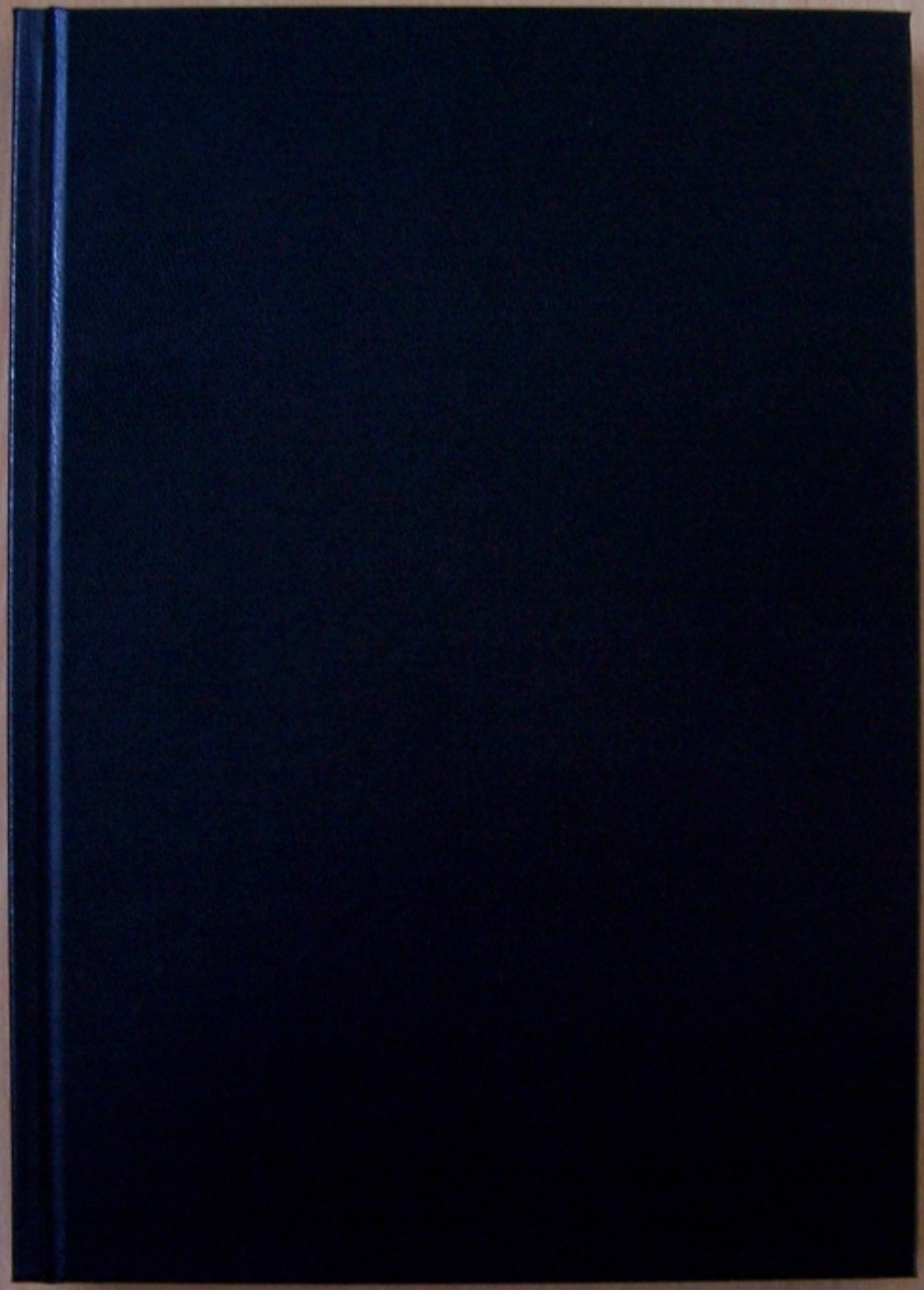
Grammarware Applications to Recovery, Convergence and Documentation of Languages

Doctoral presentation of
Vadim V. Zaytsev
2010



Thesis tag cloud

abstract (78) approach (54) argument (52) artefacts (62) automated (59) bar (61) bgf (115) binary (60) bnf (64) case (201) change (53) chapter (147) code (93) concrete (52) contains (69) convergence (278) correction (63) corresponding (54) data (61) defined (126) definition (232) detail (66) different (152) document (130) engineering (97) example (236) existing (71) expr (230) expression (216) extraction (143) figure (52) foo (53) form (73) formal (76) format (73) generated (82) given (88) grammar (1375) grammarware (54) infrastructure (55) input (73) instance (62) int (64) iso (69) java (84) jls (58) language (783) ldf (76) list (116) manual (80) model (87) name (106) needed (62) nonterminal (291) number (62) op (90) operators (169) order (54) parser (77) parsing (108) possible (73) presented (59) process (77) production (283) programming (118) recovery (128) refactoring (67) reference (57) research (56) result (79) rules (53) schema (75) scope (55) sdf (58) section (241) semantics (73) simple (58) software (84) source (78) specification (110) standard (150) step (117) str (97) structure (103) study (109) subsection (52) suite (54) symbol (89) syntax (202) table (72) terminal (111) thesis (62) tools (56) transformation (334) type (59) used (275) version (87) work (110) xbgf (149) xml (84)

Recovery, Convergence and Documentation of Languages

by
Vadim Zaytsev

September 14, 2010

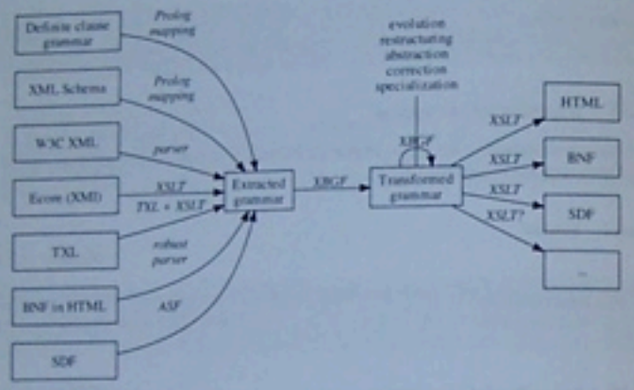


Figure 6.2: The life cycle of a language grammar in the transformational environment: from a grammar knowledge possessing software artefact on the left to the usable working grammar on the right.

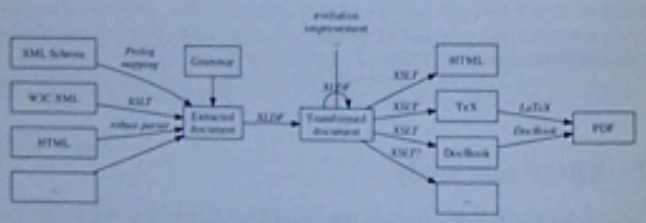


Figure 6.3: The life cycle of a language document in the prototype language document infrastructure: from the structure source on the left to the extracted document gradually transformed to its ultimate form and finally pretty-printed for presentation on the right.

language improvement, any grammar maintenance activities, etc) and then the final form is computed.

In the prototype of this chapter we started with an XML Schema definition. We have the tools to map definitions of XML elements, groups and other entities to grammar productions, for which the extractor from chapter 4 is reused. We also developed new tools to map XSD annotations to LDF textual paragraphs. Once an LDF document is ready, one can use XLDF commands to transform it. These commands can utilise secondary sources of information such as test suites to fill in the gaps in the language documentation. Transformations written in XLDF can take this LDF as an input and allow for adaptation, evolution, beautification, etc, as discussed earlier. Eventually the LDF document is considered ready for presentation, and a range of generator tools allow to make a PDF file out of it, a TEX source or an HTML web page.

