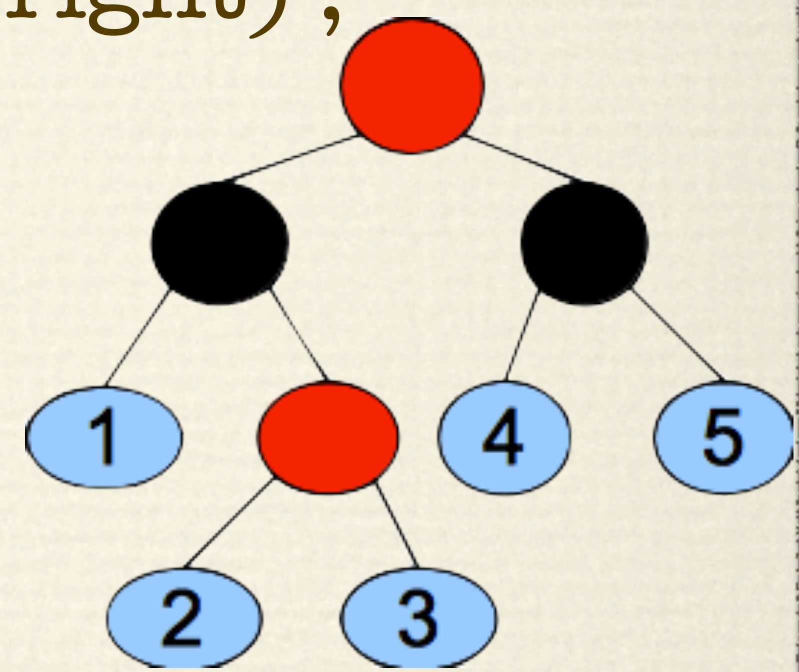
Polychromatic trees

```
data CTree = leaf(int N)
           | red(CTree left, CTree right)
           | black(CTree left, CTree right) ;

rb = red(black(leaf(1), red(leaf(2), leaf(3))),
         black(leaf(4), leaf(5)));

// TODO: count red nodes
```



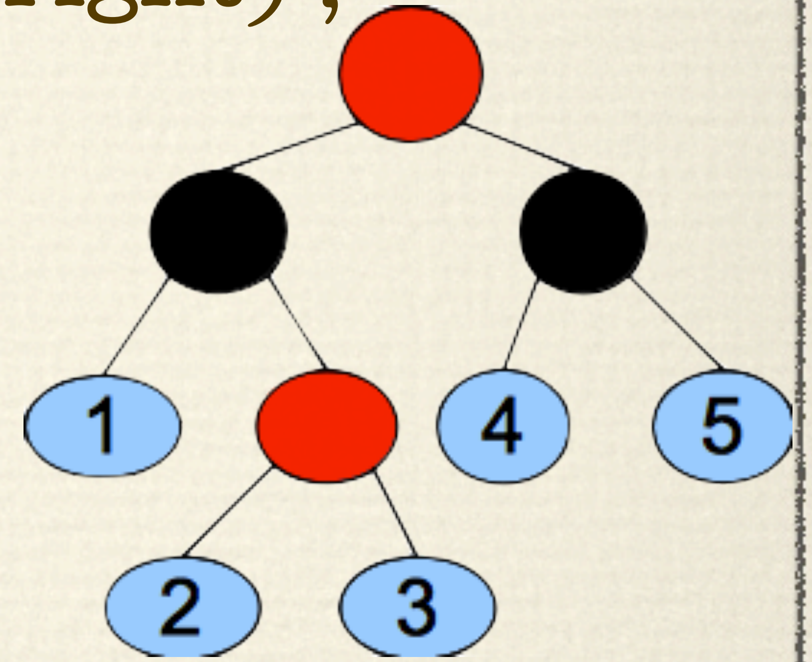
Polychromatic trees

```
data CTree = leaf(int N)
           | red(CTree left, CTree right)
           | black(CTree left, CTree right) ;

public int cntRed(CTree t) {
switch(t){
   case leaf(_): return 0;
   case red(L,R):
      return 1 + cntRed(L) + cntRed(R);
   case black(L,R):
      return cntRed(L) + cntRed(R);
}}
```



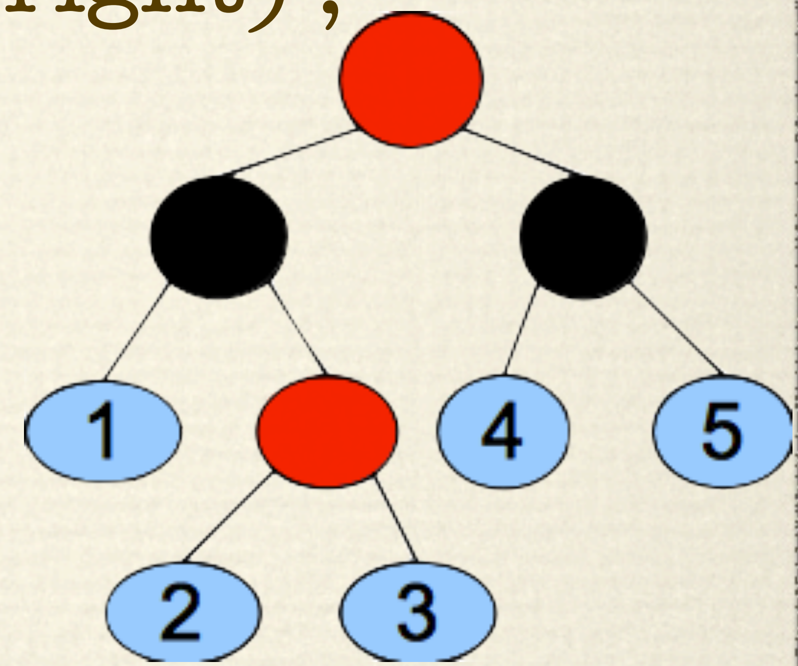
Polychromatic trees

```
data CTree =  leaf(int N)
            | red(CTree left, CTree right)
            | black(CTree left, CTree right) ;

public int cntRed(CTree t) {
  int c = 0;
  visit(t){case red(_,_): c += 1;};
  return c;
}
```



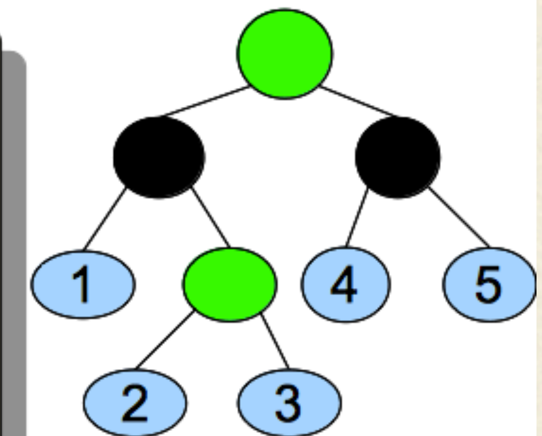
Polychromatic trees

```
data CTree =  leaf(int N)
           |  red(CTree left, CTree right)
           |  black(CTree left, CTree right) ;

public int cntRed(CTree t) =
 size([n | /n:red(_,_) := t]);
```

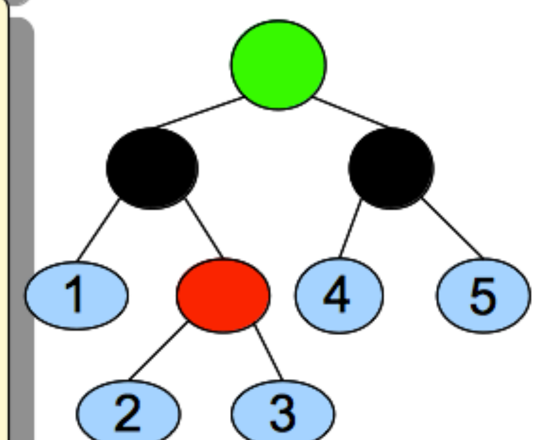


Full/shallow/deep

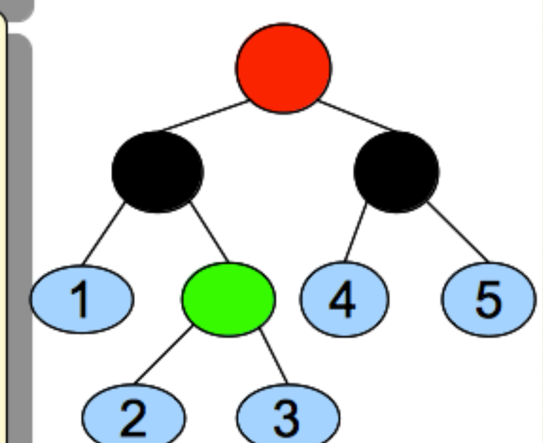
```
public CTree frepl(CTree T) {
    return visit (T) {
        case red(CTree T1, Ctree T2) =>  green(T1, T2)
    };
}
```




```
public Ctree srepl(CTree T) {
    return top-down-break visit (T) {
        case red(Ctree T1, CTree T2) =>  green(T1, T2)
    };
}
```




```
public Ctree drepl(Ctree T) {
    return bottom-up-break visit (T) {
        case red(CTree T1, CTree T2) =>  green(T1, T2)
    };
}
```



To summarise

- Rascal is a language workbench
- Sets, relations, maps…
- Grammars, ADTs…
- Cool comprehensions
- Cool pattern matching
- Many more features
- http://rascal-mpl.org

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

Work in progress

Frank Kovalchek, Work in progress at the base of the Eiffel Tower, 2013, CC-BY.

POWERED BY

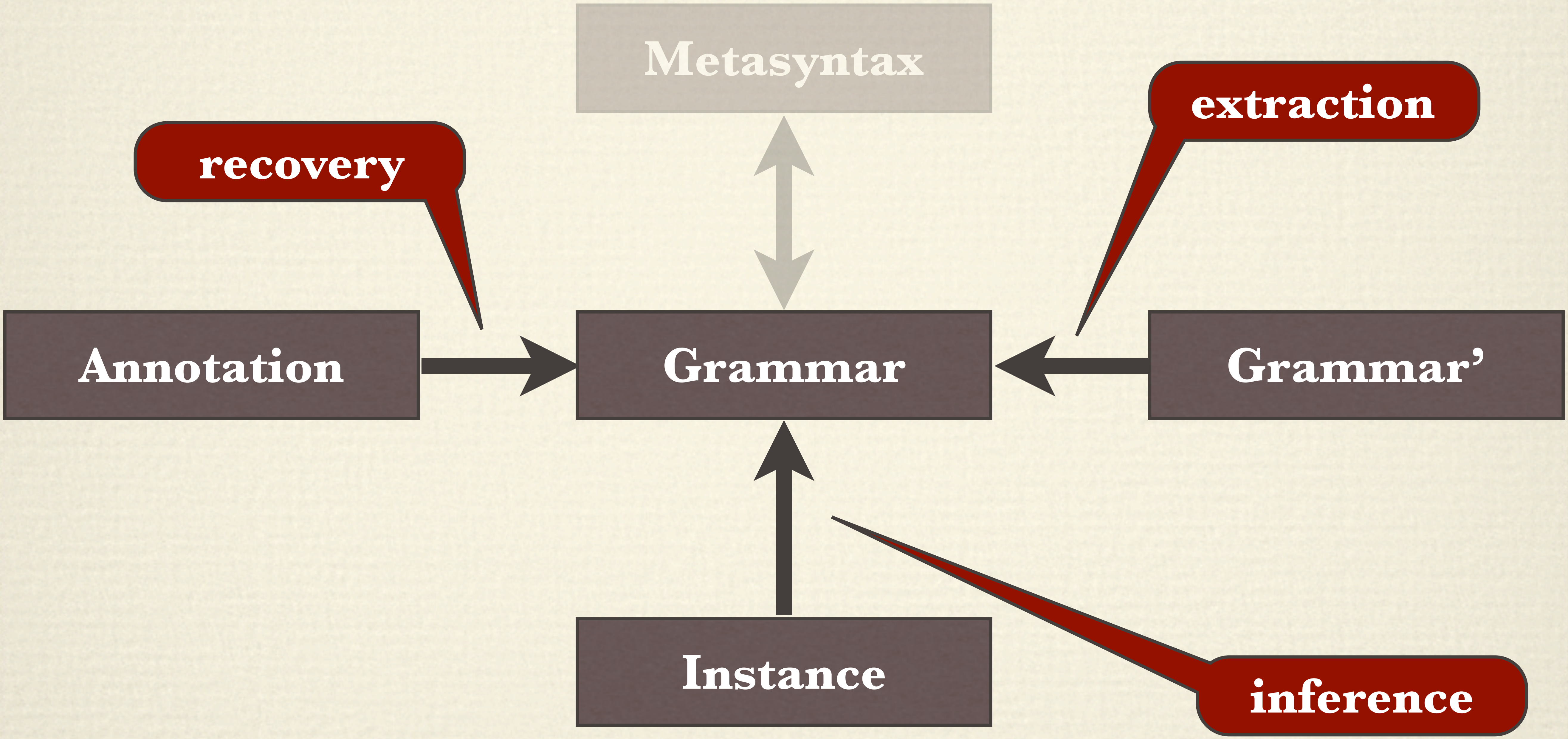
GRAMMARLAB

Grammar extraction

Grammar extraction

- Given is an artefact containing grammar knowledge:
  - a grammar
  - a parser specification
  - a metamodel
  - grammarware source code
  - a data schema
  - documentation
- How to extract a grammar from it?

Grammar extraction



Variety of notations

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

- Such details can be modeled
- We compose a notation specification

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

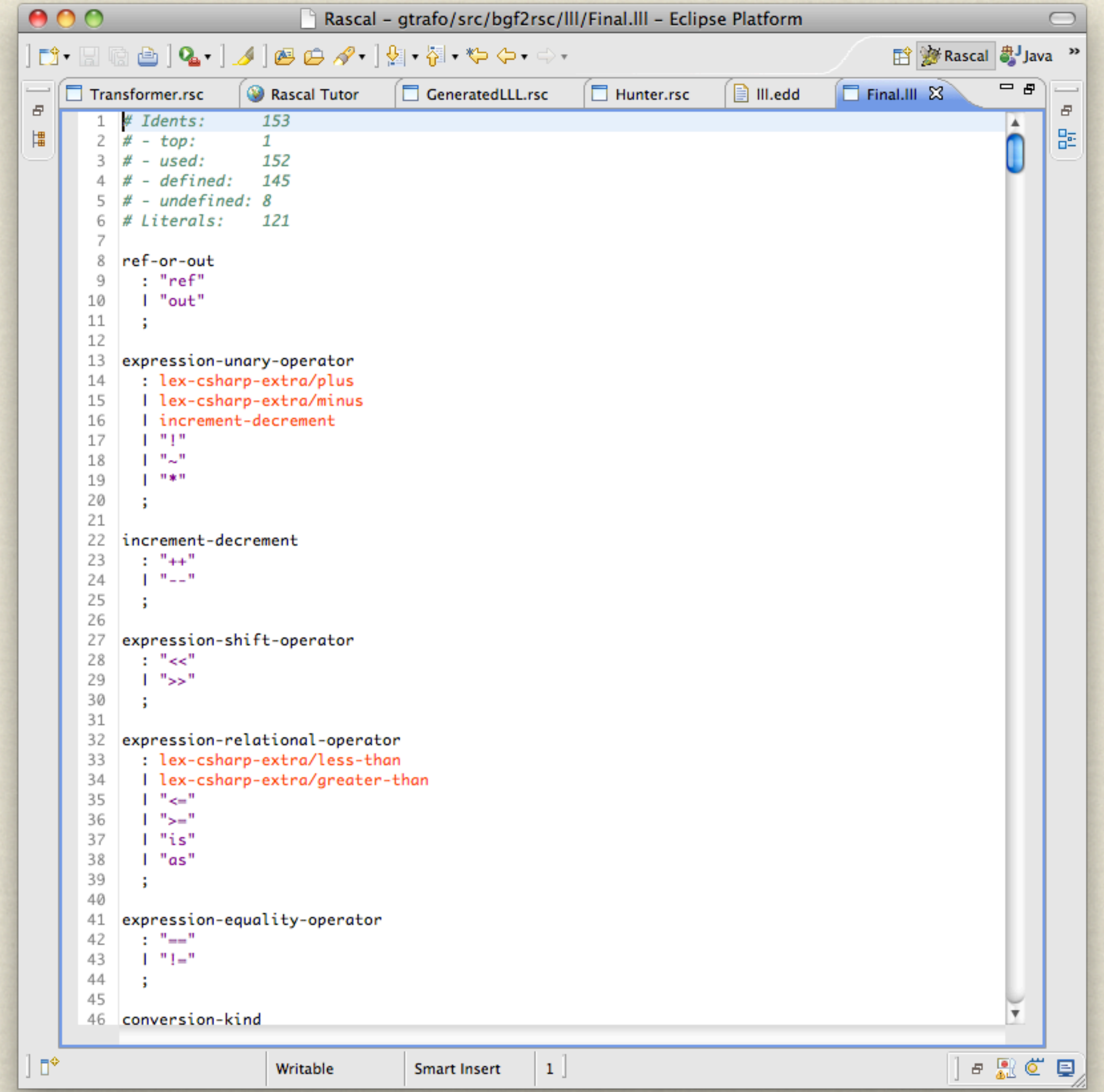
Grammar notations: BNF

Arithmetic Expressions, Boolean Expressions, and Expressions

⟨factor⟩ :≡ ⟨number⟩ $\overline{\text{or}}$ ⟨function⟩ $\overline{\text{or}}$ ⟨variable⟩ $\overline{\text{or}}$ ⟨subscr var⟩
$\overline{\text{or}}$ (⟨ar exp⟩) $\overline{\text{or}}$ ⟨factor⟩ ↑ ⟨ar exp⟩ ↓
⟨term⟩ :≡ ⟨factor⟩ $\overline{\text{or}}$ ⟨term⟩ × ⟨factor⟩ $\overline{\text{or}}$ ⟨term⟩ / ⟨factor⟩
⟨ar exp⟩ :≡ ⟨term⟩ $\overline{\text{or}}$ + ⟨term⟩ $\overline{\text{or}}$ - ⟨term⟩ $\overline{\text{or}}$ ⟨ar exp⟩ + ⟨term⟩
$\overline{\text{or}}$ ⟨ar exp⟩ - ⟨term⟩