6.7.1 Extraction

For us the central part of any language document is the grammar behind it. At the point when we started composing the XBGF manual, the grammar of XBGF has already been specified by an XML Schema definition `shared/xsd/xbgf.xdf` in SLPS [263]. This schema was not used directly in parsing by the Prolog program that handled the transformations, but validation checks were performed with it.

XSD to BGF mapping has also already been established as a part of FL case study—see section 4.3 and Listing 4.10. We needed only to extend it to design XSD to LDF mapping. It was decided that every XSD construct that defined a schema entity should be mapped to a separate top-level section of a language document. Those constructs were: XML elements, XML attributes, named content types, groups and attribute groups—each of them was mapped to a nonterminal symbol for BGF and to a section describing this nonterminal for LDF.

In XSD it is possible to annotate any construct with a piece of text, and that feature was extensively utilised during schema development phase to provide comments for XBGF operators. With `xsd:annotation` and `xsd:documentation` tags we basically inserted typical language documentation information right into the schema. The idea came naturally to map such annotations to the textual content of the corresponding sections of the language manual.

Two front matter sections were decided to be filled differently: foreword and normative references. The top level annotations (those assigned to the whole document and not to a specific definition) were mapped to foreword and the list of imported XML Schema definitions became normative references. After filling out details like the document title and author we were ready to produce a correct LDF document for any given XSD.

6.7.2 Transformation

Since the structure of the language document generated by XSD to LDF extractor was very simple and too straightforward, we needed document transformation steps to reorder the sections, to add lacking textual content, to connect and pretty-print samples, etc. The transformation suite explained in section 6.6 was used for that.

lijke taal) kan beschrijven. Zo wordt het mogelijk om een document op semi-automatische wijze te verbeteren, te verifiëren, aan te passen of te herstructureren. Ook wordt het mogelijk semi-automatisch een PDF- of HTML-versie van een document te genereren.

De voornaamste contributies van dit proefschrift zijn de volgende:

- Het stappenplan voor herwinning van een grammatica en andere inzichten op dat gebied — zie [257] en Hoofdstuk 3.
- De lichtgewicht verificatietechniek genaamd "grammaticale convergentie" — zie [166, 167, 168, 258, 259] en Hoofdstukken 4–5.
- De ontwikkeling van de grammaticale onttrekkers, met name de regel-gebaseerde — zie [168] en Hoofdstuk 5.
- De 18 verschillende grammatica's geproduceerd door deze onttrekkers — zie [260].
- De gedetailleerde analyse van meer dan 40 huidige taalstandaarden en taalhandboeken — zie [262] en Hoofdstuk 6.
- Het datamodel voor het taalspecificatiedomein — zie [262] en Hoofdstuk 6.
- Het opstellen en het prototyperen van de taaldocumentatie infrastructuur — zie [143, 258, 259] en Hoofdstuk 6.
- De 6 domein-specifieke talen voor grammarware en de door onze infrastructuur geproduceerde taaldocumenten voor hen — zie [258, 259, 261] en Hoofdstukken 6–7.
- De krachtige set operatoren voor grammaticale transformaties — zie [168, 261] en Hoofdstuk 7.

Met uitzondering van online documenten, zijn er in totaal acht publicaties op basis van dit proefschrift, waarvan één journal paper [168], één ISO document [143], twéé extended abstracts [257, 258] en vier proceedings papers [166, 167, 259, 262].

Bibliography

[1] AcuCorp Inc. *AcuCobol-85 Reference Manual*, 1999.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers. Principles, Techniques and Tools*. Addison-Wesley, 1985.

[3] J. Albert, D. Giammarresi, and D. Wood. Normal Form Algorithms for Extended Context-free Grammars. *Theoretical Computer Science*, 267(1-2):35–47, 2001.

[4] E. Allen. *Bug Patterns in Java*. APress L. P., 2002.

[5] T. L. Alves, P. F. Silva, J. Visser, and J. N. Oliveira. Strategic Term Rewriting and Its Application to a VDMSL to SQL Conversion. In *FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005, Proceedings*, volume 3582 of *LNCS*, pages 399–414. Springer, 2005.

[6] T. L. Alves and J. Visser. A Case Study in Grammar Engineering. In *Post-proceedings of 1st International Conference on Software Language Engineering (SLE'08)*, LNCS. Springer, 2009. To appear.

[7] American National Standards Institute. `www.ansi.org`. Accessed in June 2009.

[8] J. Axelsson, M. Birbeck, M. Dubinko, B. Epperson, M. Ishikawa, S. McCarron, A. Navarro, and S. Pemberton. XHTML™2.0. *W3C Working Draft*, 26 July 2006. `www.w3.org/TR/2006/WD-xhtml2-20060726`.

[9] J. W. Backus. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. In S. de Picciotto, editor, *Proceedings of the International Conference on Information Processing*, pages 125–131, Unesco, Paris, 1960.

[10] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised Report on the Algorithmic Language ALGOL 60. *Numerische Mathematik*, 4:420–453, Springer-Verlag, Berlin, Heidelberg, New York, 1963. International Federation for Information Processing 1962. Edited by Peter Naur.

[11] J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, H. L. Herrick, R. A. Hughes, L. B. Mitchell, R. A. Nelson, R. Nutt, D. Sayre, P. B. Sheridan, H. Stern, and I. Ziller. *The Fortran Automatic Coding System for the IBM 704 EDPM. Programmer's Reference Manual*. Applied Science Division and Programming Research Department, International Business Machines Corporation, 590 Madison Ave., New York 22, NY, October 15 1956.

[12] D. T. Barnard. Syntax Error Handling Techniques. Technical Report Technical Report 81-125, Queen's University, Department of Computing and Information Science, September 1981. 23 pages.

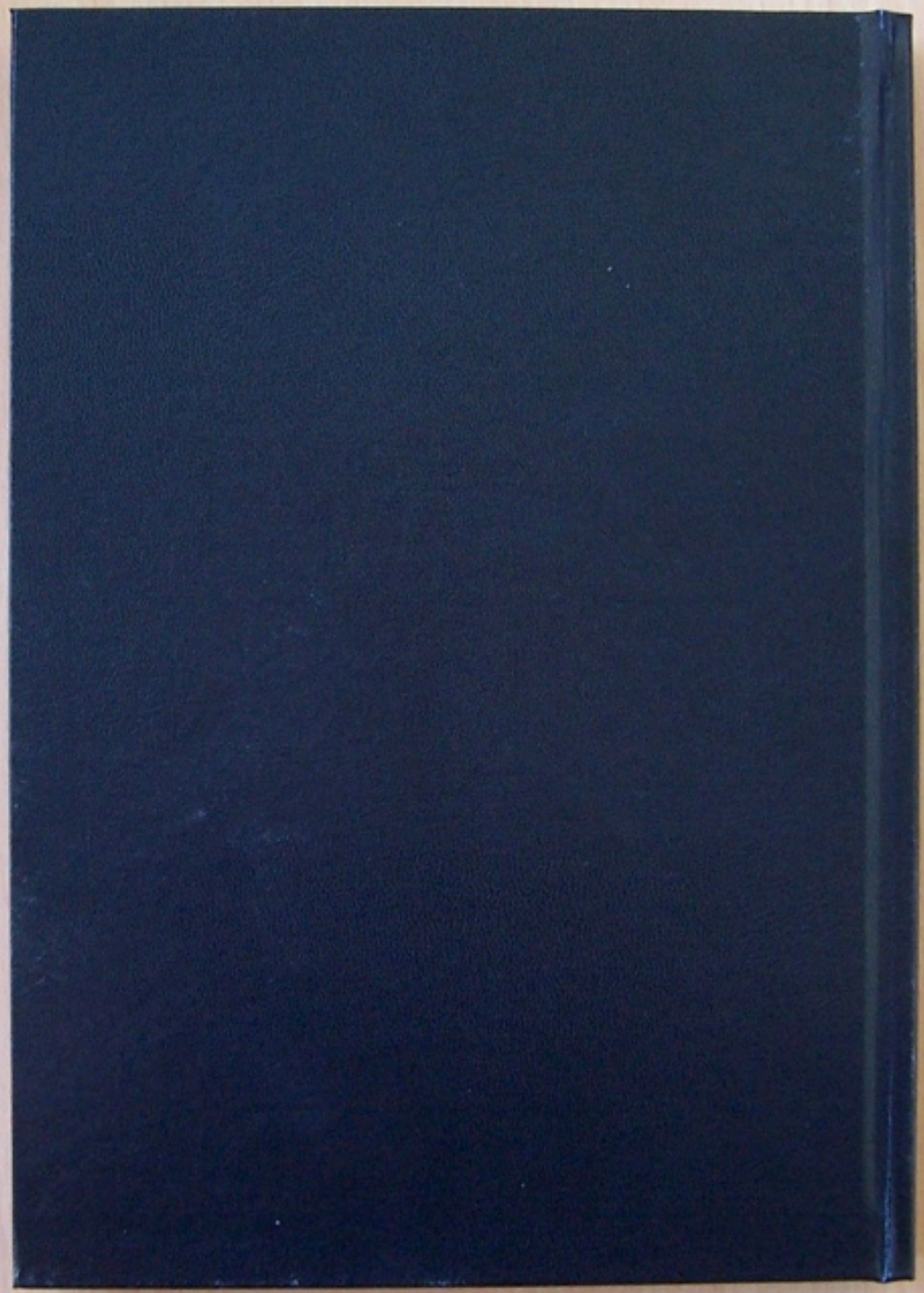
[13] D. T. Barnard and R. C. Holt. Hierarchic Syntax Error Repair for LR Grammars. *International Journal of Computer and Information Sciences*, 11(4):231–258, 1982.

[14] P. Berdaguer, A. Cunha, H. Pacheco, and J. Visser. Coupled Schema Transformation and Data Conversion for XML and SQL. In *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007, Nice, France, January 14-15, 2007*, volume 4354 of *LNCS*, pages 290–304. Springer, 2007.

[15] A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. *W3C Recommendation*, 23 January 2007. `www.w3.org/TR/2007/REC-xpath20-20070123`.

Bibliography

- ★ Vadim Zaytsev,
  Correct C♯ Grammar too Sharp for ISO, GTTSE 2005
- ★ Steven Klusener, Vadim Zaytsev,
  Language Standardization Needs Grammarware, ISO, 2005
- ★ Ralf Lämmel and Vadim Zaytsev,
  An Introduction to Grammar Convergence, iFM 2009
- ★ Vadim Zaytsev,
  Language Convergence Infrastructure, GTTSE 2009
- ★ Ralf Lämmel and Vadim Zaytsev,
  Recovering Grammar Relationships for the JLS, SCAM 2009 & SQJ
- ★ Ralf Lämmel and Vadim Zaytsev,
  Reverse Engineering Grammar Relationships, WSR 2010
- ★ Vadim Zaytsev and Ralf Lämmel,
  A Unified Format for Language Documents, SLE 2010



The End

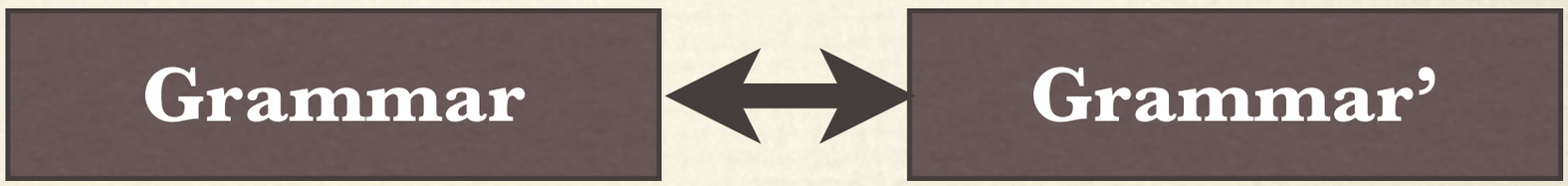
What to expect

- ✓ Context of grammar engineering
- ✓ Metamodels & paradigms
- ✓ Lots of visualisations
- ✓ Reports on actual achievements
- ✓ ???
- ✓ Profit!