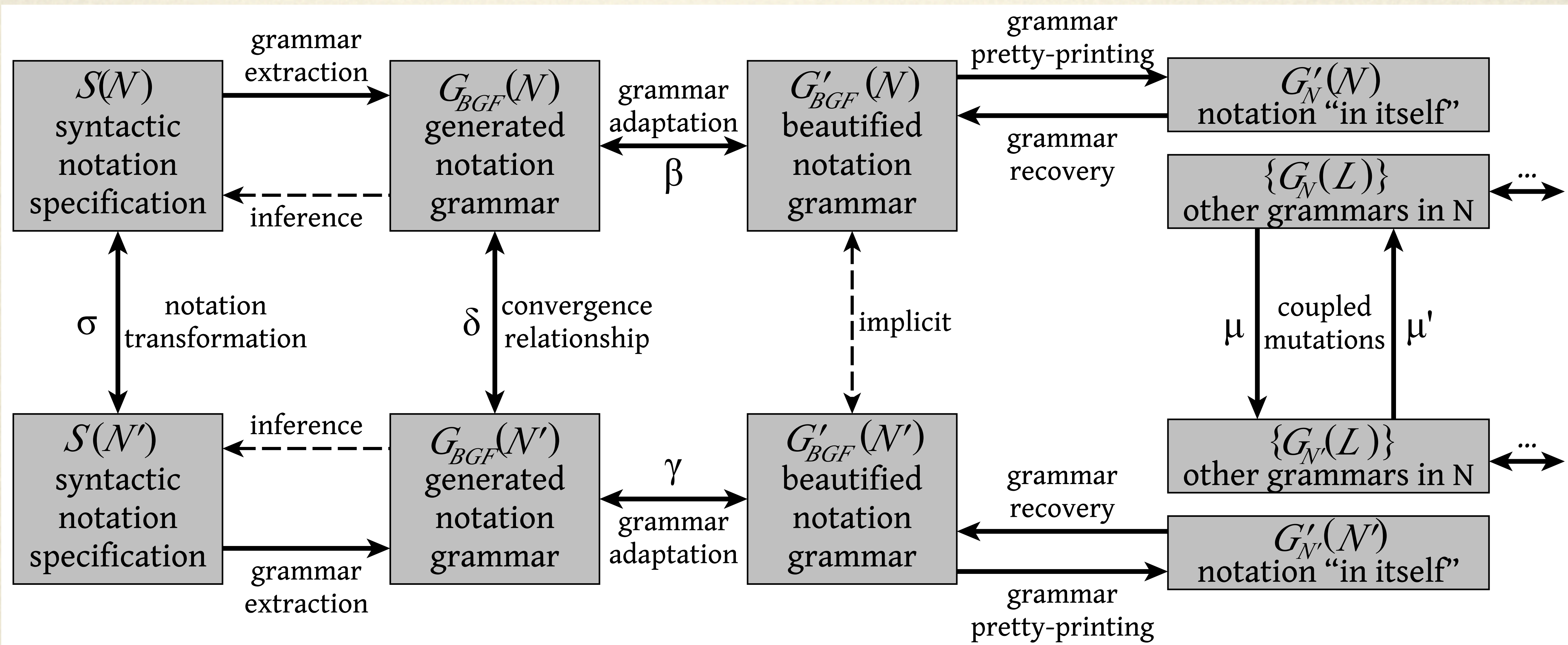
⟨ar exp A⟩ :≡ ⟨ar exp⟩
⟨relation⟩ :≡ $<$ $\overline{\text{or}}$ $>$ $\overline{\text{or}}$ $\leq$ $\overline{\text{or}}$ $\geq$ $\overline{\text{or}}$ $=$ $\overline{\text{or}}$ $\neq$
⟨rel exp⟩ :≡ (⟨ar exp⟩ ⟨relation⟩ ⟨ar exp A⟩)
⟨bool term⟩ :≡ 0 $\overline{\text{or}}$ 1 $\overline{\text{or}}$ ⟨rel exp⟩ $\overline{\text{or}}$ ⟨function⟩ $\overline{\text{or}}$
⟨variable⟩ $\overline{\text{or}}$ ⟨subscr var⟩ $\overline{\text{or}}$ (⟨bool exp⟩)
$\overline{\text{or}}$ ¬ ⟨bool term⟩
⟨bool exp⟩ :≡ ⟨bool term⟩ $\overline{\text{or}}$ ⟨bool exp⟩ ⋁ ⟨bool term⟩
$\overline{\text{or}}$ ⟨bool exp⟩ ⋀ ⟨bool term⟩ $\overline{\text{or}}$
⟨bool exp⟩ ≡ ⟨bool term⟩

⟨exp⟩ :≡ ⟨ar exp⟩ $\overline{\text{or}}$ ⟨bool exp⟩



Zaytsev, <u>*Notation-Parametric Grammar Recovery*</u>, LDTA 2012.

Metasyntactic evolution



Zaytsev, *Language Evolution, Metasyntactically*, EC-EASST 49 (BX), 2012.

EBNF in GrammarLab

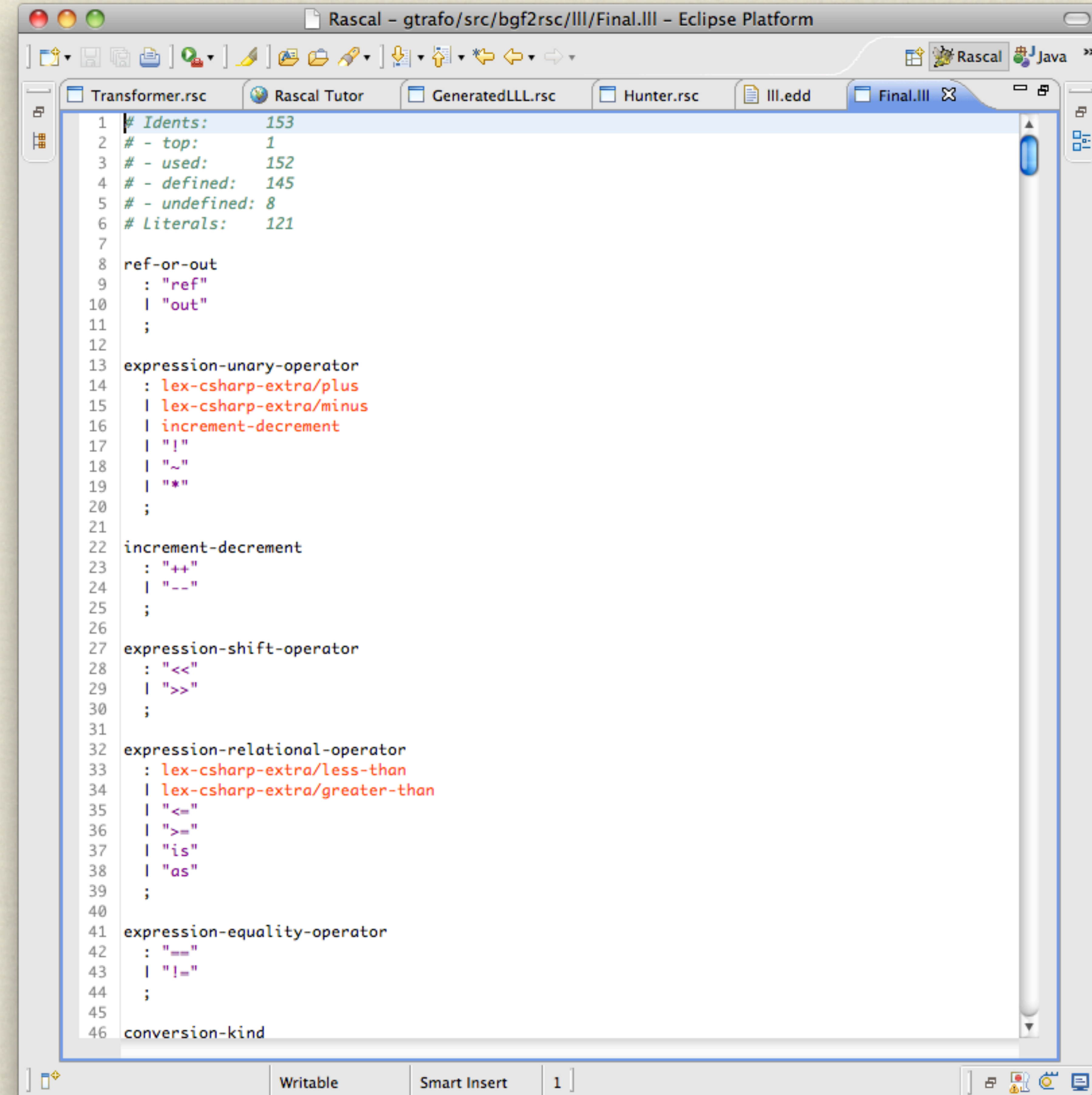
```
alias EBNF = map[Metasymbol,str];
data Metasymbol
    = start_grammar_symbol()
    | end_grammar_symbol()
    | epsilon_metasymbol()
    | defining_symbol()
    | multiple_defining_symbol()
    | terminator_symbol()
    | definition_separator_symbol()
    | start_option_symbol()
    | end_option_symbol()
    | start_nonterminal_symbol()
    | end_nonterminal_symbol()
…
```

EBNF in GrammarLab

```
public EBNF DefaultEBNF = (
    epsilon_metasymbol(): "ε",
    defining_symbol(): " ::= ",
    terminator_symbol(): " ;\n",
    definition_separator_symbol(): " | ",
    start_group_symbol(): "(",
    end_group_symbol(): ")",
    start_terminal_symbol(): "\"",
    end_terminal_symbol(): "\"",
    postfix_option_symbol(): "?",
    …
);
```

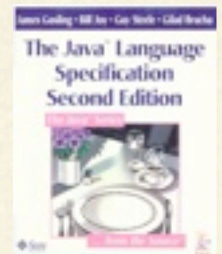
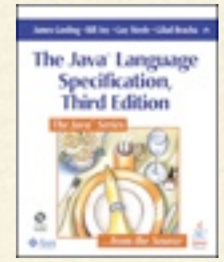
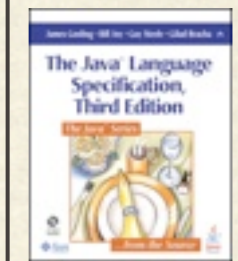
IDE support

```
EBNF ebnf = …;
edd2rsc(ebnf, |cwd:///EBNFGrammar.rsc|);

…

Tree getEBNF(str s, loc z)
   = parse(#EBNFGrammar,z);
public void registerEBNF()
   = registerLanguage("EBNF","bnf", getEBNF);
```



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

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

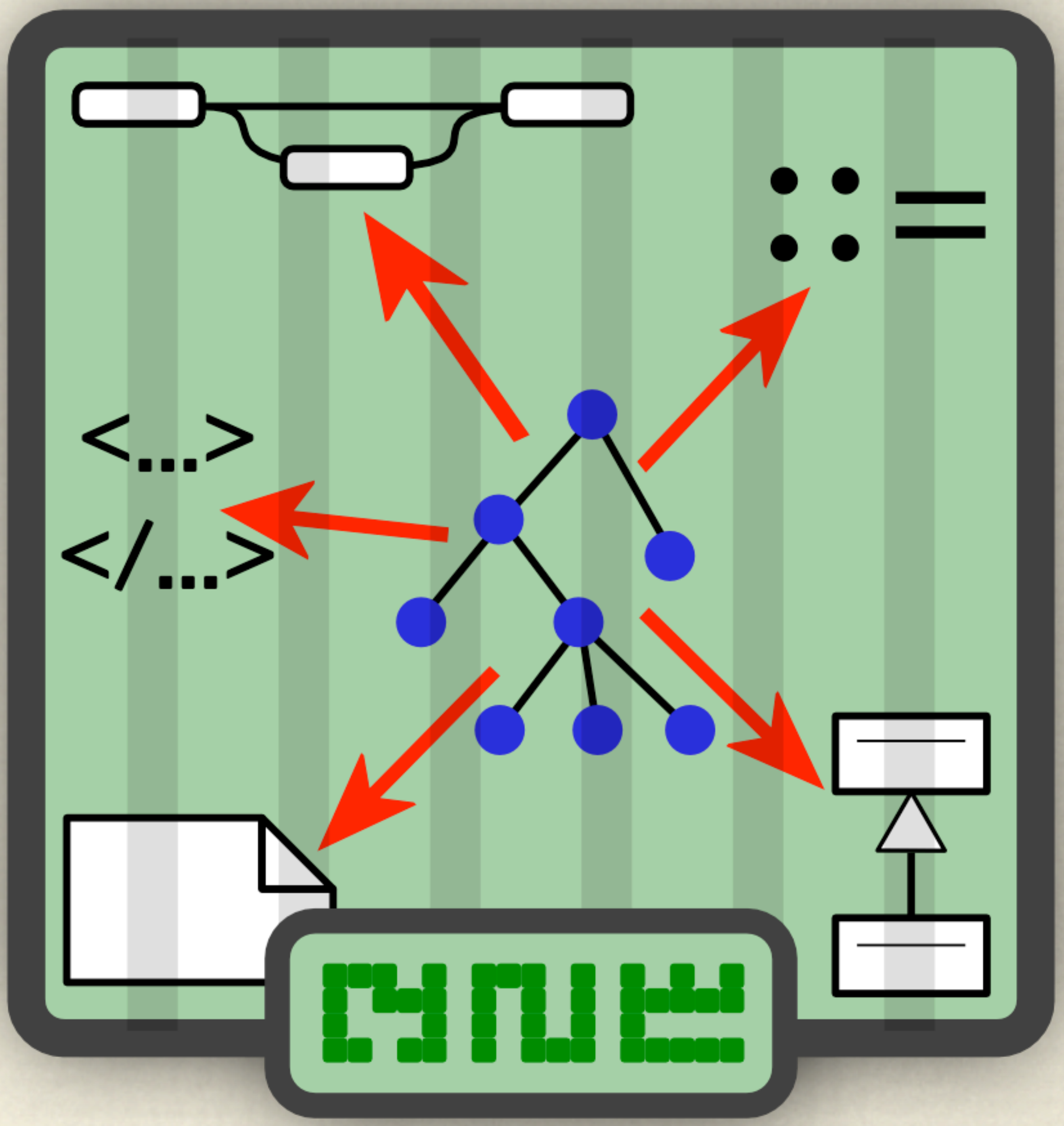
Lämmel, Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SQJ 19:2, March 2011.

Extraction & recovery

```
EBNF ebnf = …;
GGrammar bgf
  = ebnf2bgf(ebnf, |cwd:///grammar.ebnf|);


GGrammar bgf
  = hunter(ebnf, |cwd:///grammar.ebnf|);
// possibly with errors
```

Notation-parametric recovery

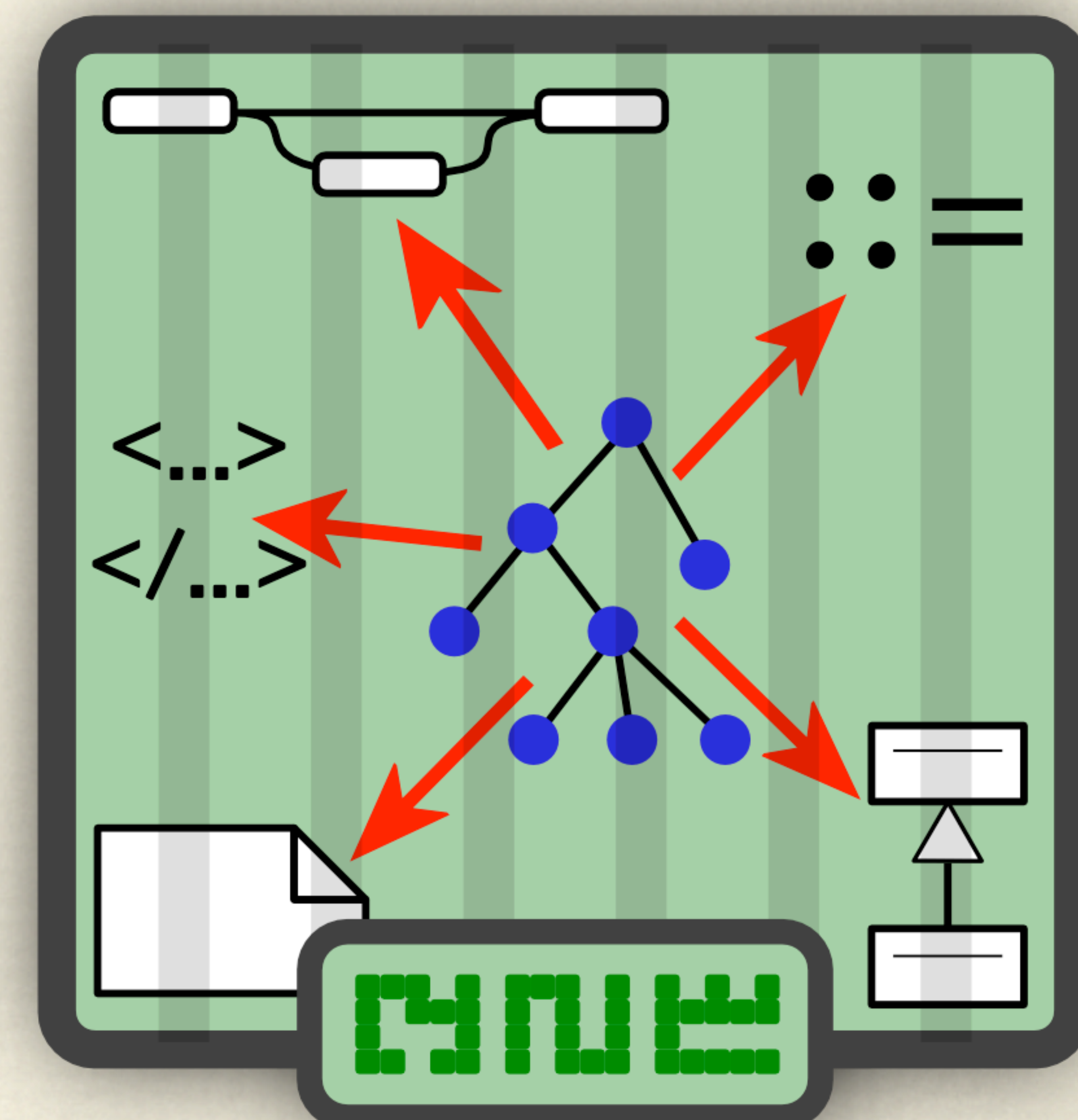
- Perform a robust heuristic-based recovery process
- Successful for grammars of Ada, C, C++, C♯, Dart, Eiffel, Modula, MediaWiki, LLL, EBNF, etc.



Notation-parametric recovery

- Perform a robust heuristic-based recovery process
- Successful for grammars of Ada, C, C++, C♯, Dart,

  Eiffel, Modula, MediaWiki, LLL, EBNF, etc.



Zaytsev, *Grammar Zoo: A Repository of Experimental Grammarware*. Under review for SCP EST5, 2013.