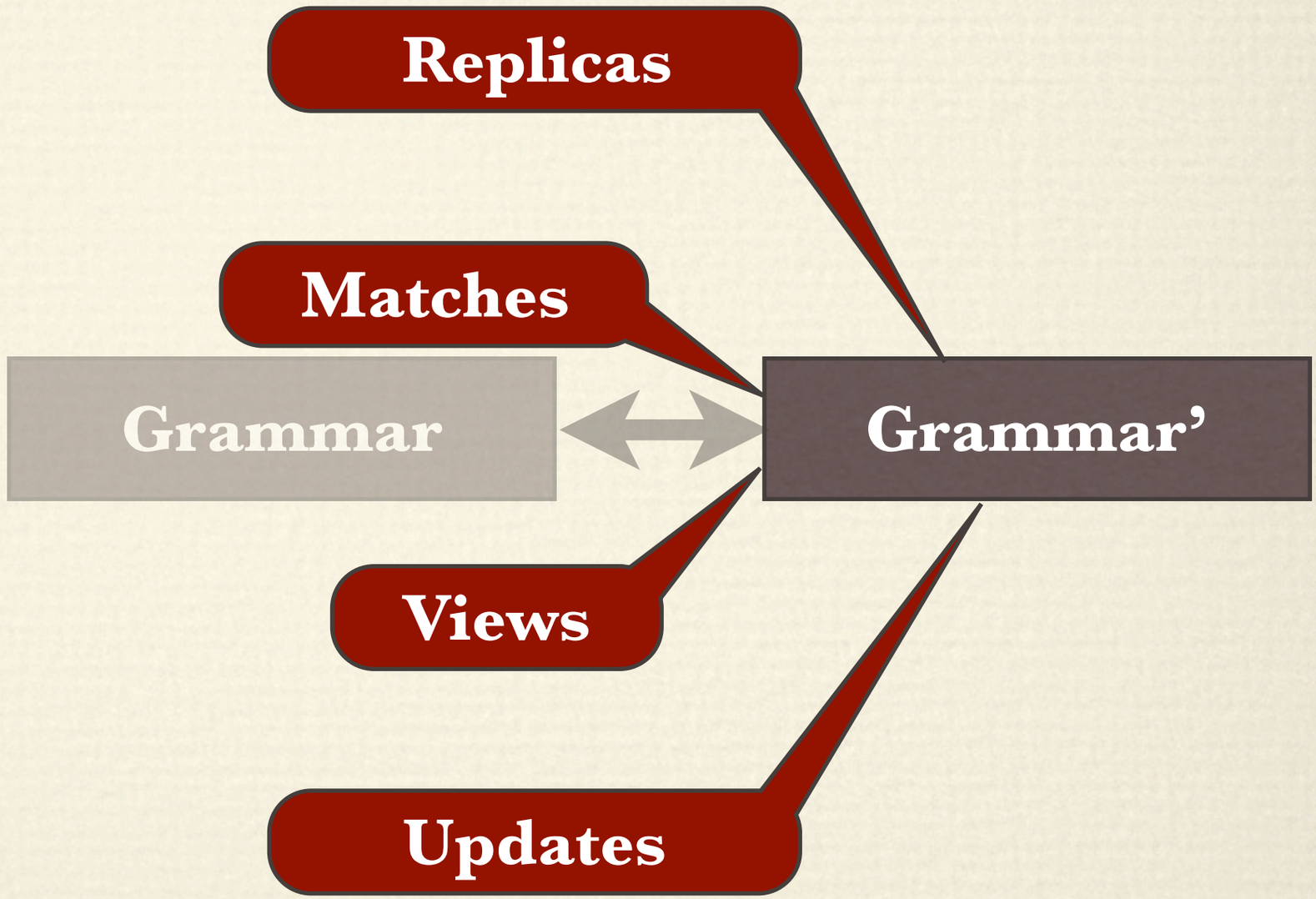
Grammar world

Grammar

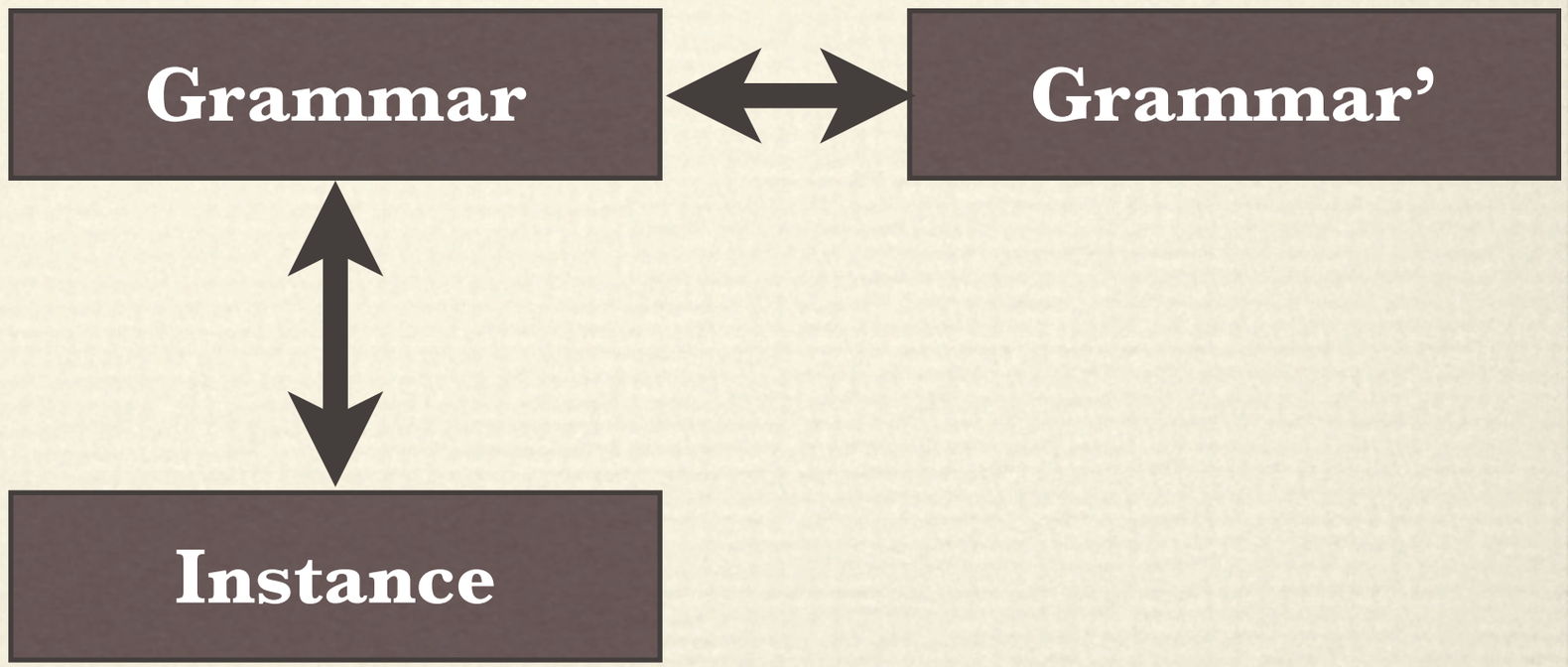
Grammar world