Grammar extraction

from other sources

Other available extractors

- Eclipse Modeling Framework

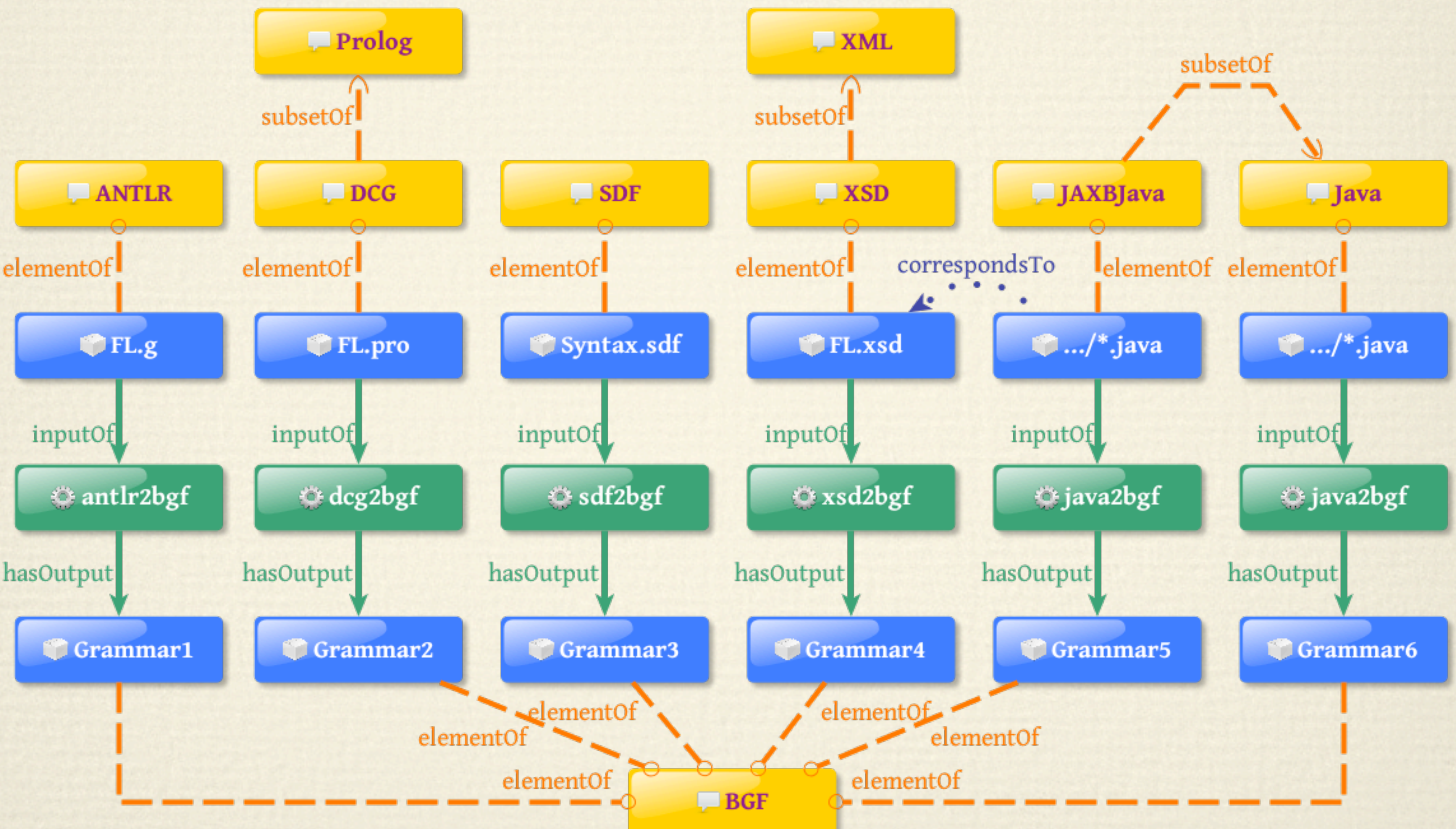
```
GGrammar bgf
    = ecore2bgf(|cwd:///metamodel.ecore|);
```

- XML Schema Definition

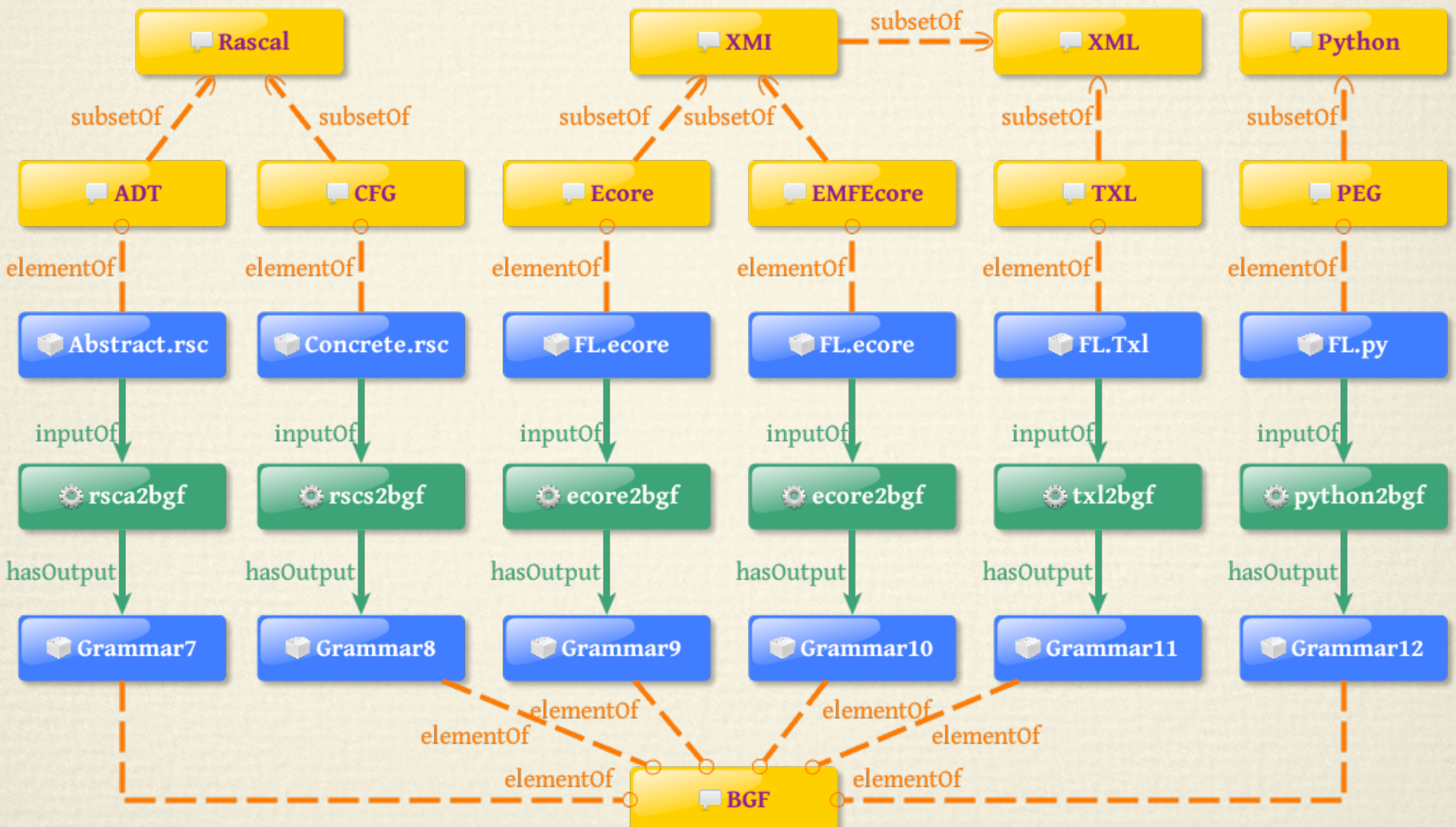
```
GGrammar bgf
    = xsd2bgf(|cwd:///schema.xsd|);
```

Not all extractors have already been migrated to use Rascal

Flashy megamodel!



Flashy megamodel!!!



Grammar Zoo

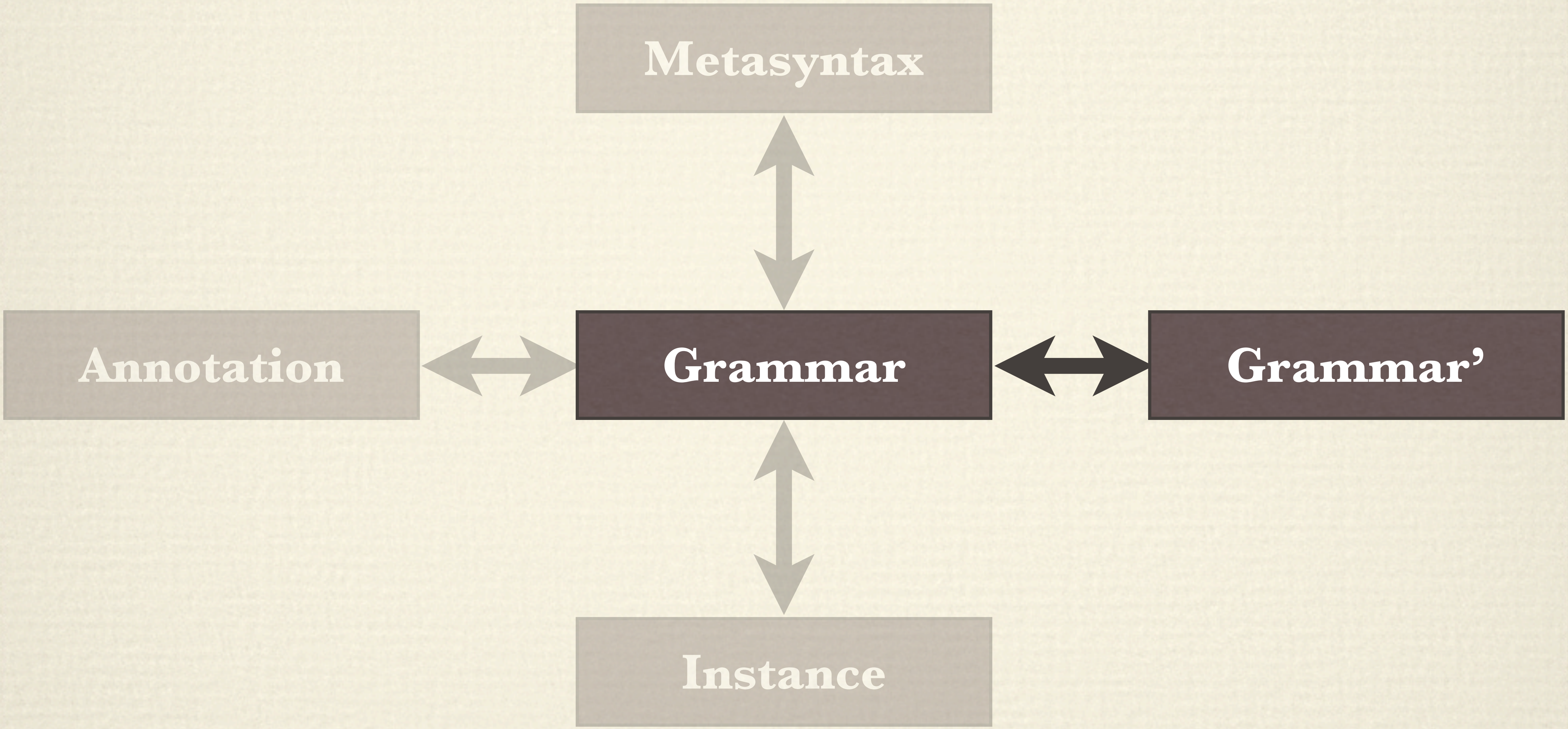
- 569 grammars
  - Ada, ASM, ATL, ATOM, BASIC, BibTeX, C, C++, C#, CSV, Dart, Dot, DTD, EBNF, Eiffel, Fortran, HTML, Java, JavaScript, JSON, Modula, Occam, ODF, Pascal, PIF, PL/I, SVG, UML, Wiki, XML, XPath, XSLT, XQuery, …

Grammar evolution

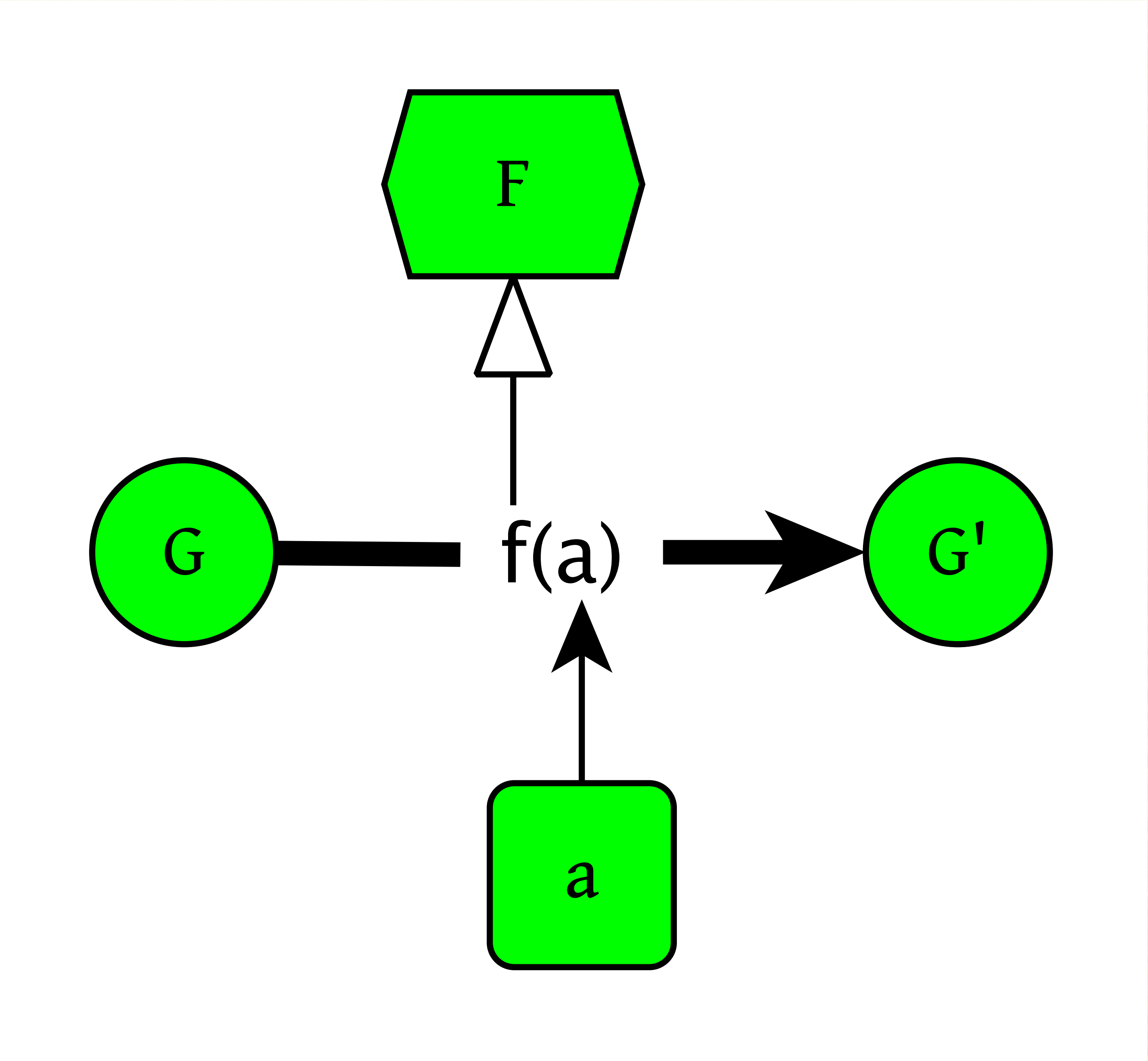
Grammar evolution

- Any change is a transformation
- Why transform?
  - adaptation
  - normalisation
  - beautification
  - inconsistency management
  - version control
- Documented, well-understood, compositional change
- Good for representing relationships