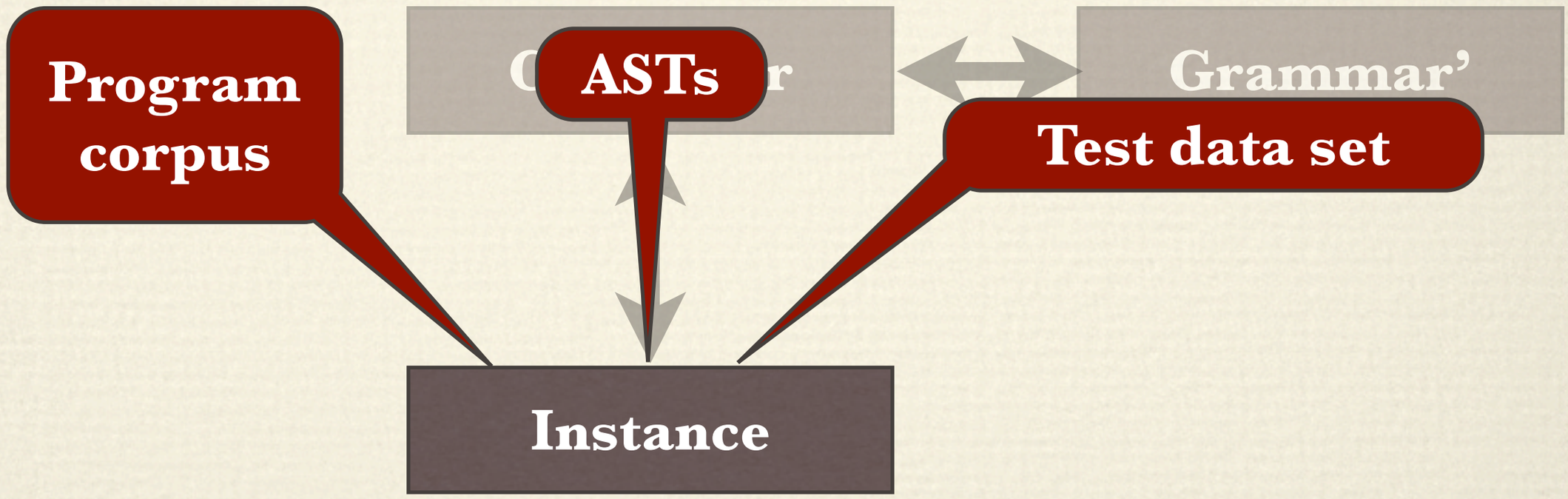
Grammar world: grammars



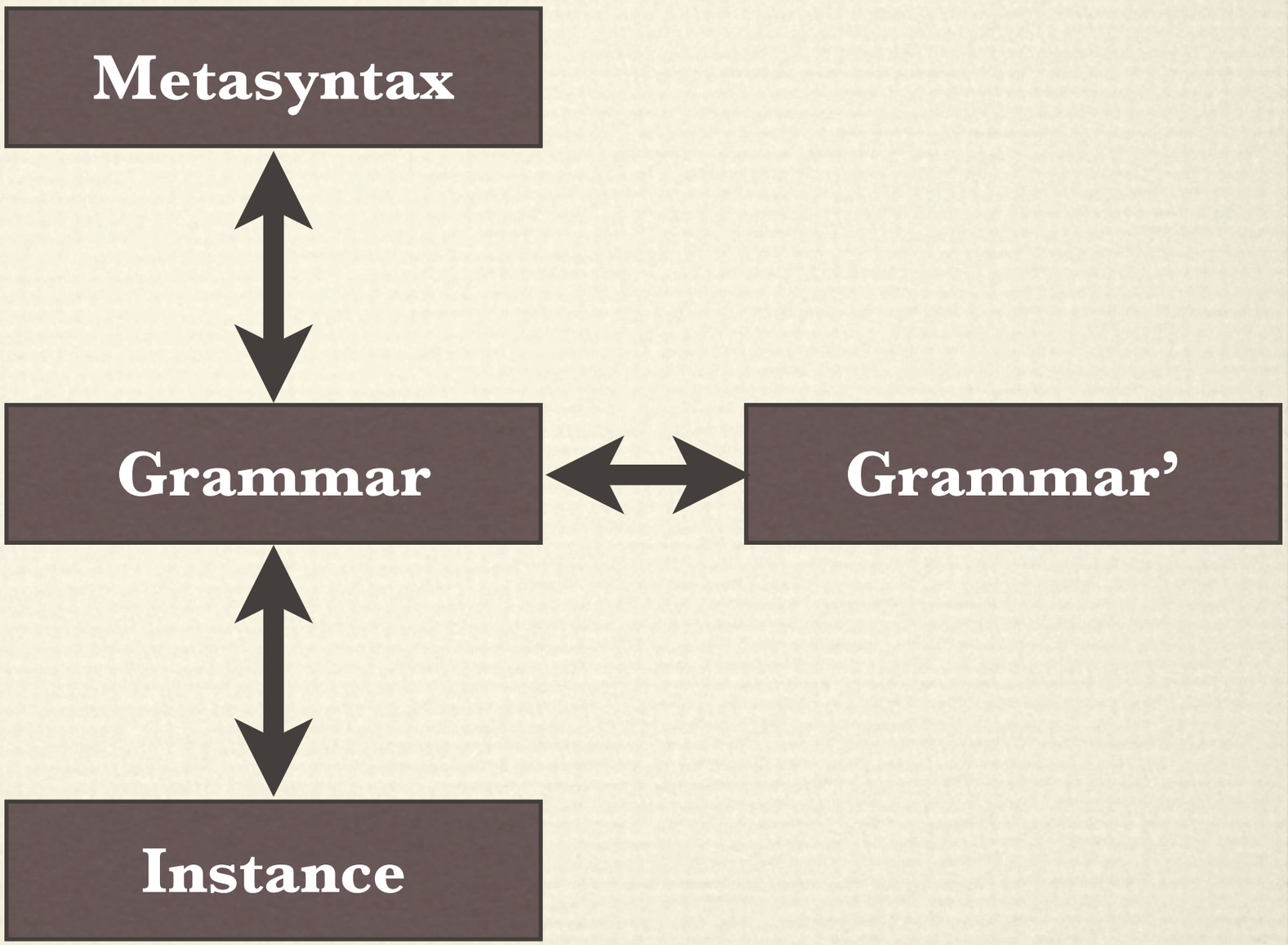
Grammar world



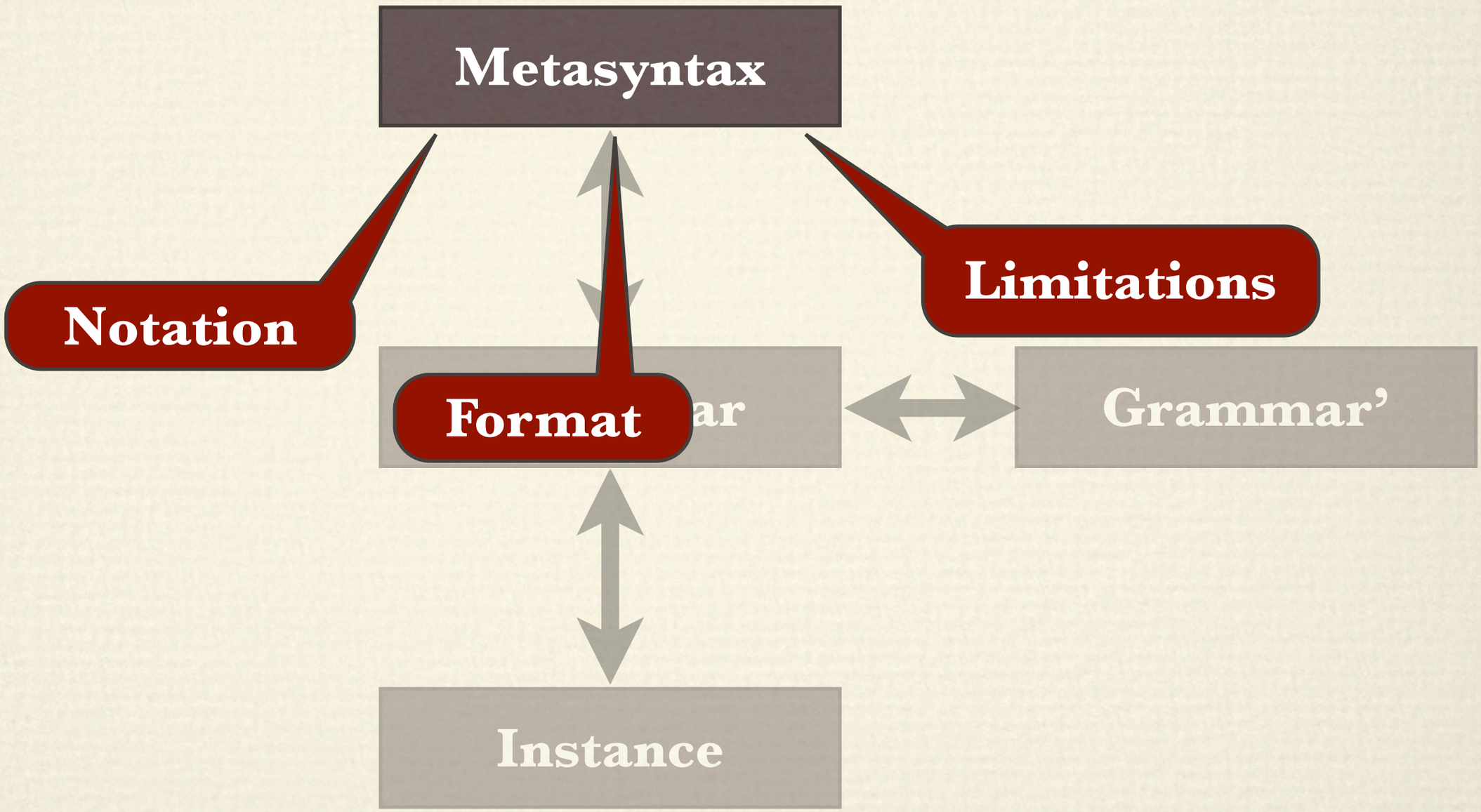