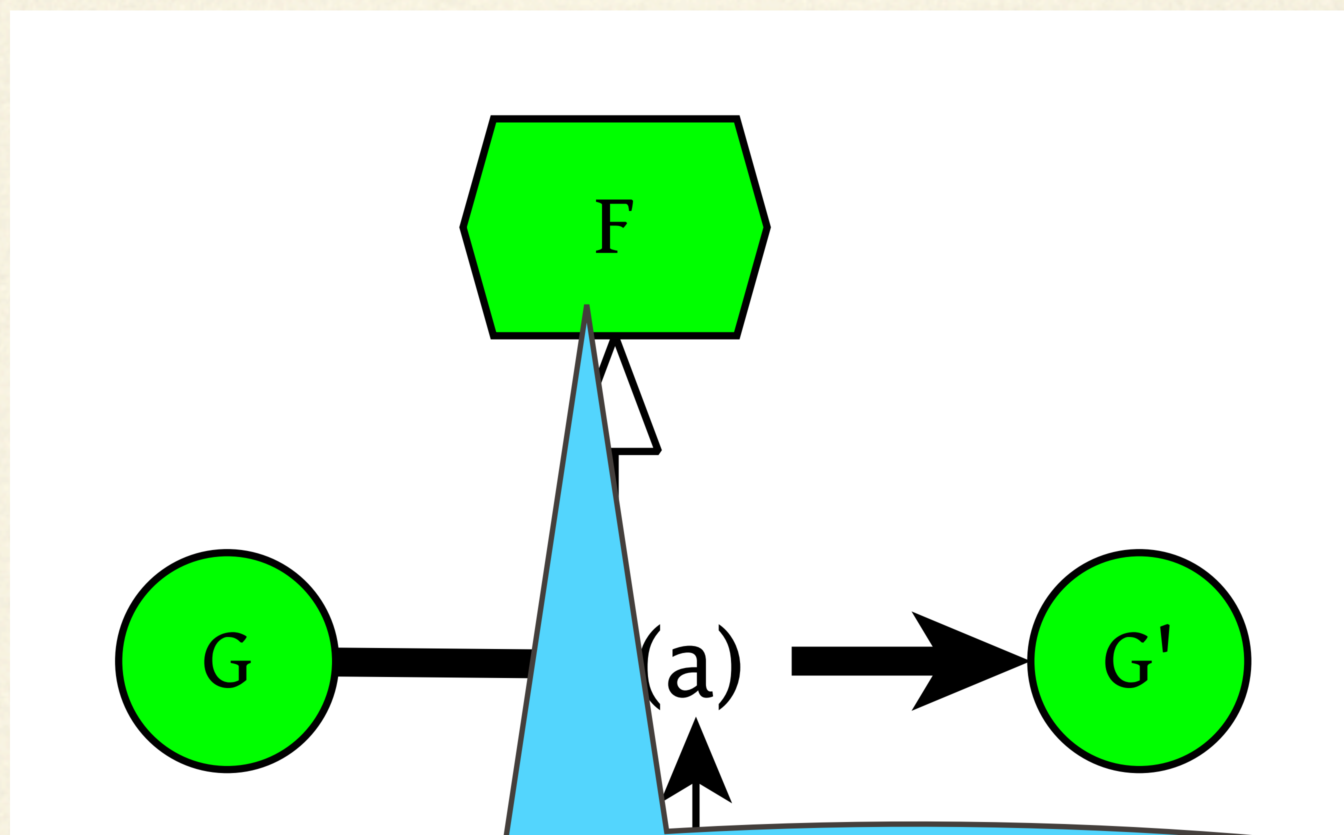
Grammar evolution



Transformation components



Transformation components



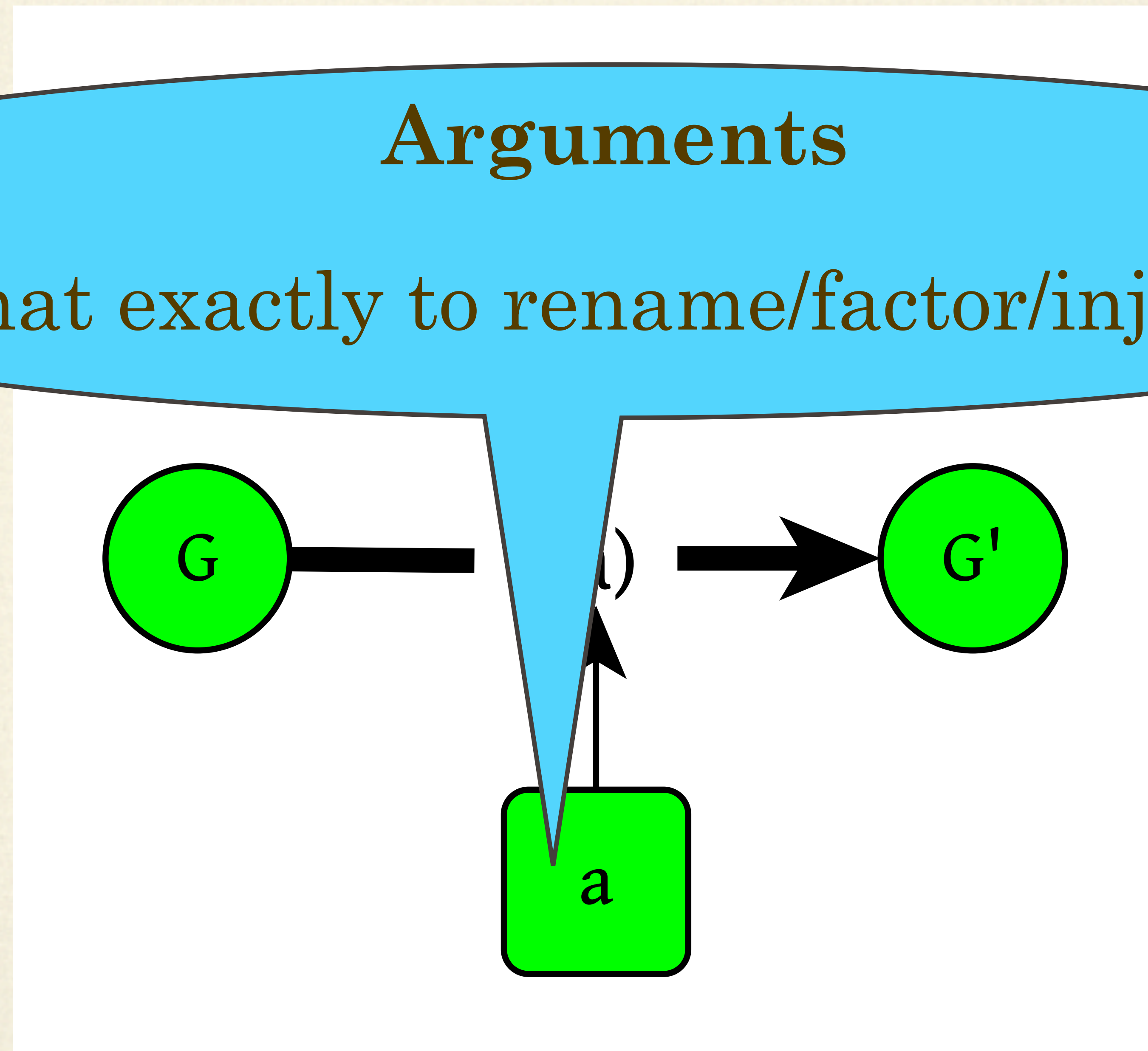
Operator

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

Transformation components

Arguments

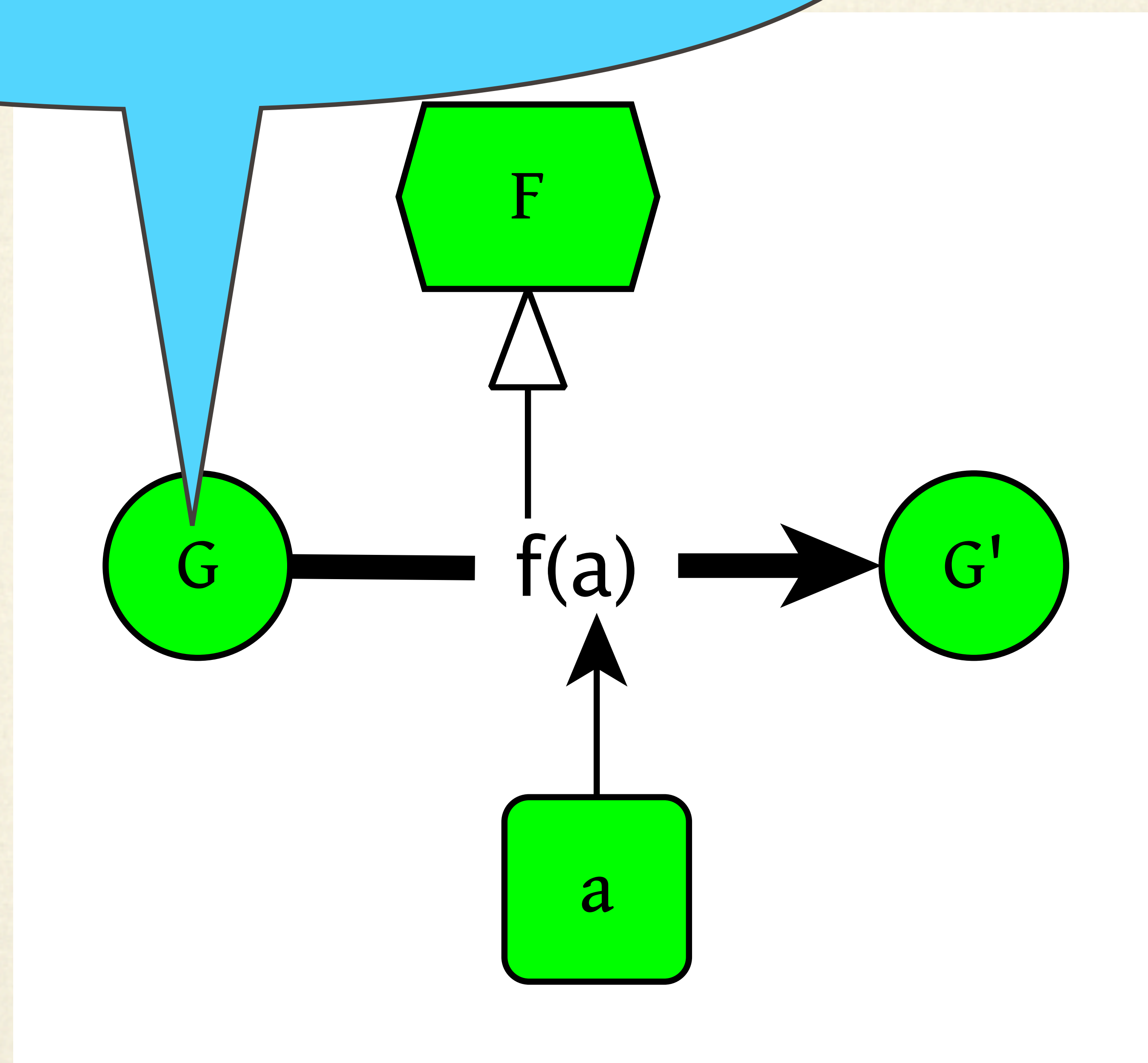
- what exactly to rename/factor/inject/...?



Transformation components

Input grammar

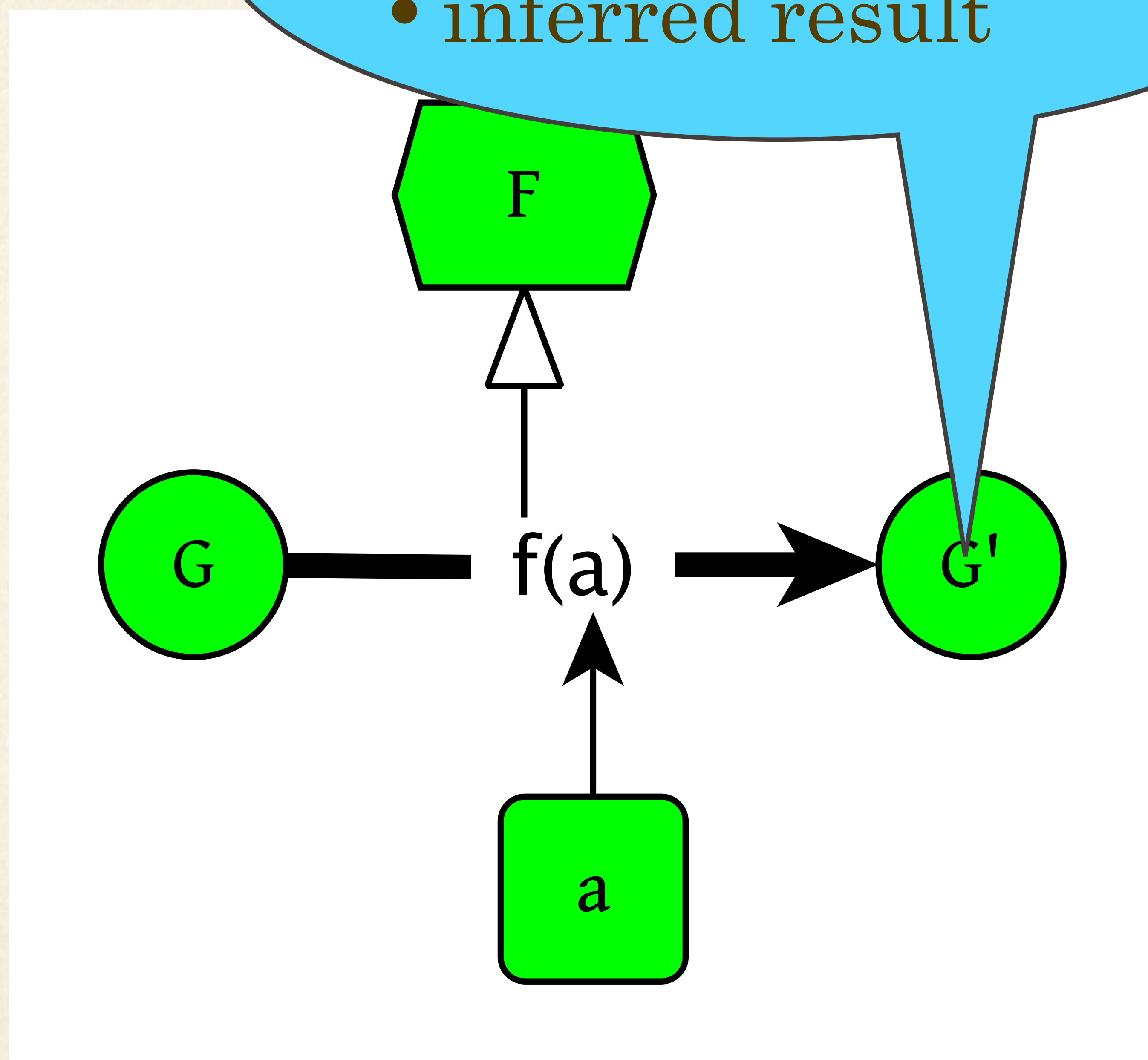
- determines applicability



Transformation components

Output grammar

- inferred result



Excerpt from the operator suite

Operator [LZ09, LZ11, ...]	**Group [LZ11]**	**Class [CREP08]**	**Delta [GKP07]**
bypass()	preserving	updative	not breaking
factor(x, y)	preserving	updative	not breaking
introduce$(n ::= rhs)$	preserving	additive	not breaking
eliminate(n)	preserving	subtractive	not breaking
widen(x, y)	increasing	additive	not breaking
define$(n ::= rhs)$	revising	additive	not breaking
renameN(a, b)	preserving	updative	resolvable
fold(n)	preserving	additive	resolvable
extract$(n ::= rhs)$	preserving	additive	resolvable
unfold(n)	preserving	subtractive	resolvable
disappear(p, m)	decreasing	subtractive	resolvable
permute(p, q)	revising	updative	resolvable
concretize(p, m)	revising	additive	resolvable
abstractize(p, m)	revising	subtractive	resolvable
project(p, m)	revising	subtractive	resolvable
narrow(x, y)	decreasing	subtractive	unresolvable
inject(p, m)	revising	additive	unresolvable
replace(x, y)	revising	—	unresolvable

Zaytsev, *Negotiated Grammar Transformation*, JOT. (under review)

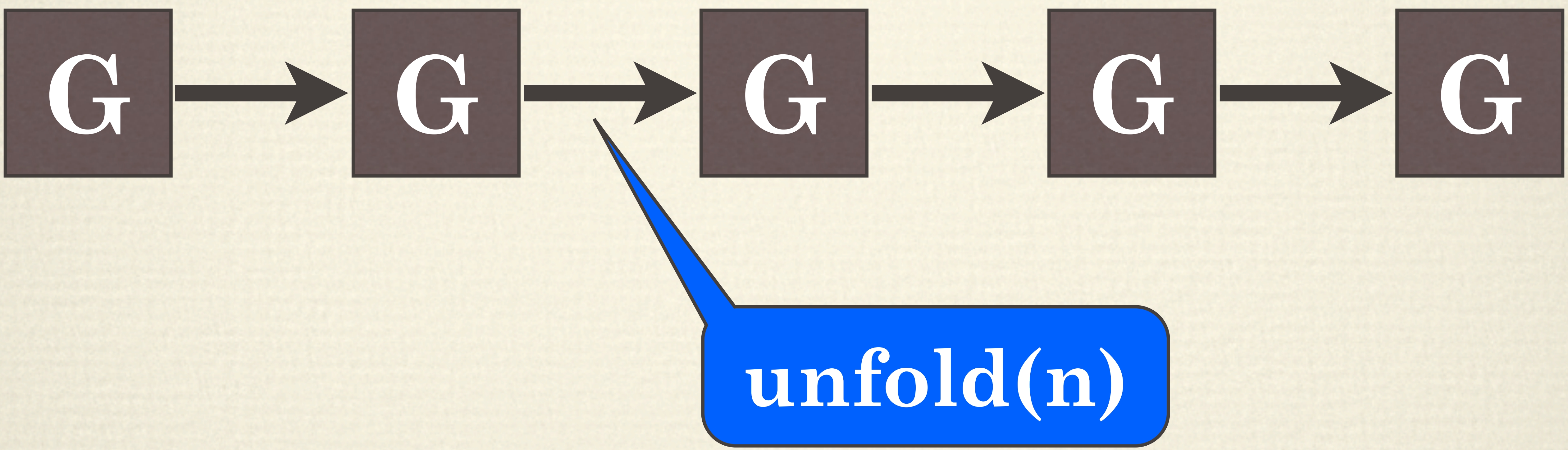
Transform to converge

	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

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

Grammar programming

Grammar programming

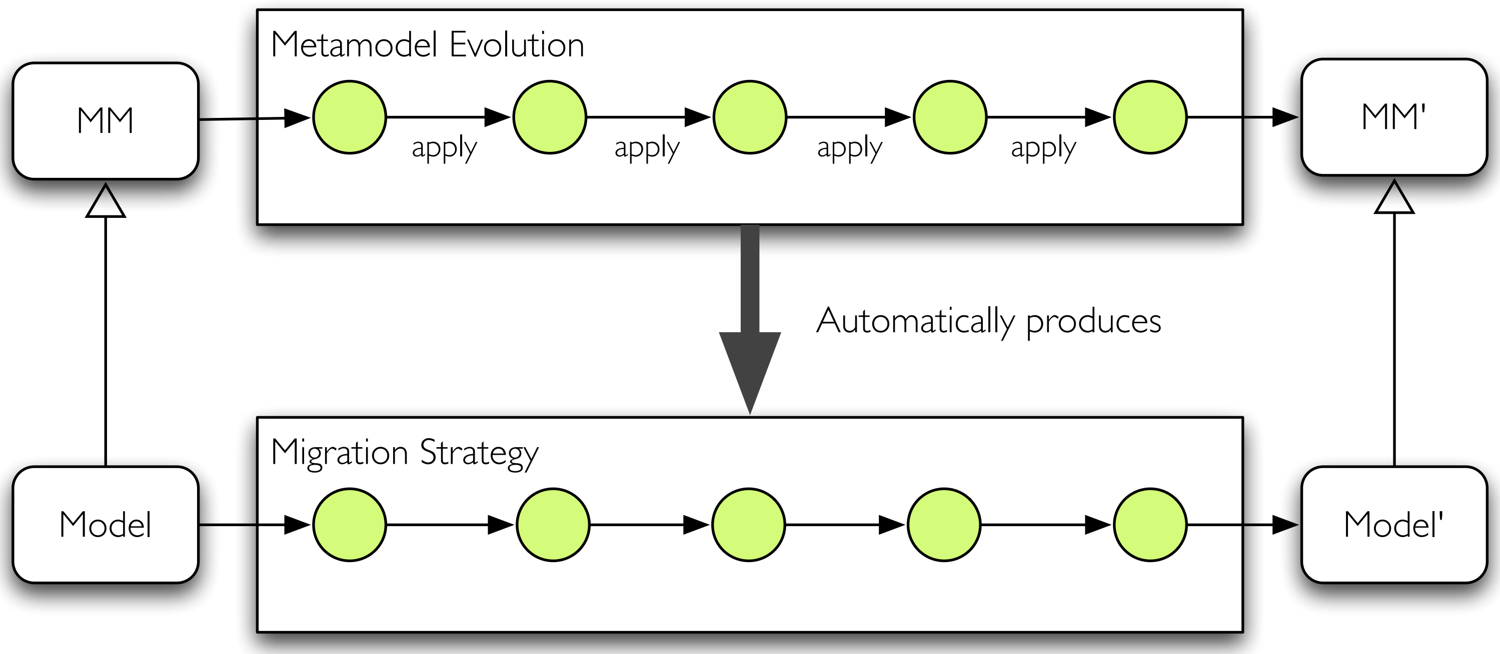


Meanwhile in modelware...

Model Migration Approaches:

Operator-based co-evolution

+ migration strategy is "free"
- tool lock in
- operator set completeness
- reverse engineering challenging



J.R. Williams, R.F. Paige, F.A.C. Polack, *Searching for Model Migration Strategies*. ME 2012.

A transformation sequence

```
expr ::= …;
atom ::= ID | INT  | '(' expr ')';
```

abstractize

```
expr ::= …;
atom ::= ID | INT  | expr;
```

vertical

```
expr ::= …;
atom ::= ID;
atom ::= INT;
atom ::= expr;
```

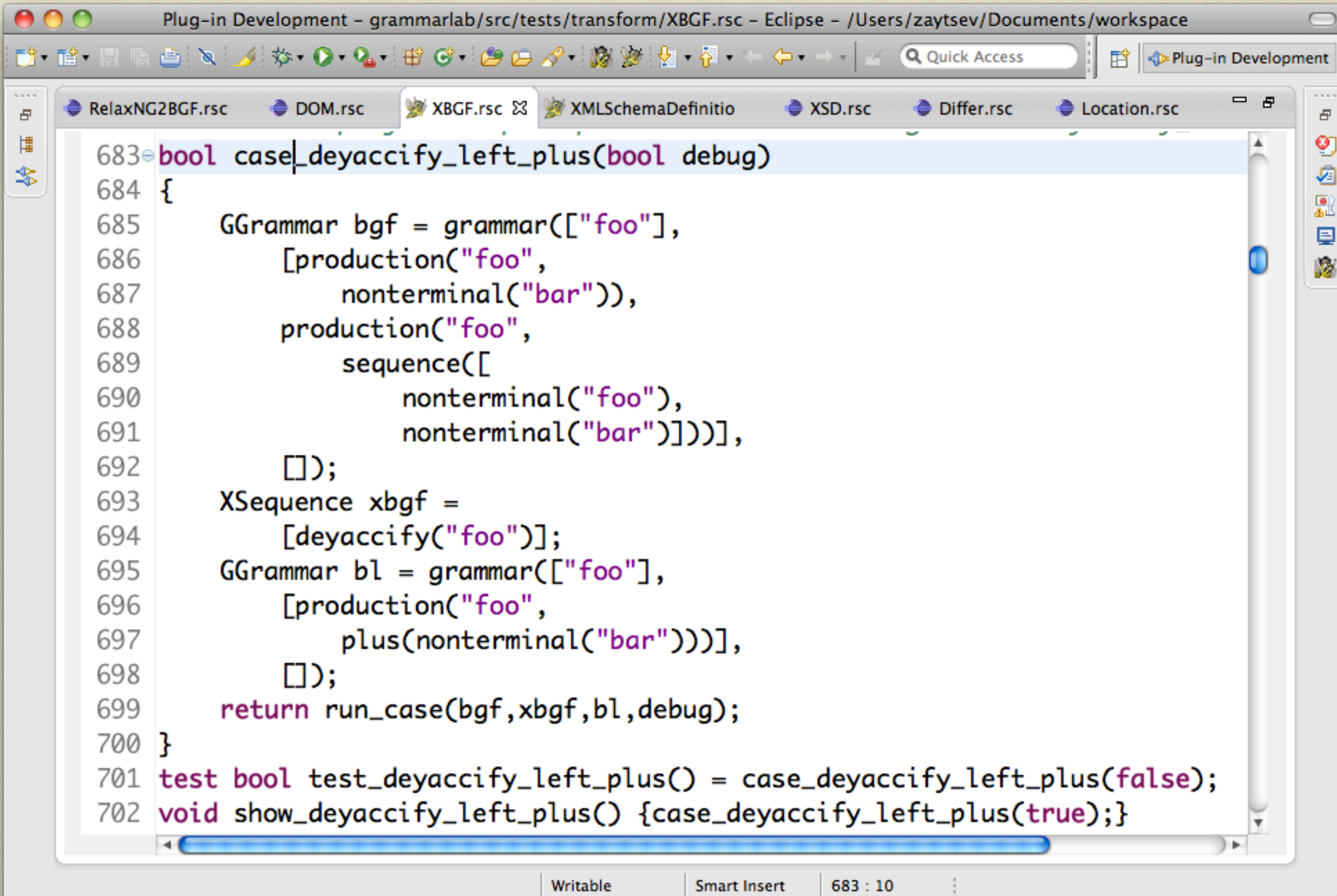
unite

```
expr ::= …;
expr ::= ID;
expr ::= INT;
expr ::= expr;
```

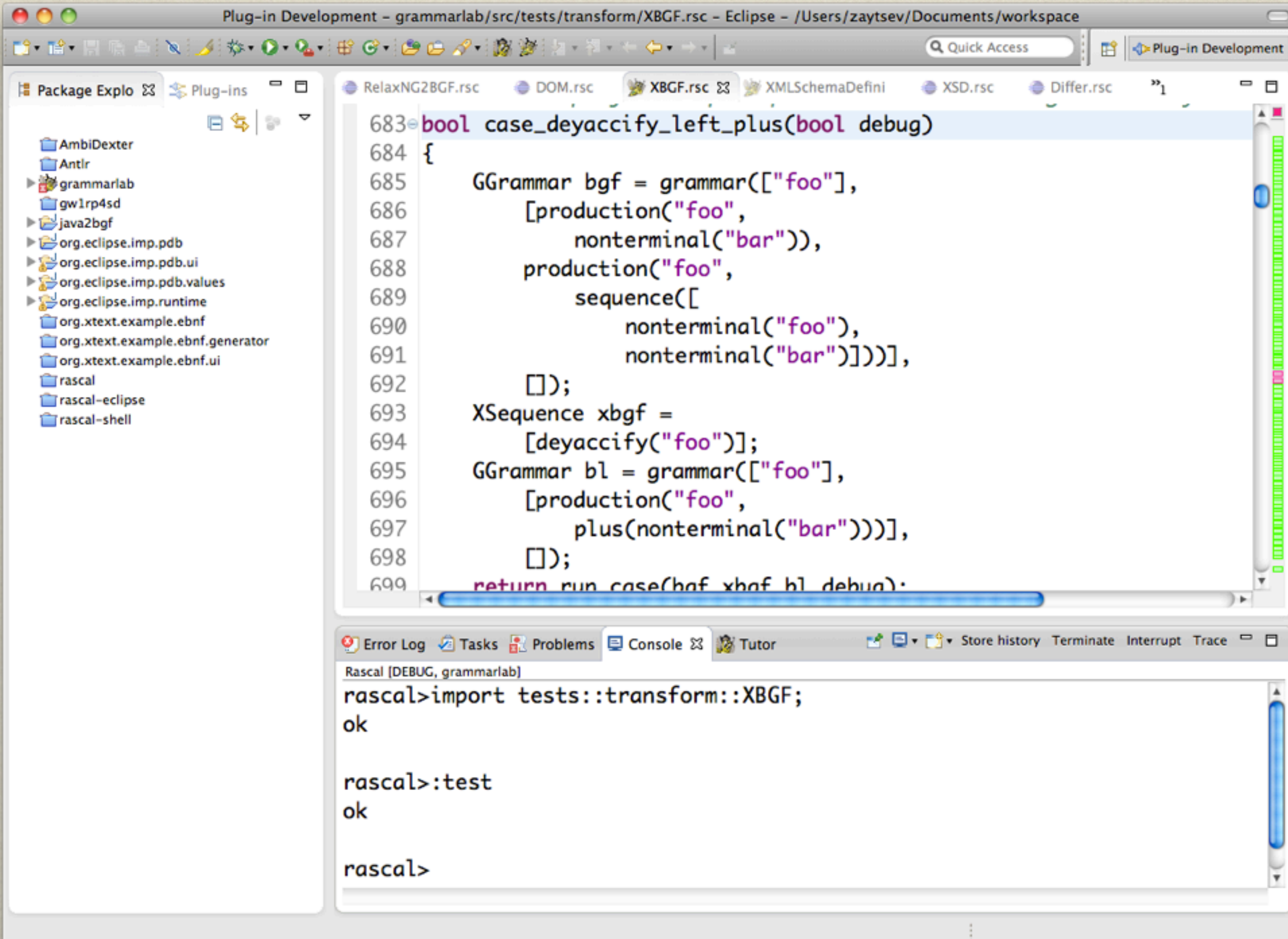
abridge

```
expr ::= …;
expr ::= ID;
expr ::= INT;
```

R. Lämmel, V. Zaytsev, _An Introduction to Grammar Convergence_. IFM 2009, LNCS 5423.



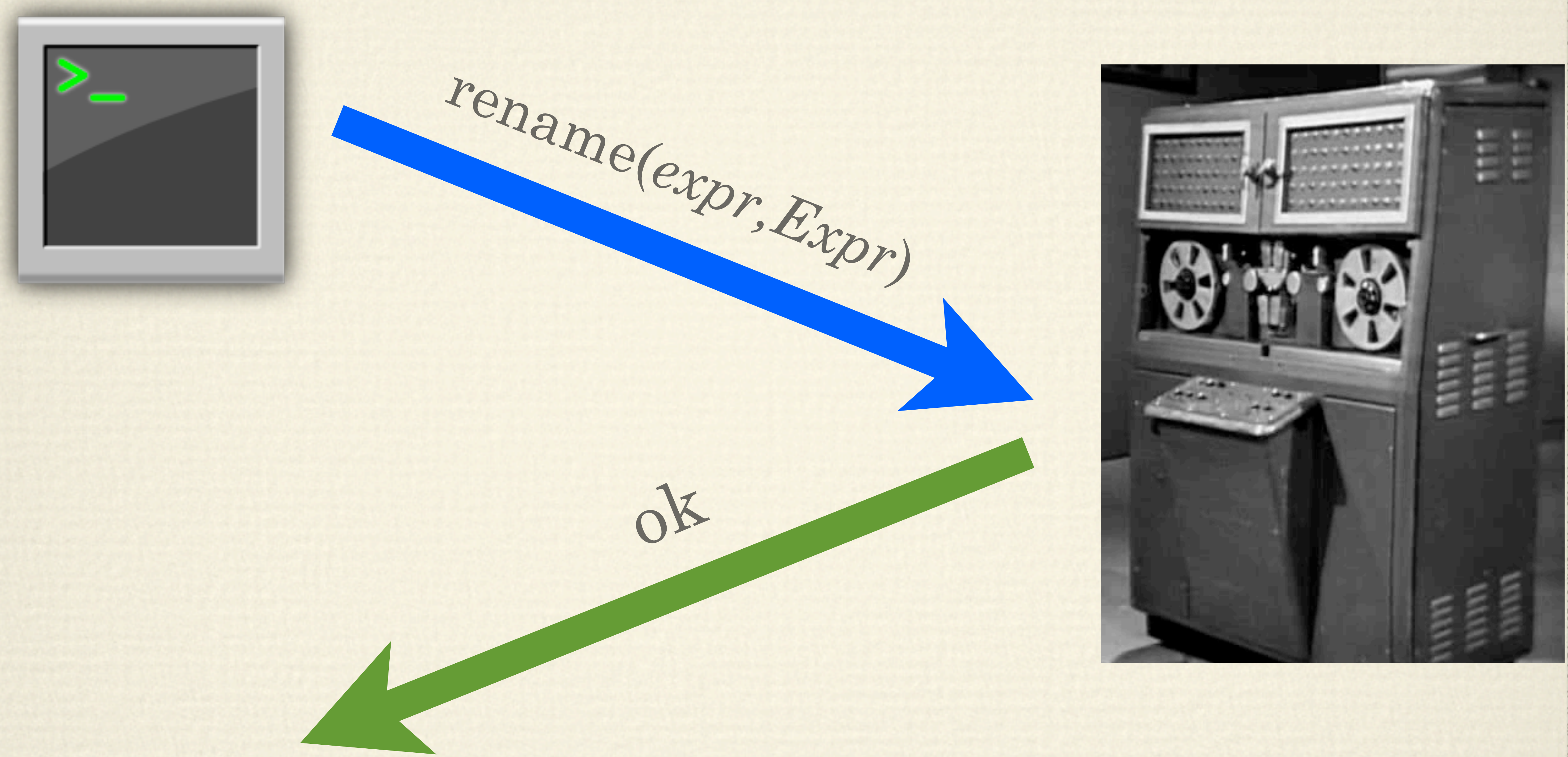
Plug-in Development - grammarlab/src/tests/transform/XBGF.rsc - Eclipse - /Users/zaytsev/Documents/workspace
Quick Access
Plug-in Development
RelaxNG2BGF.rsc
DOM.rsc
XBGF.rsc
XMLSchemaDefinitio
XSD.rsc
Differ.rsc
Location.rsc
683 bool case_deyaccify_left_plus(bool debug)
684 {
685   GGrammar bgf = grammar(["foo"],
686   [production("foo",
687   nonterminal("bar")),
688   production("foo",
689   sequence([
690   nonterminal("foo"),
691   nonterminal("bar")]))],
692   []);
693   XSequence xbgf =
694   [deyaccify("foo")];
695   GGrammar bl = grammar(["foo"],
696   [production("foo",
697   plus(nonterminal("bar")))],
698   []);
699   return run_case(bgf,xbgf,bl,debug);
700 }
701 test bool test_deyaccify_left_plus() = case_deyaccify_left_plus(false);
702 void show_deyaccify_left_plus() {case_deyaccify_left_plus(true);}
Writable
Smart Insert
683 : 10


Plug-in Development - grammarlab/src/tests/transform/XBGF.rsc - Eclipse - /Users/zaytsev/Documents/workspace
Quick Access
Plug-in Development
Package Explo
Plug-ins
AmbiDexter
Antlr
grammarlab
gw1rp4sd
java2bgf
org.eclipse.imp.pdb
org.eclipse.imp.pdb.ui
org.eclipse.imp.pdb.values
org.eclipse.imp.runtime
org.xtext.example.ebnf
org.xtext.example.ebnf.generator
org.xtext.example.ebnf.ui
rascal
rascal-eclipse
rascal-shell
RelaxNG2BGF.rsc
DOM.rsc
XBGF.rsc
XMLSchemaDefini
XSD.rsc
Differ.rsc
bool case_deyaccify_left_plus(bool debug)
{
GGrammar bgf = grammar(["foo"],
[production("foo",
nonterminal("bar")),
production("foo",
sequence([
nonterminal("foo"),
nonterminal("bar")]))],
[]);
XSequence xbgf =
[deyaccify("foo")];
GGrammar bl = grammar(["foo"],
[production("foo",
plus(nonterminal("bar")))],
[]);
Error Log
Tasks
Problems
Console
Tutor
Store history
Terminate
Interrupt
Trace
Rascal [DEBUG, grammarlab]
rascal>import tests::transform::XBGF;
ok
rascal>:test
ok
rascal>

Negotiated transformations

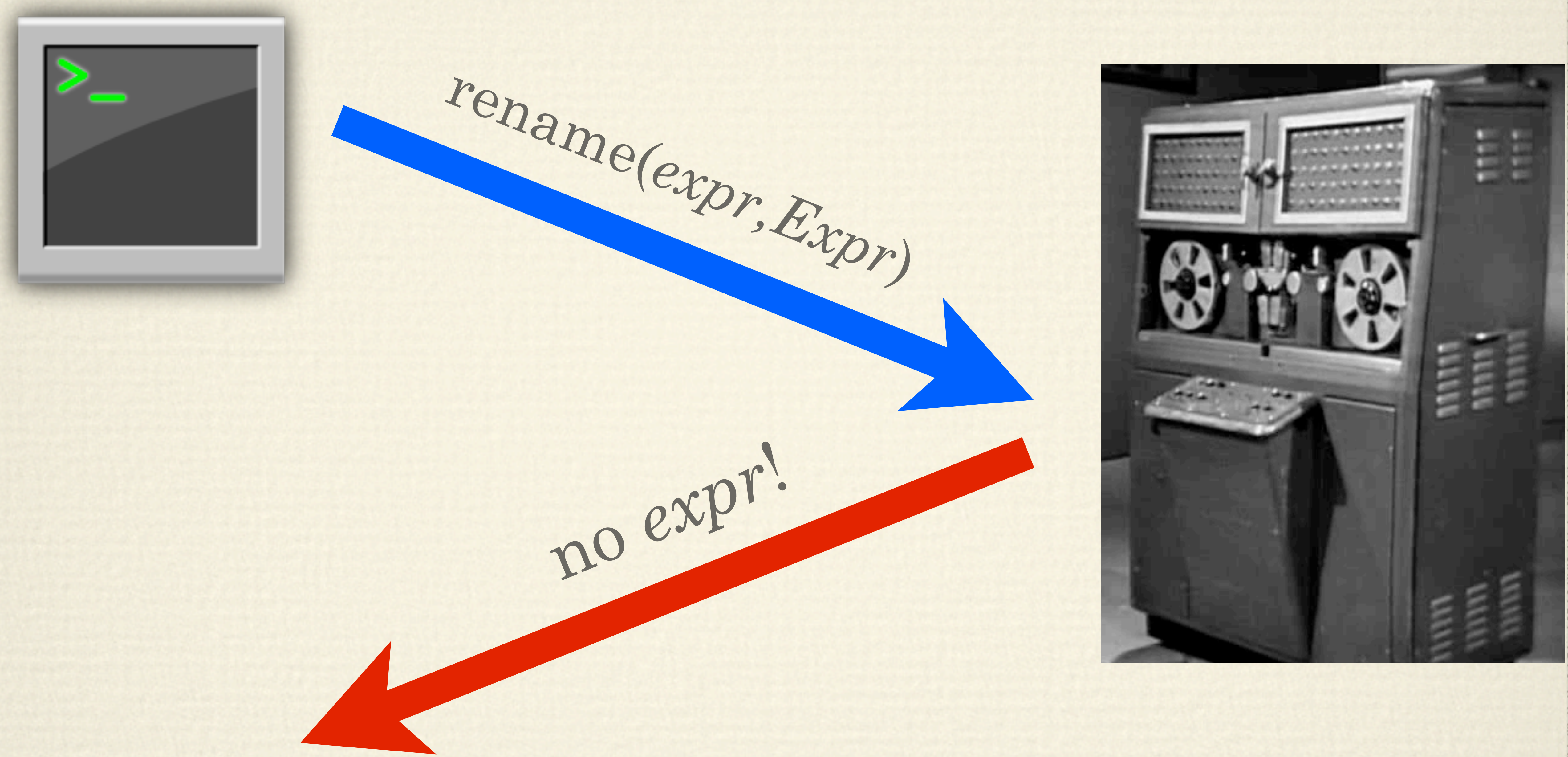
Doctor Who — Second Doctor — Season 6 — Episode 3 — The Invasion — Part 2