Grammar world: instances



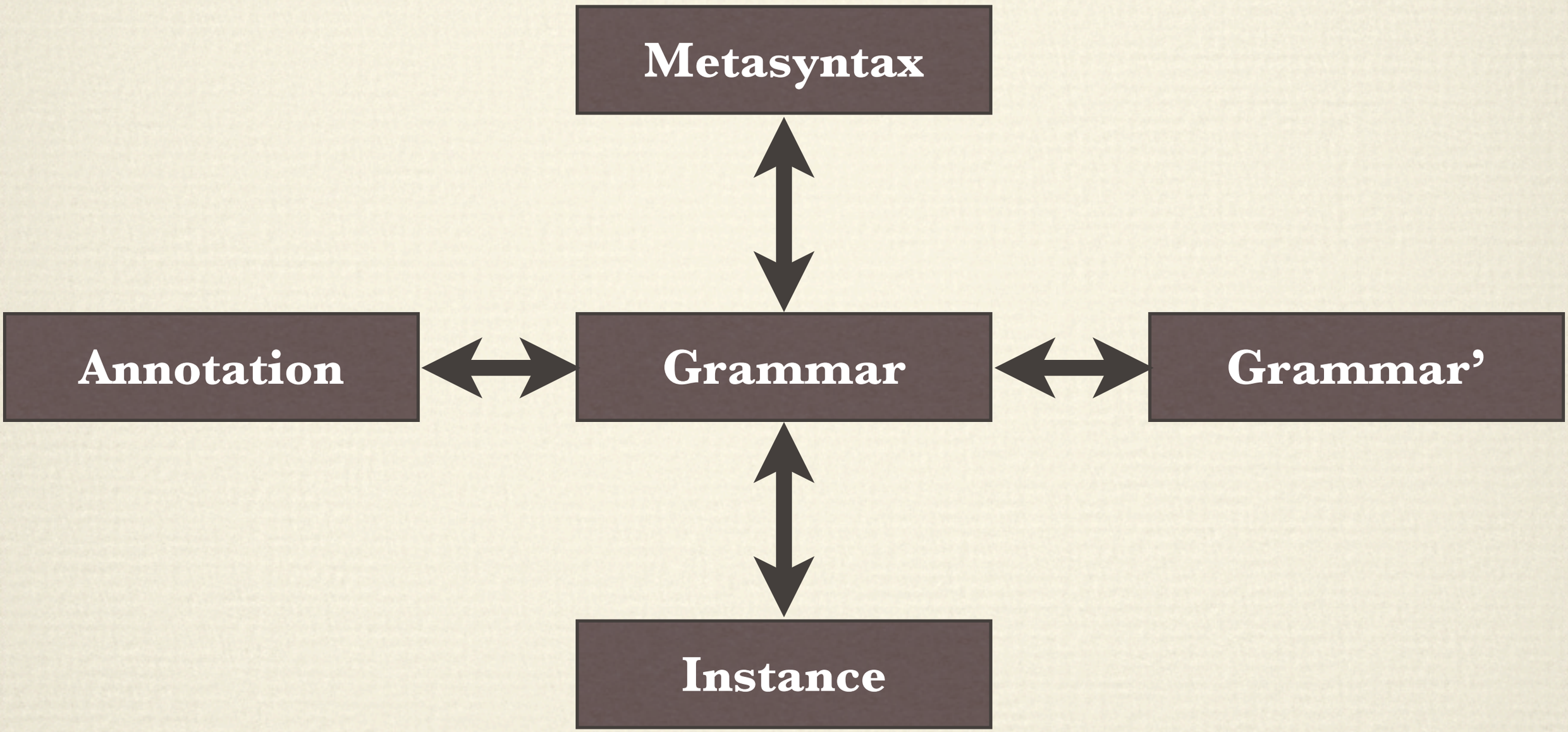
Grammar world



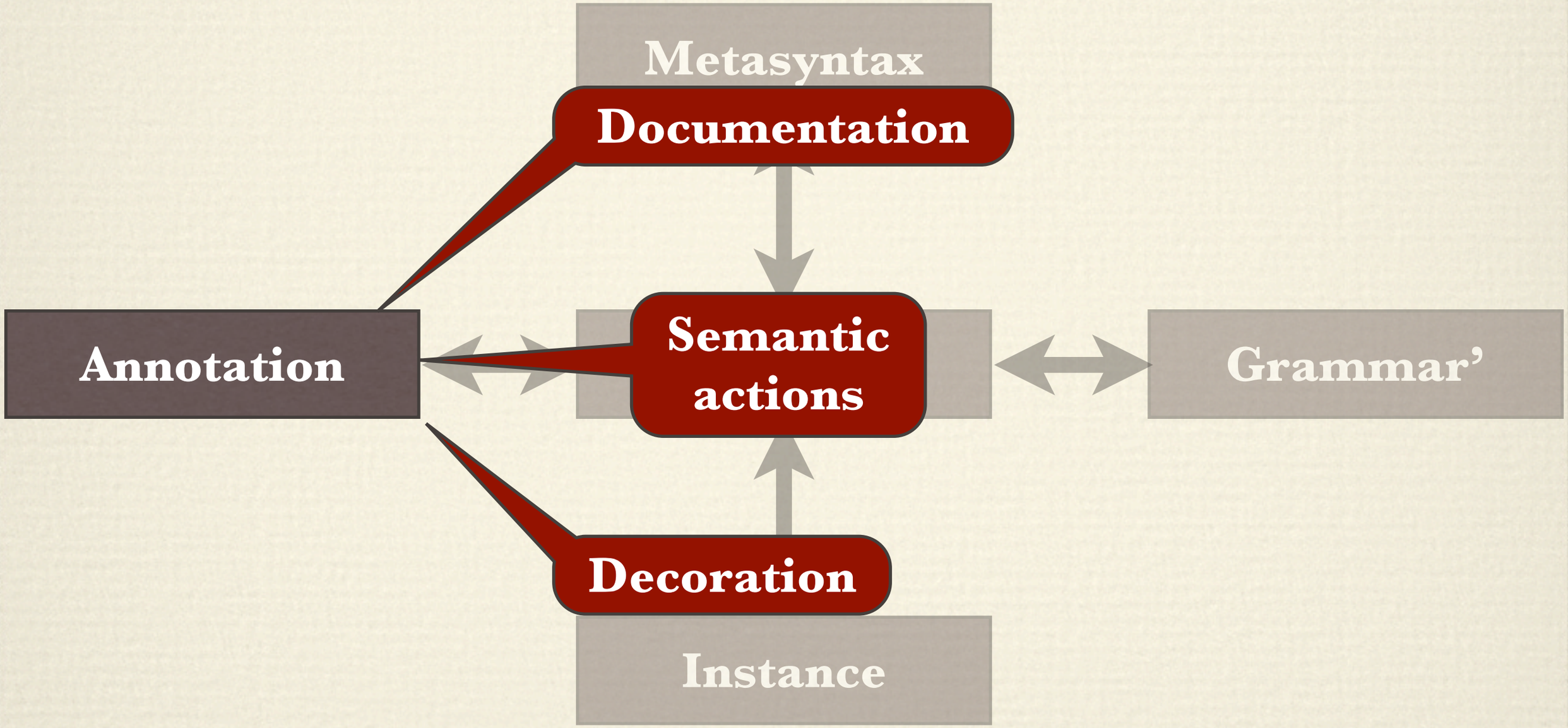