Rename a nonterminal: success



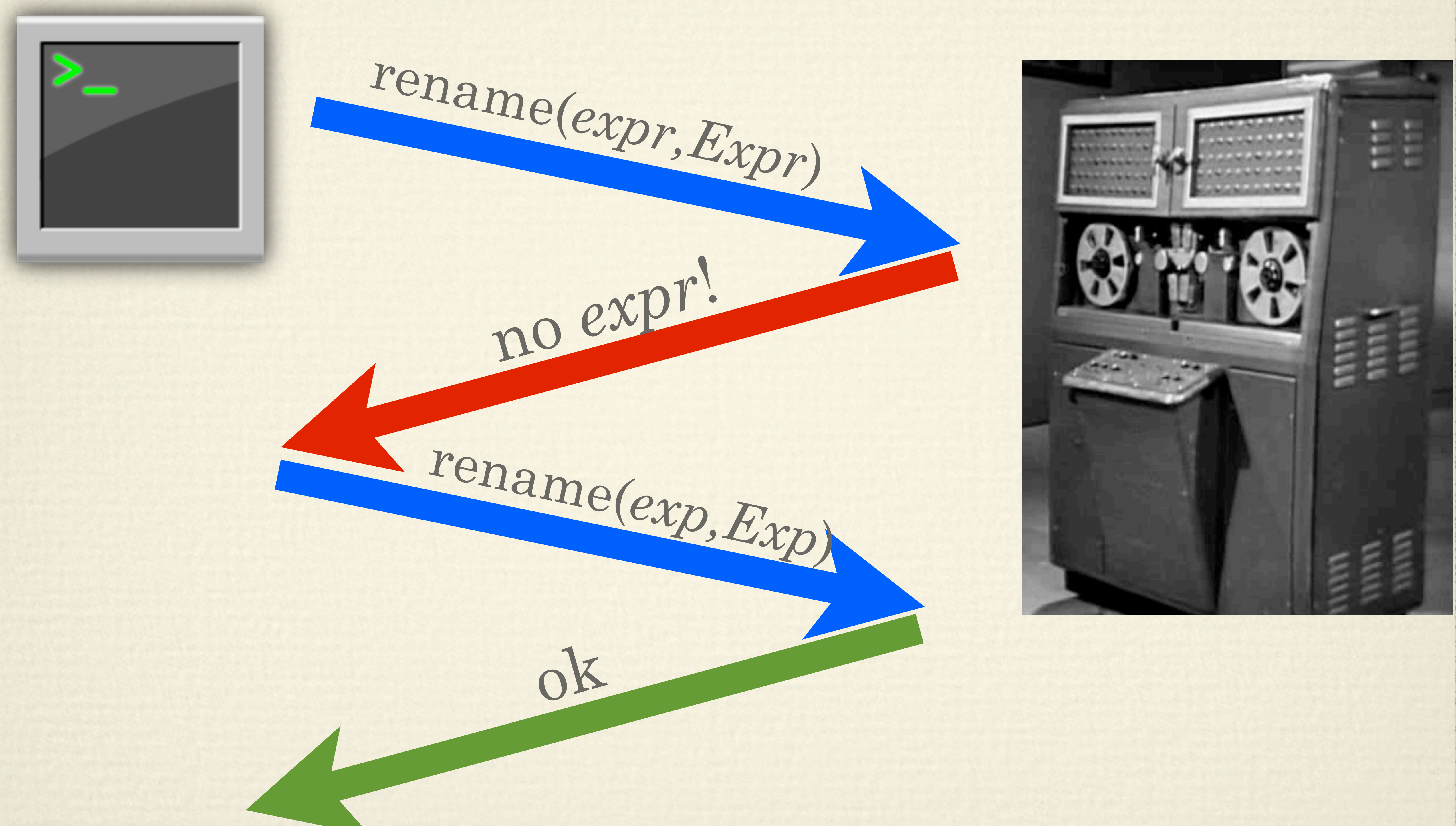
Zaytsev, <u>*Negotiated Grammar Transformation*</u>, XM2012 @ MODELS 2012. JOT.

Rename a nonterminal: failure



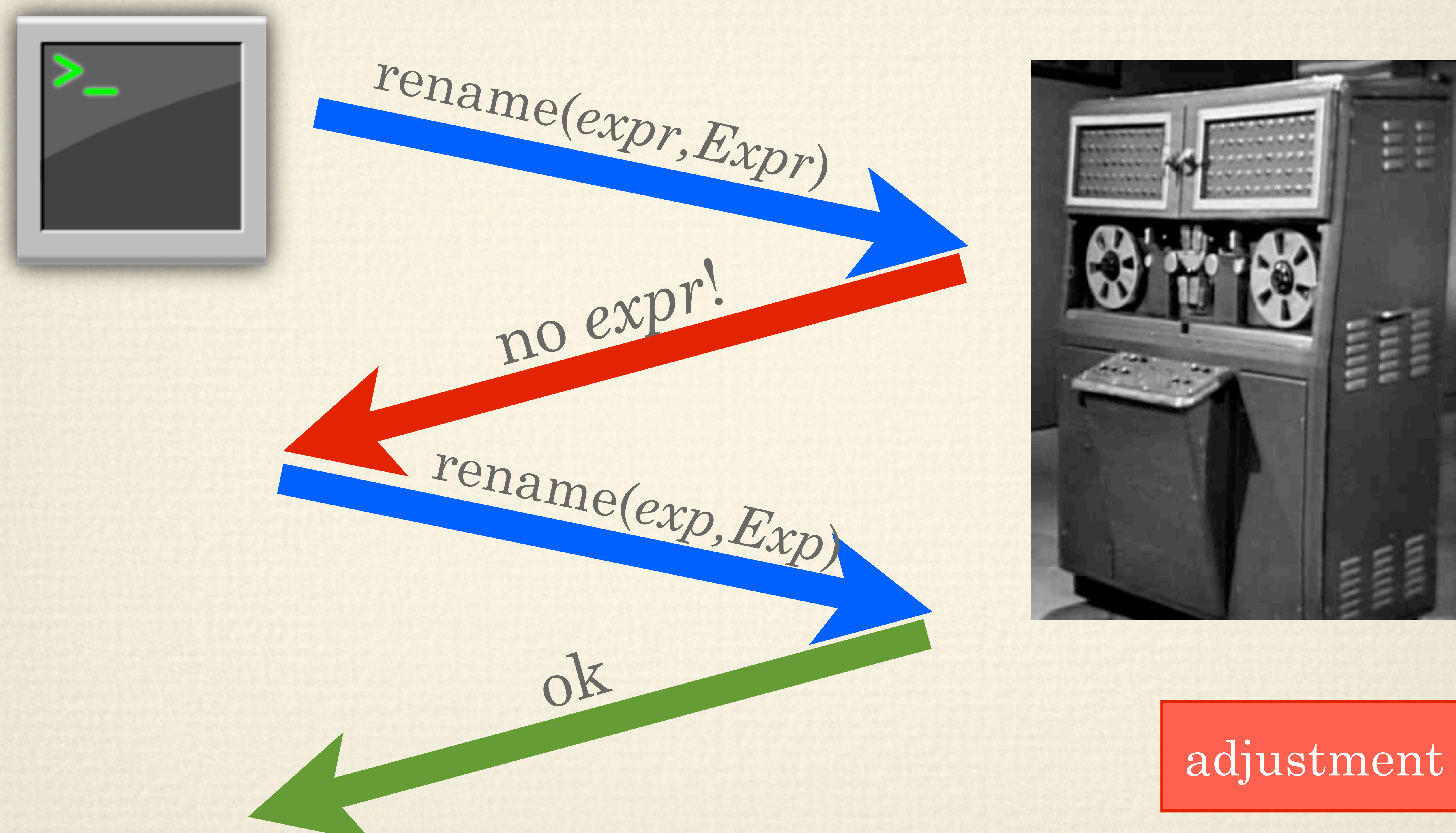
Zaytsev, <u>*Negotiated Grammar Transformation*</u>, XM2012 @ MODELS 2012. JOT.

Rename a nonterminal: negotiation



Zaytsev, <u>*Negotiated Grammar Transformation*</u>, XM2012 @ MODELS 2012. JOT.

Rename a nonterminal: negotiation



adjustment

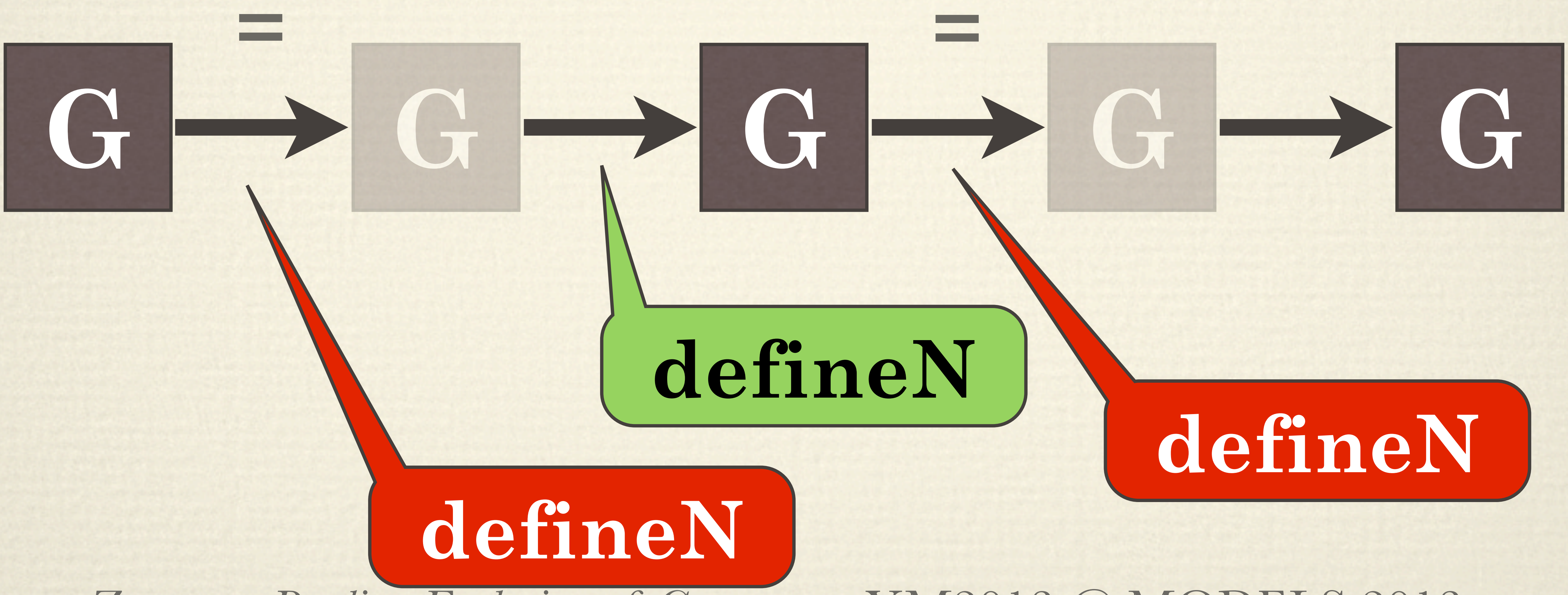
Zaytsev, _Negotiated Grammar Transformation_, XM2012 @ MODELS 2012. JOT.

KEEP
CALM
AND
CONSIDER
IT
DONE

Pending evolution

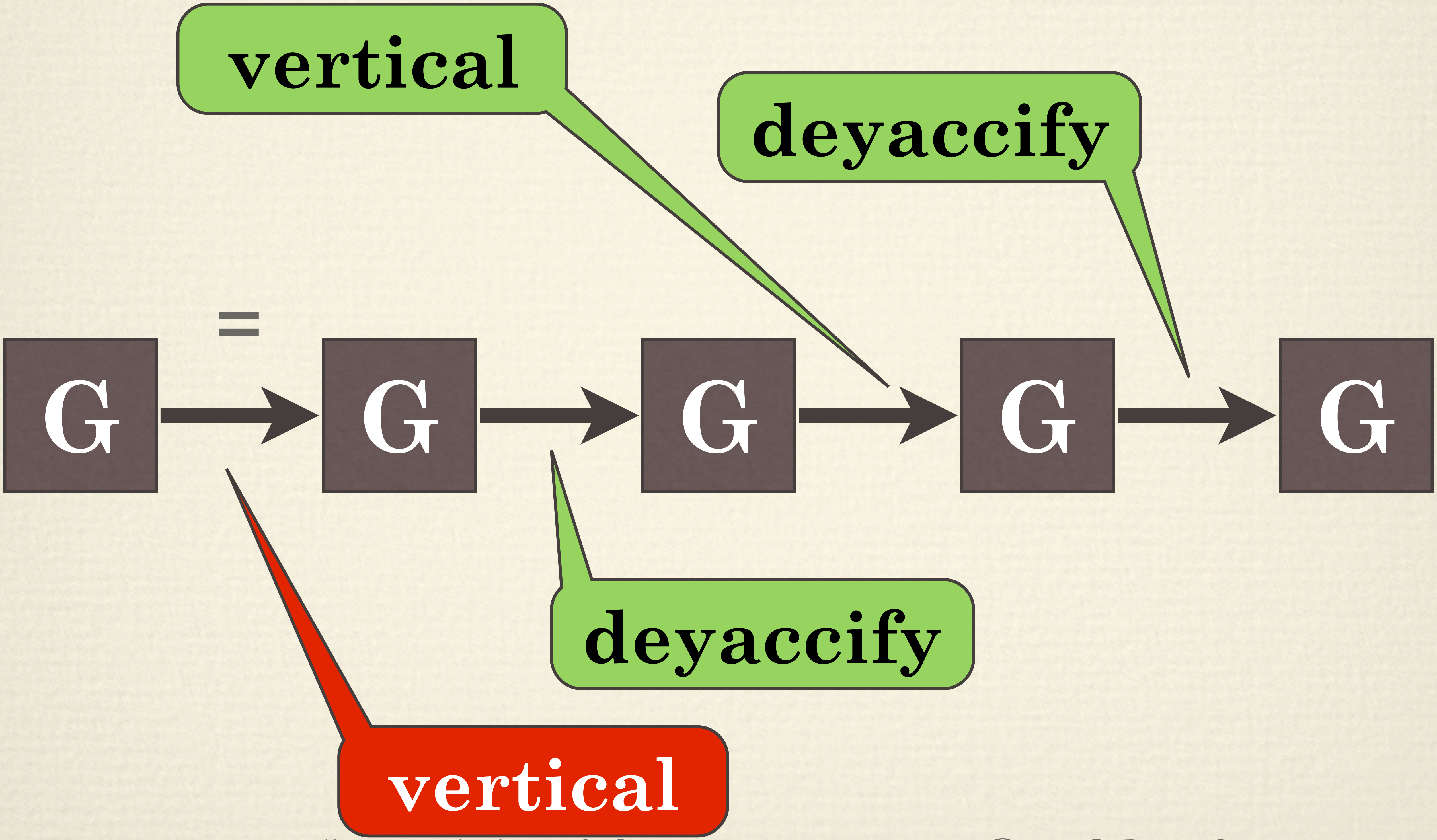
Grammar programming

Optional execution



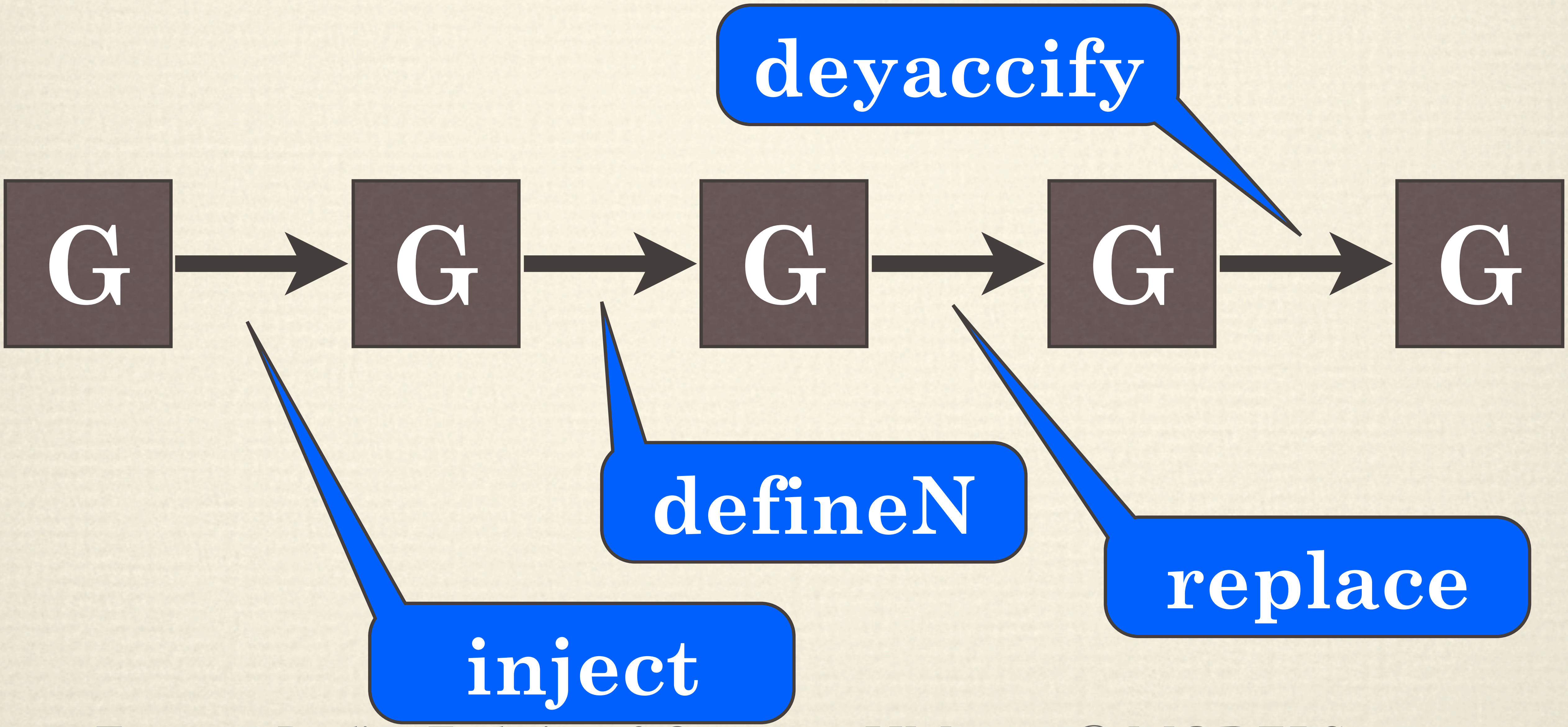
Zaytsev, *Pending Evolution of Grammars*, XM2013 @ MODELS 2013.

Asserting preconditions



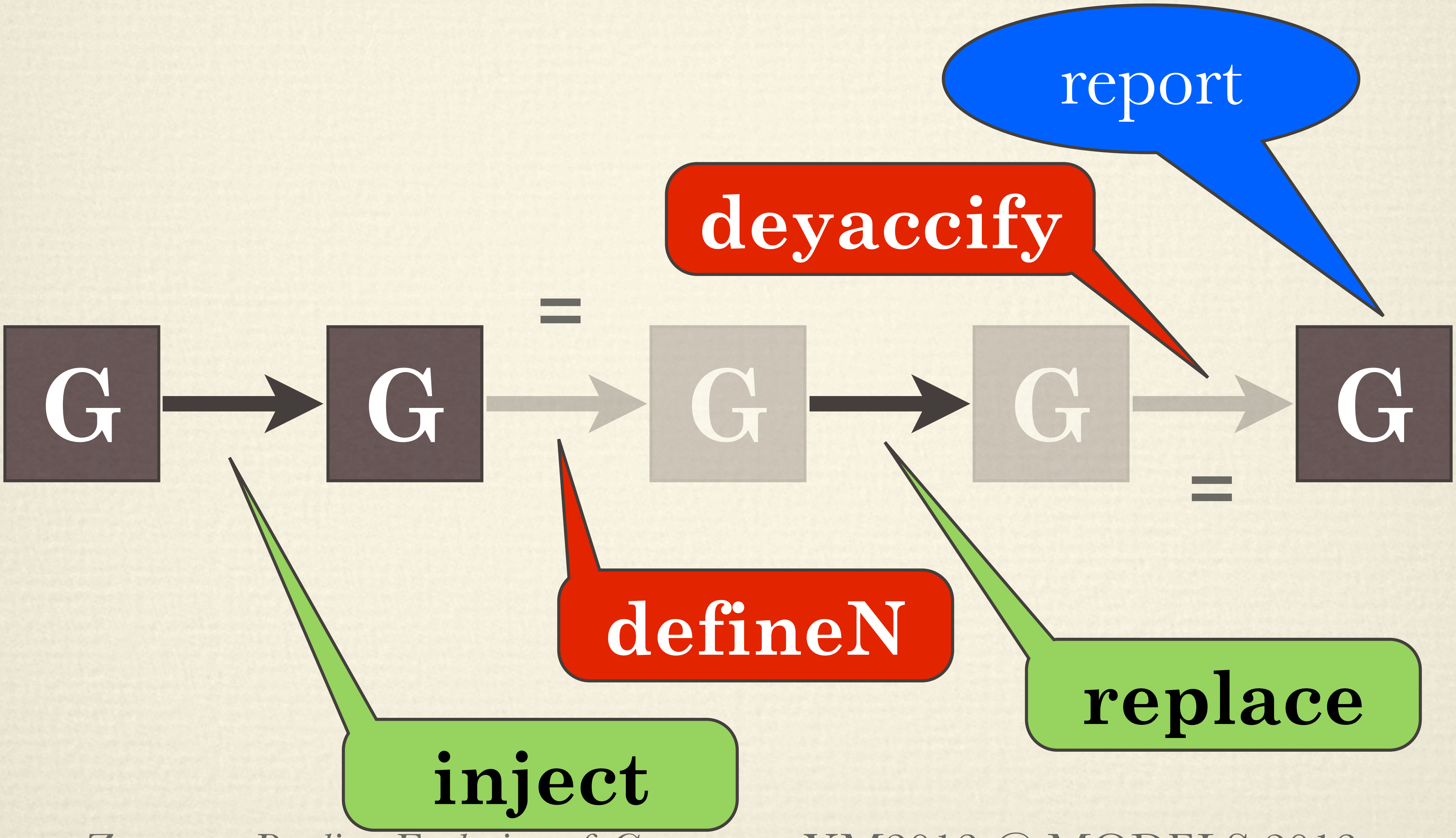
Zaytsev, *Pending Evolution of Grammars*, XM2013 @ MODELS 2013.

Reuse correction scripts



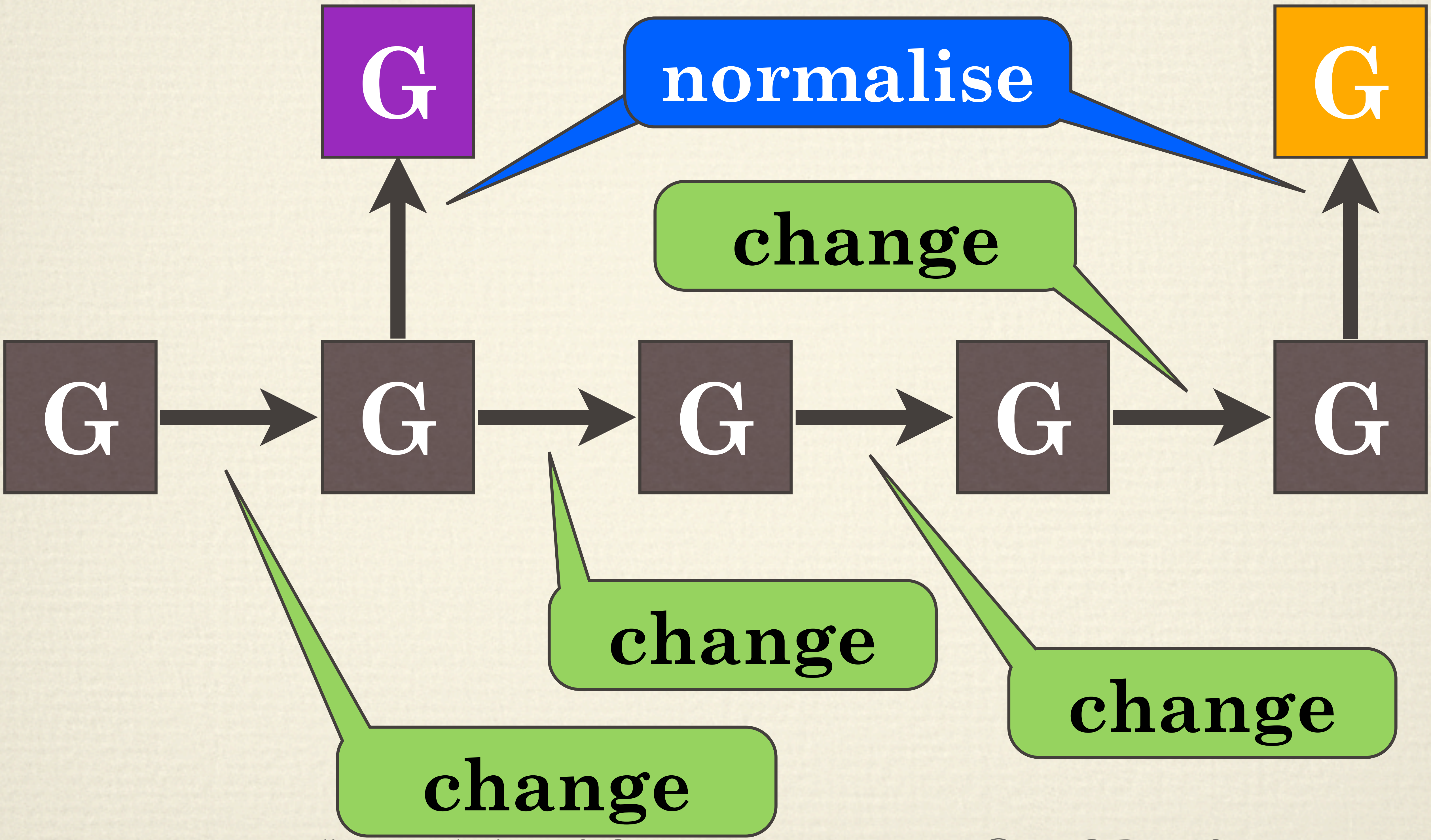
Zaytsev, *Pending Evolution of Grammars*, XM2013 @ MODELS 2013.

Reuse correction scripts



Zaytsev, *Pending Evolution of Grammars*, XM2013 @ MODELS 2013.

Normalise before export



Zaytsev, *Pending Evolution of Grammars*, XM2013 @ MODELS 2013.

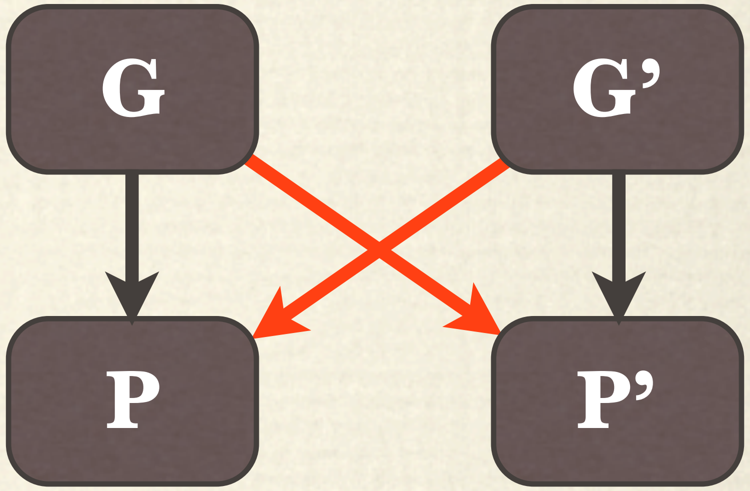
Operator-based evolution analysis

	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

Grammar testing

Grammar-based testing

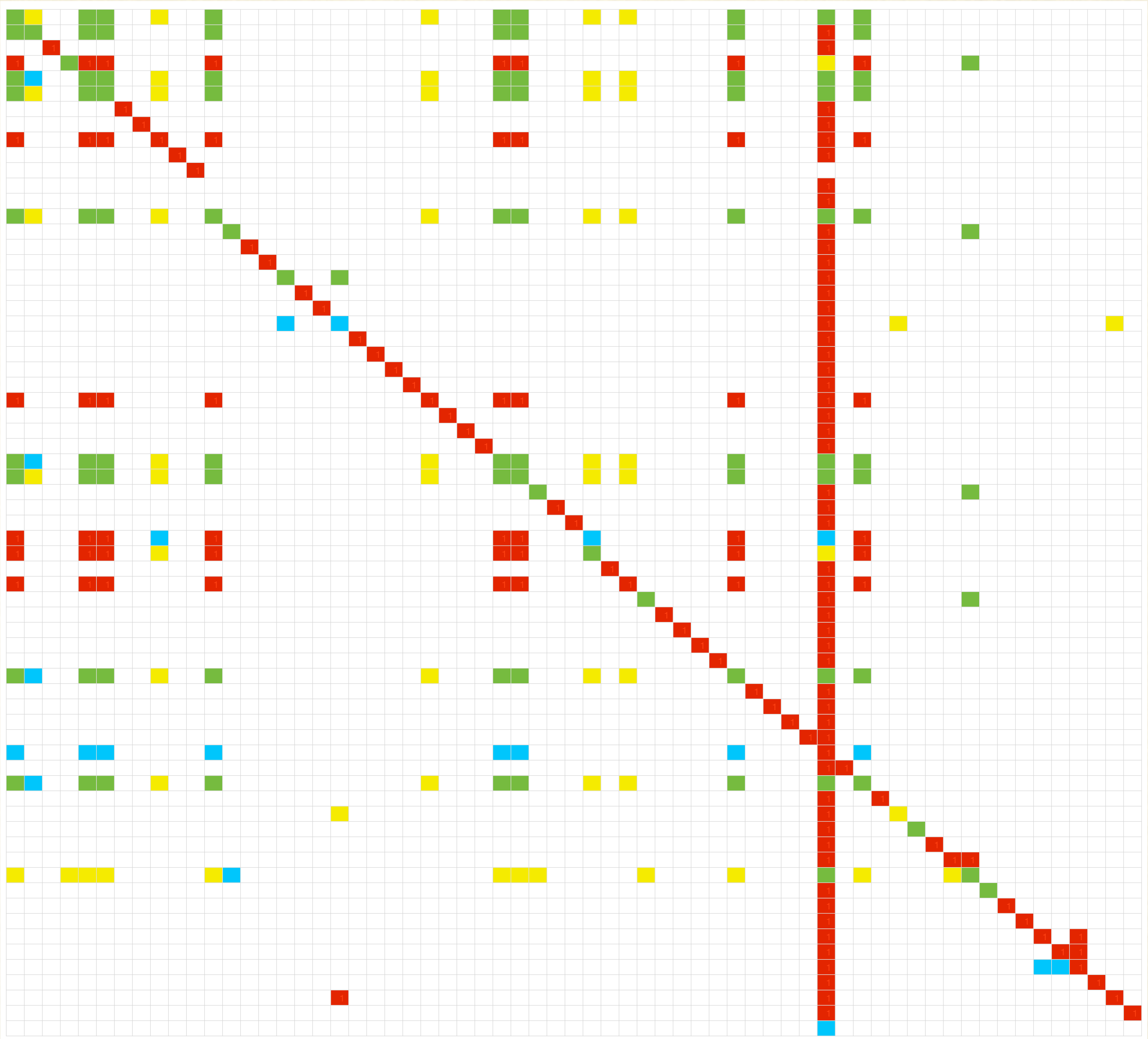
- Asymmetric comparison:
  - Reference grammar vs. parser under test
- Symmetric comparison:
  - Differential testing
- Systematic test data generation
  - Controlled combinatorial coverage
- Larger sets of smaller test data items
- Nonterminal matching
- Non-context-free effects



Coverage criteria

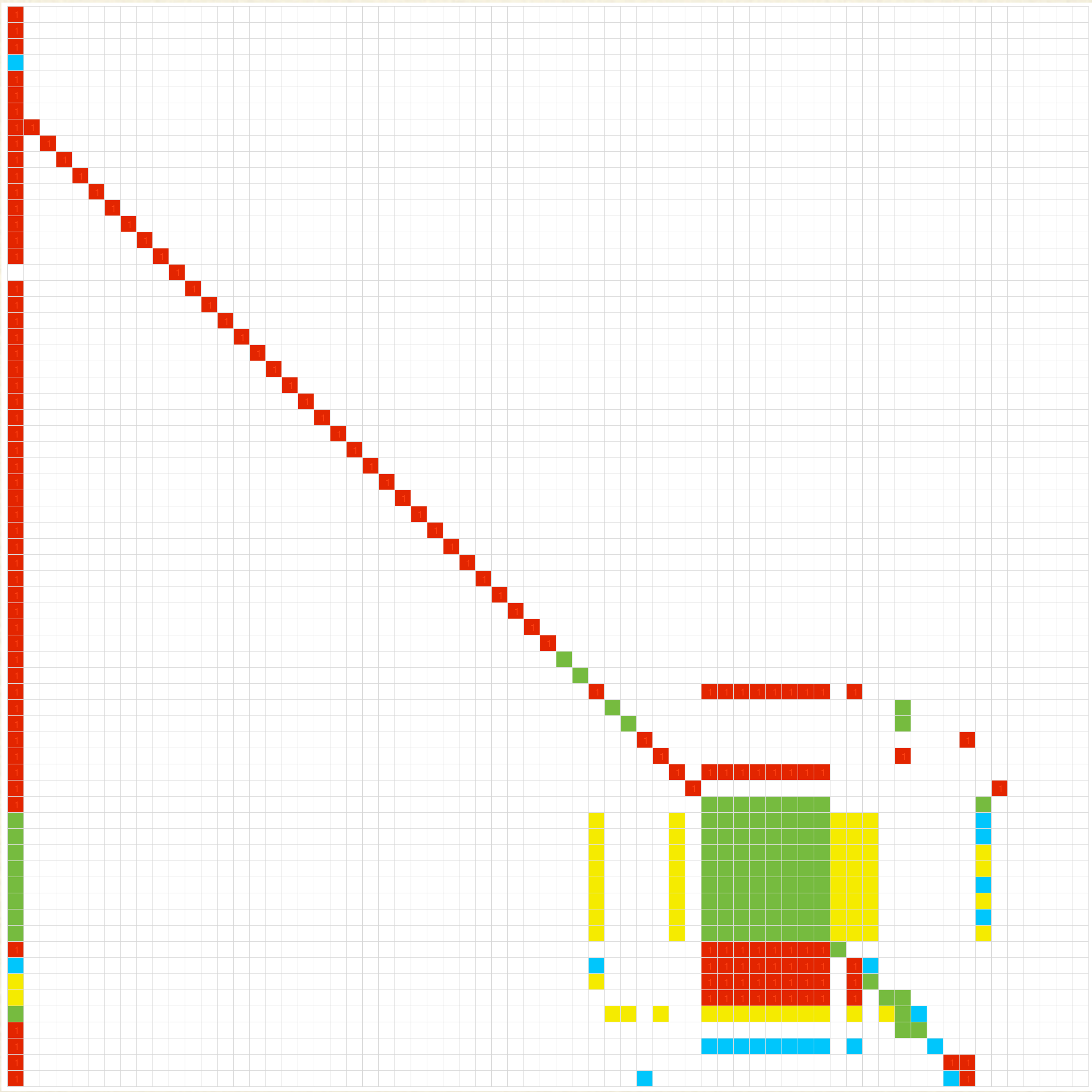
- **Trivial** coverage: if the test data set is not empty.
- **Nonterminal** coverage: if each nonterminal is exercised at least once.
- **Production** coverage: if each production rule in the grammar is exercised at least once.
- **Branch** coverage: each branch of ? | *+
- **Unfolding** coverage: each production of each right hand side nonterminal occurrence
- **Context-dependent branch coverage**!