Grammar world: metamodels



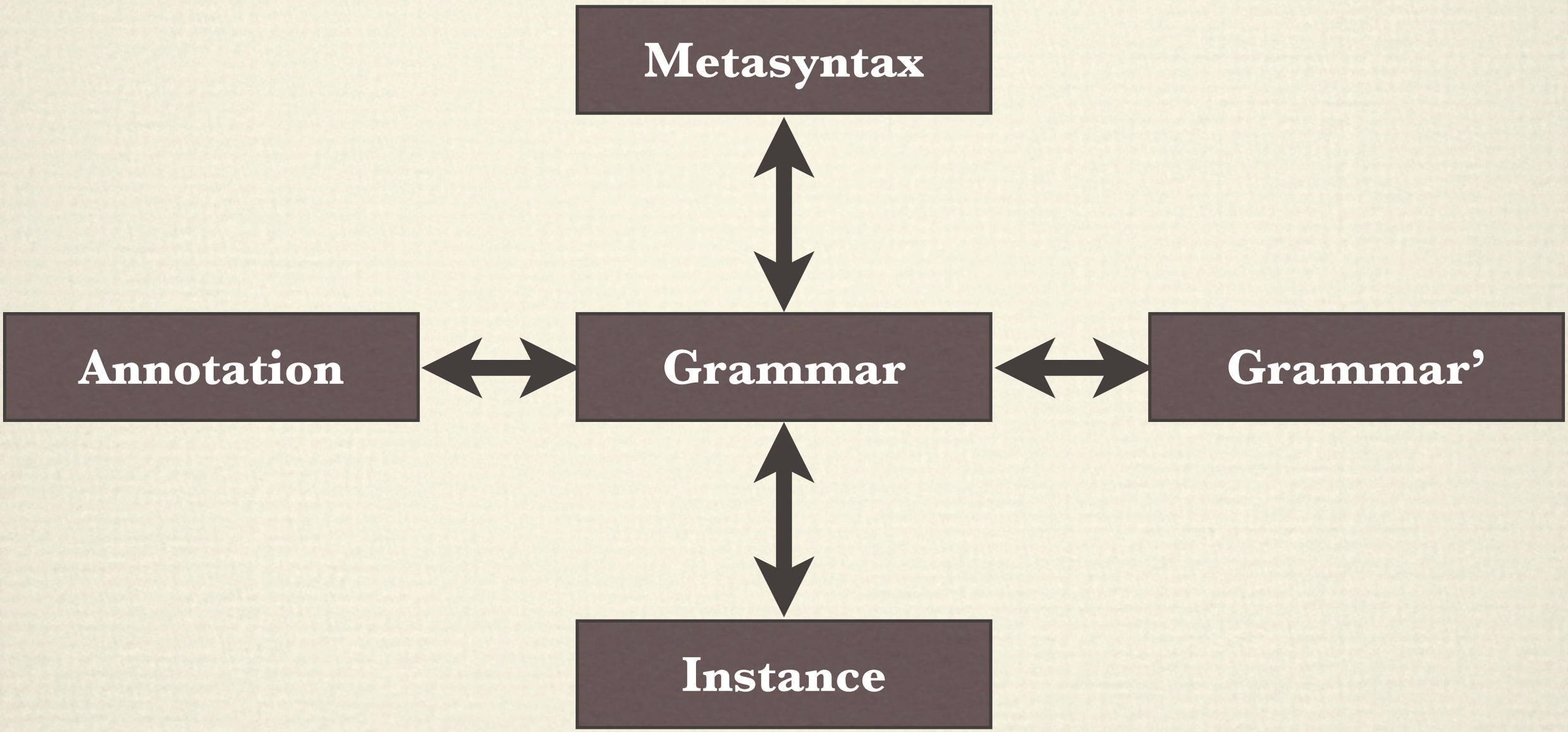
Grammar world



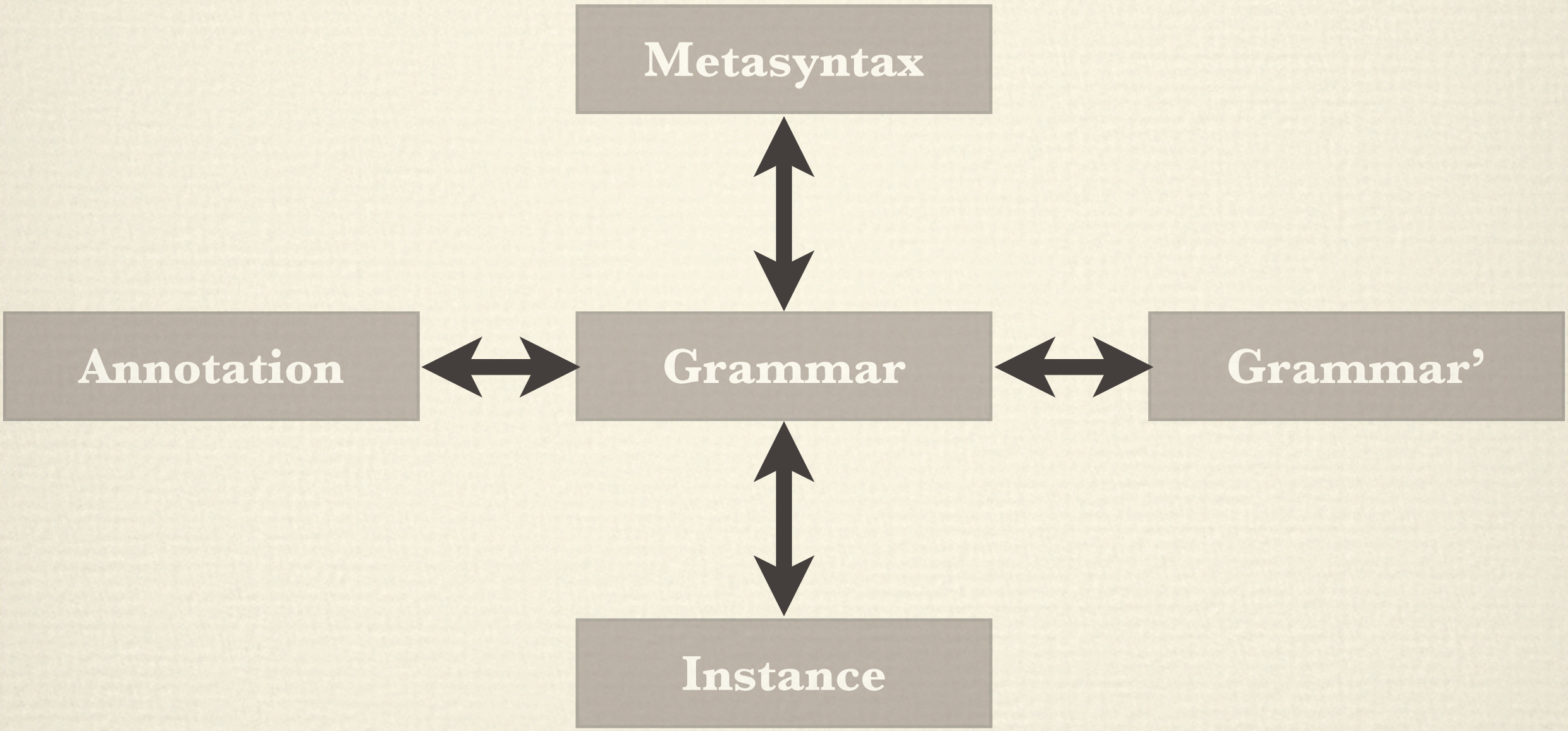