Nonterminal matching



Nonterminal matching

Nonterminal matching



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

Nonterminal matching

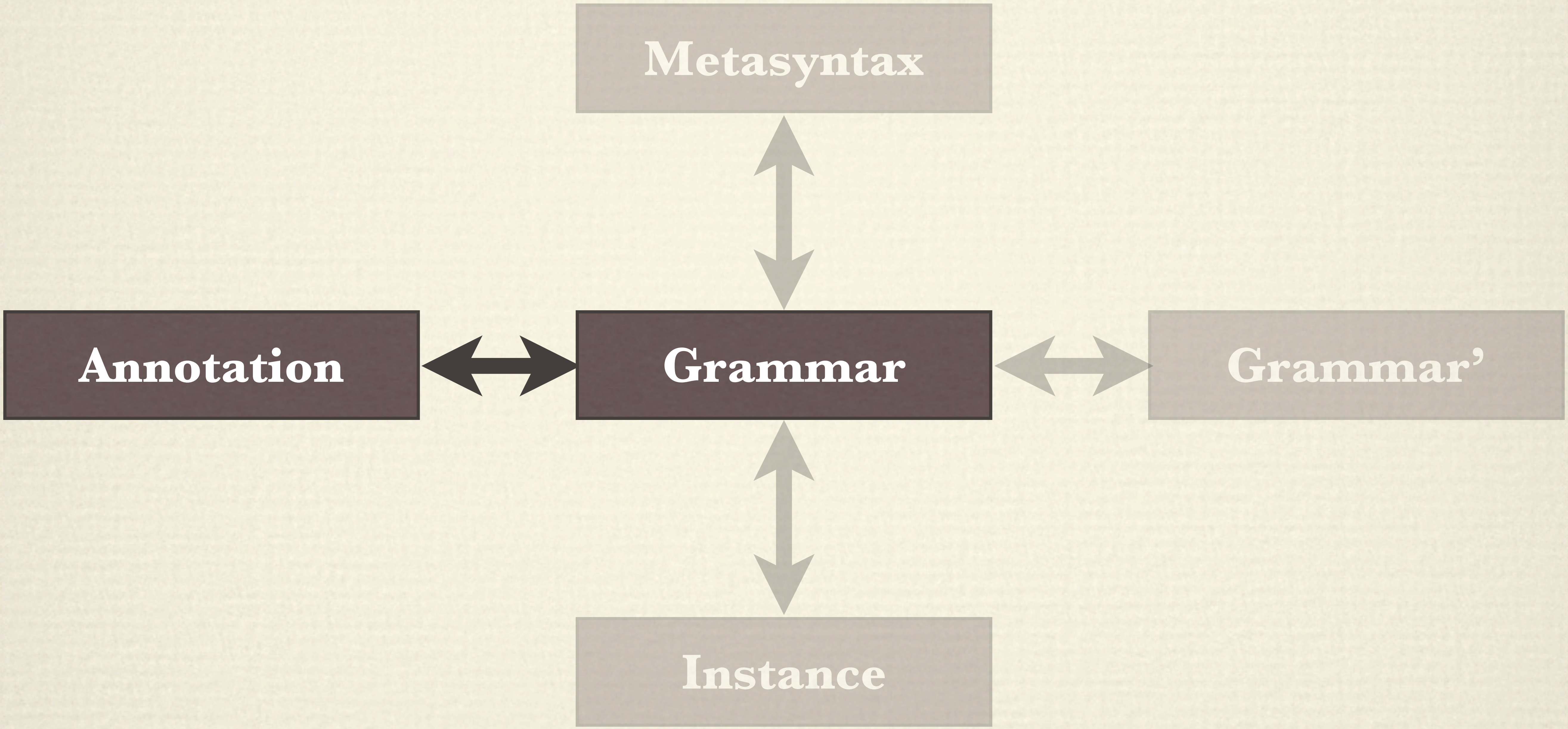
Fischer, Lämmel, Zaytsev, *Comparison of Context-free Grammars Based on Parsing Generated Test Data*, SLE 2011, LNCS 6940, 2012

Language documentation

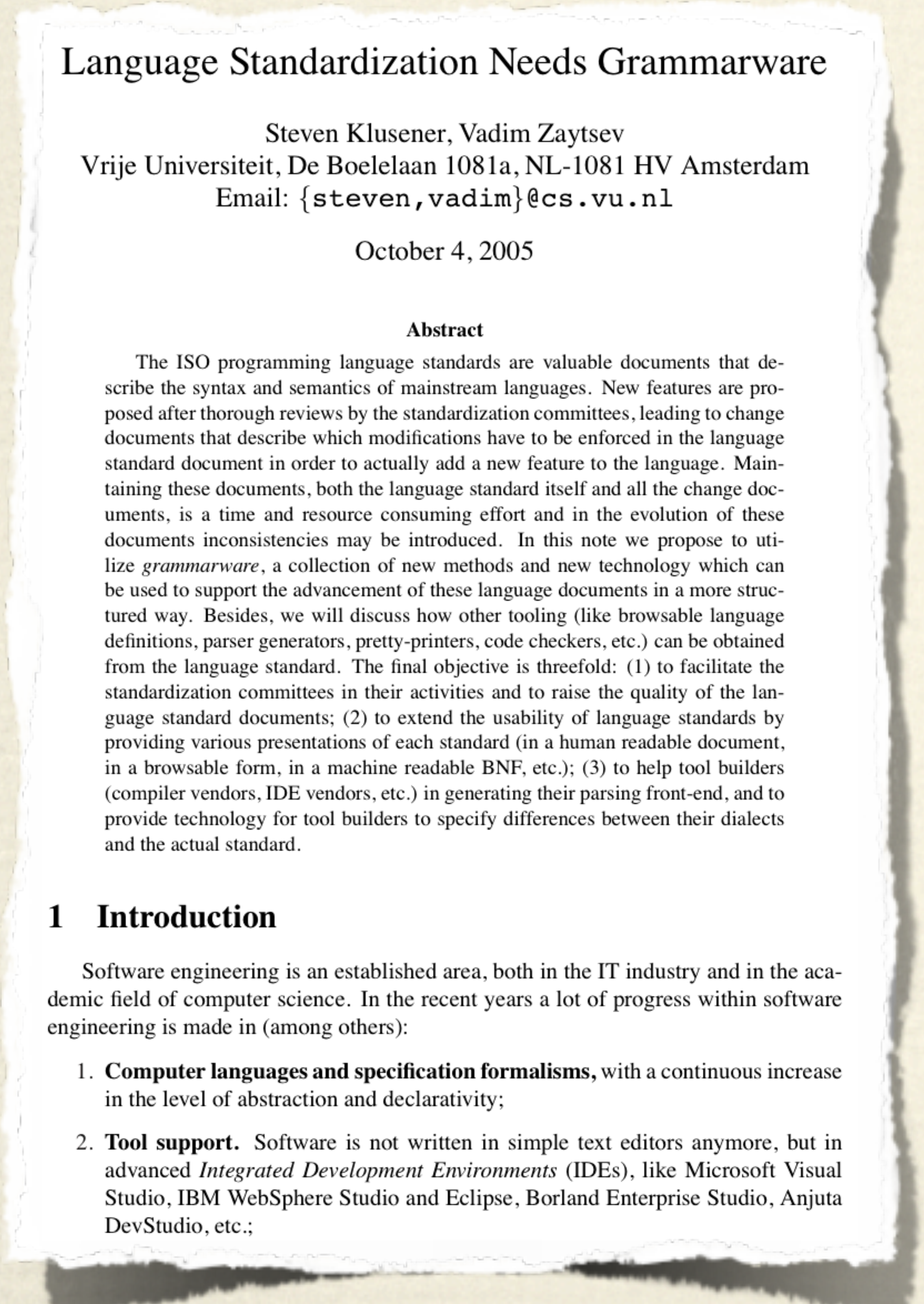
Language documentation

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

Language documentation



Documenting grammars



Language Standardization Needs Grammarware

Steven Klusener, Vadim Zaytsev
Vrije Universiteit, De Boelelaan 1081a, NL-1081 HV Amsterdam
Email: {steven,vadim}@cs.vu.nl

October 4, 2005

Abstract

The ISO programming language standards are valuable documents that describe the syntax and semantics of mainstream languages. New features are proposed after thorough reviews by the standardization committees, leading to change documents that describe which modifications have to be enforced in the language standard document in order to actually add a new feature to the language. Maintaining these documents, both the language standard itself and all the change documents, is a time and resource consuming effort and in the evolution of these documents inconsistencies may be introduced. In this note we propose to utilize *grammarware*, a collection of new methods and new technology which can be used to support the advancement of these language documents in a more structured way. Besides, we will discuss how other tooling (like browsable language definitions, parser generators, pretty-printers, code checkers, etc.) can be obtained from the language standard. The final objective is threefold: (1) to facilitate the standardization committees in their activities and to raise the quality of the language standard documents; (2) to extend the usability of language standards by providing various presentations of each standard (in a human readable document, in a browsable form, in a machine readable BNF, etc.); (3) to help tool builders (compiler vendors, IDE vendors, etc.) in generating their parsing front-end, and to provide technology for tool builders to specify differences between their dialects and the actual standard.

1 Introduction

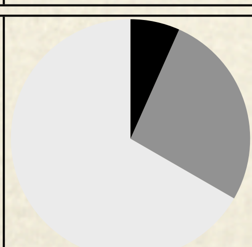
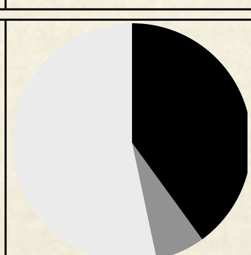
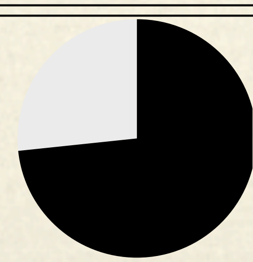
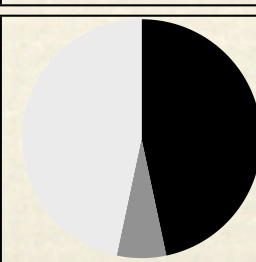
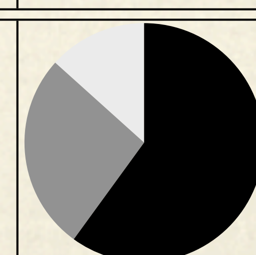
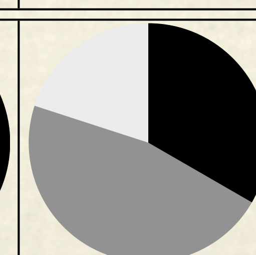
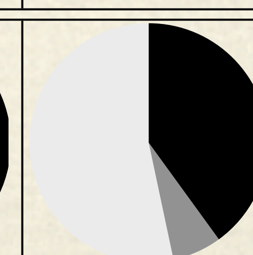
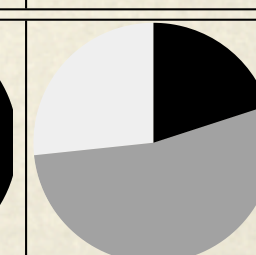
Software engineering is an established area, both in the IT industry and in the academic field of computer science. In the recent years a lot of progress within software engineering is made in (among others):

1. **Computer languages and specification formalisms,** with a continuous increase in the level of abstraction and declarativity;
2. **Tool support.** Software is not written in simple text editors anymore, but in advanced *Integrated Development Environments* (IDEs), like Microsoft Visual Studio, IBM WebSphere Studio and Eclipse, Borland Enterprise Studio, Anjuta DevStudio, etc.;

- IAL, FORTRAN
- ISO/IEC
- ECMA
- W3C
- OMG
- Design Patterns
- …

Klusener, Zaytsev, *Language Stardardisation Needs Grammarware*, ISO/IEC JTC1/SC22 Document N3977, 2005.

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.

Grammar export

Generated Rascal

```
@contributor{BGF2Rascal exporter - SLPS - http://github.com/grammarware/slps/wiki/BGF2Rascal}
module CSharp

import ParseTree;
import util::IDE;
import IO;

layout WS = [\t-\n\r\ ]* !>> [\t-\n\r\ ];

start syntax CompilationUnit = UsingDirectives? GlobalAttributes? NamespaceMemberDeclarations?;

syntax UsingDirectives
        = UsingDirective
        | (UsingDirectives UsingDirective);

syntax UsingDirective
        = UsingAliasDirective
        | UsingNamespaceDirective;

syntax UsingAliasDirective
        = "using" Identifier "=" NamespaceOrTypeName ";";

...
```

Generated SDF

module Main

exports
 context-free start-symbols Compilation-unit
  sorts
      …
  context-free syntax
      Using-directives? Global-attributes? Namespace-member-declarations? → Compilation-unit

      Using-directive → Using-directives
      Using-directives Using-directive → Using-directives

      Using-alias-directive → Using-directive
      Using-namespace-directive → Using-directive

      "using" Identifier "=" Namespace-or-type-name ";" → Using-alias-directive

…

Browsable grammars

Browsable C# 1.x Grammar

Extracted and/or recovered by **Vadim Zaytsev**, see Grammar Zoo for details.

Source used for this grammar: *ECMA-334*, December 2001, Appendix A, pages 339–364

Summary

- Number of production rules: **240**
- Number of top alternatives: **584**
- Number of defined nonterminal symbols: **240**
- Root nonterminal symbols: —
- Other top nonterminal symbols: **1**: `compilation-unit`
- Bottom nonterminal symbols: **2**: `identifier`, `literal`
- Number of used terminal symbols: **126**
- Special terminal symbols: **44**: ".", "[", "]", ",", "(", ")", "++", "--", "+", "-", "!", "~", "*", "/", "%", "<<", ">>", "<", ">", "<=", ">=", "==", "!=", "&", "^", "|", "&&", "||", "?", ":", "=", "+=", "-=", "*=", "/=", "%=", "&=", "|=", "^=", "<<=", ">>=", "{", "}", ";"
- Keywords: **82**: "bool", "decimal", "sbyte", "byte", "short", "ushort", "int", "uint", "long", "ulong", "char", "float", "double", "object", "string", "ref", "out", "this", "base", "new", "typeof", "void", "checked", "unchecked", "is", "as", "const", "if", "else", "switch", "case", "default", "while", "do", "for", "foreach", "in", "break", "continue", "goto", "return", "throw", "try", "catch", "finally", "lock", "using", "namespace", "class", "public", "protected", "internal", "private", "abstract", "sealed", "static", "readonly", "volatile", "virtual", "override", "extern", "params", "get", "set", "event", "add", "remove", "operator", "true", "false", "implicit", "explicit", "struct", "interface", "enum", "delegate", "assembly", "field", "method", "module", "param", "property"

Syntax

```
namespace-name:
        namespace-or-type-name
```

```
type-name:
        namespace-or-type-name
```

http://slps.github.io/zoo/cs/csharp-ecma-334-1.html

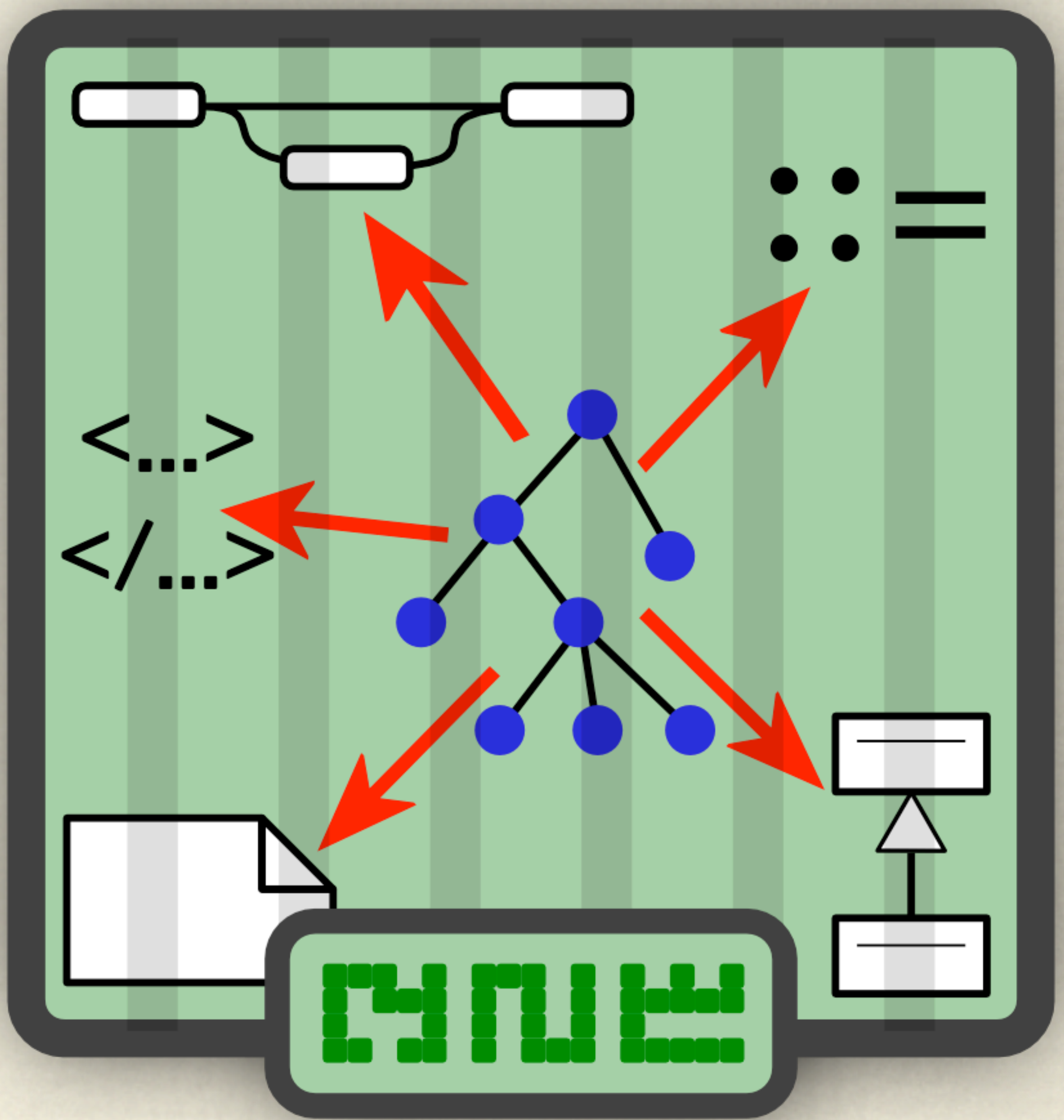
Grammar analysis

Grammar-based metrics

- PROD, VAR, TERM, …
- VOC, LEN, EFF, PLEV, LLEV, …
- LEV, CLEV, RLEV, NLEV, …
- cf.
  - J. F. Power, B. A. Malloy. *A Metrics Suite for Grammar-Based Software*. 2004.
  - T. Alves, J. Visser. *Metrication of SDF Grammars*. 2005.
  - J. Cervelle, M. Črepinšek, R. Forax, T. Kosar, M. Mernik, G. Roussel. *On Defining Quality Based Grammar Metrics*. 2009.
  - M. Črepinšek, T. Kosar, M. Mernik, J. Cervelle, R. Forax, G. Roussel. *On Automata and Language Based Grammar Metrics*. 2010.

Micropatterns

- ✦ Recognisable
- ✦ Purposeful
- ✦ Prevalent
- ✦ Simple
- ✦ Scoped
- ✦ Observed



Zaytsev, *Micropatterns in Grammars*. SLE 2013

To summarise

- Extraction
  - recovery
- Evolution
  - transformation
  - negotiated
  - pending
- Documentation
- Export

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

Questions?

(What else would you like?)

vadim@grammarware.net