Grammar world: annotations



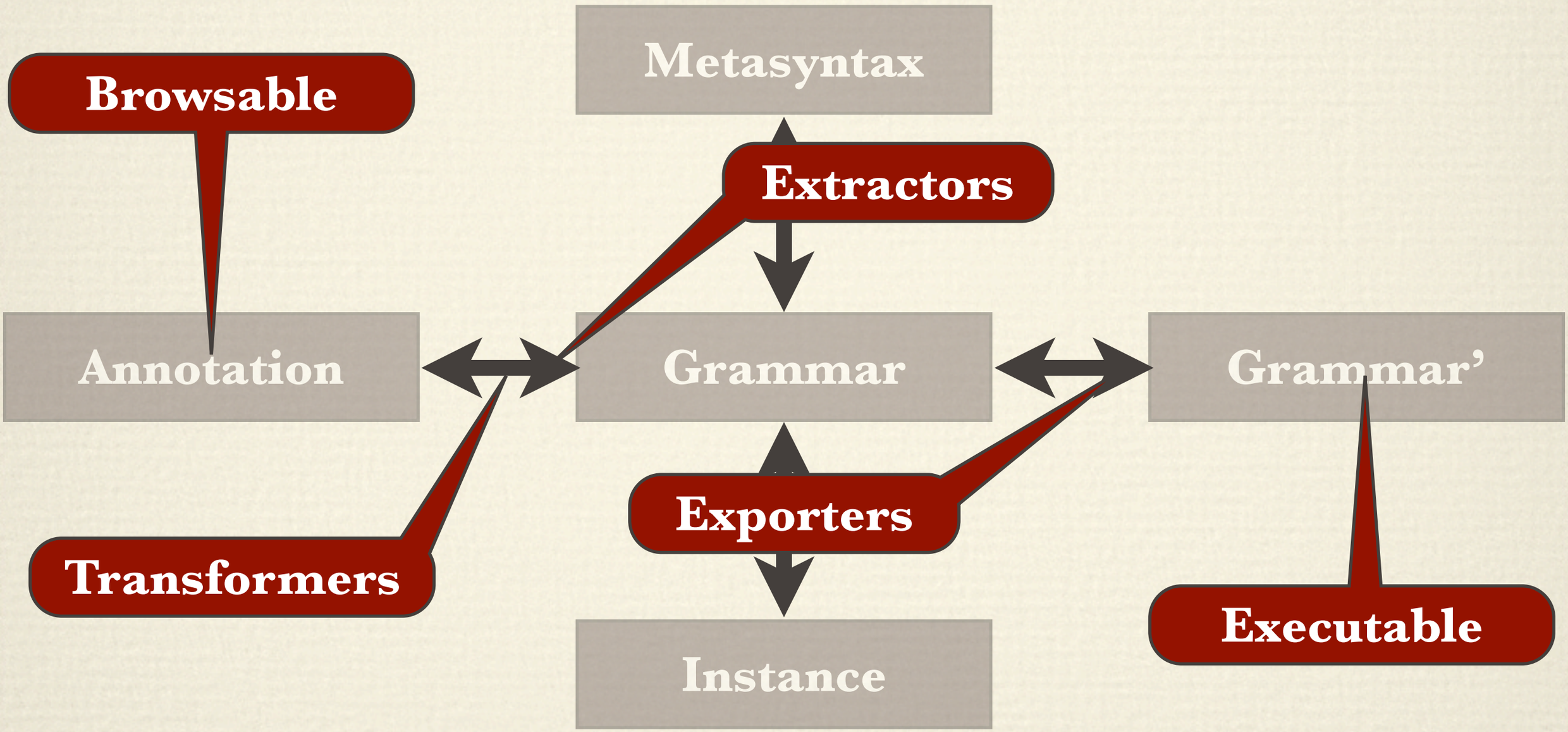
Grammar world



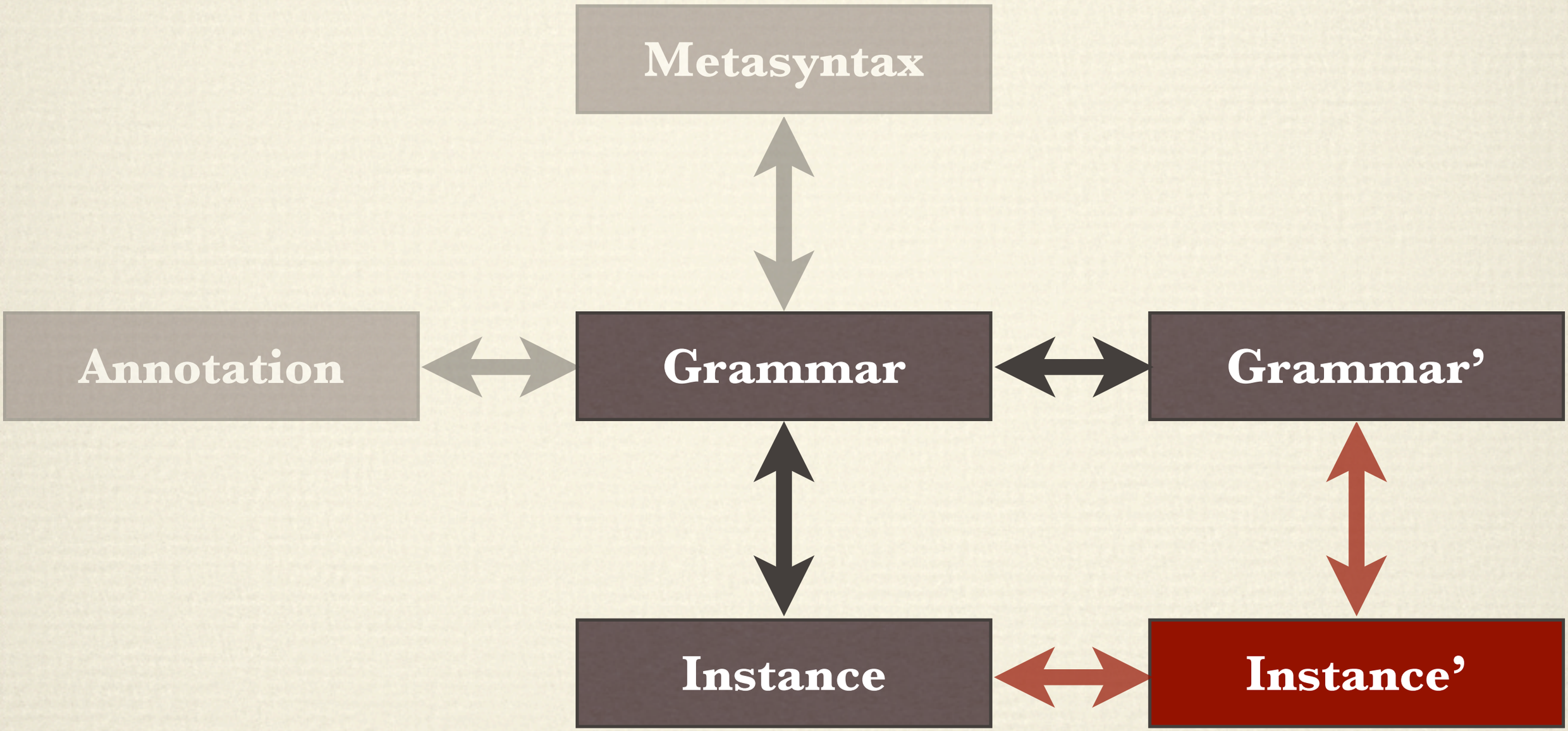
Grammar world



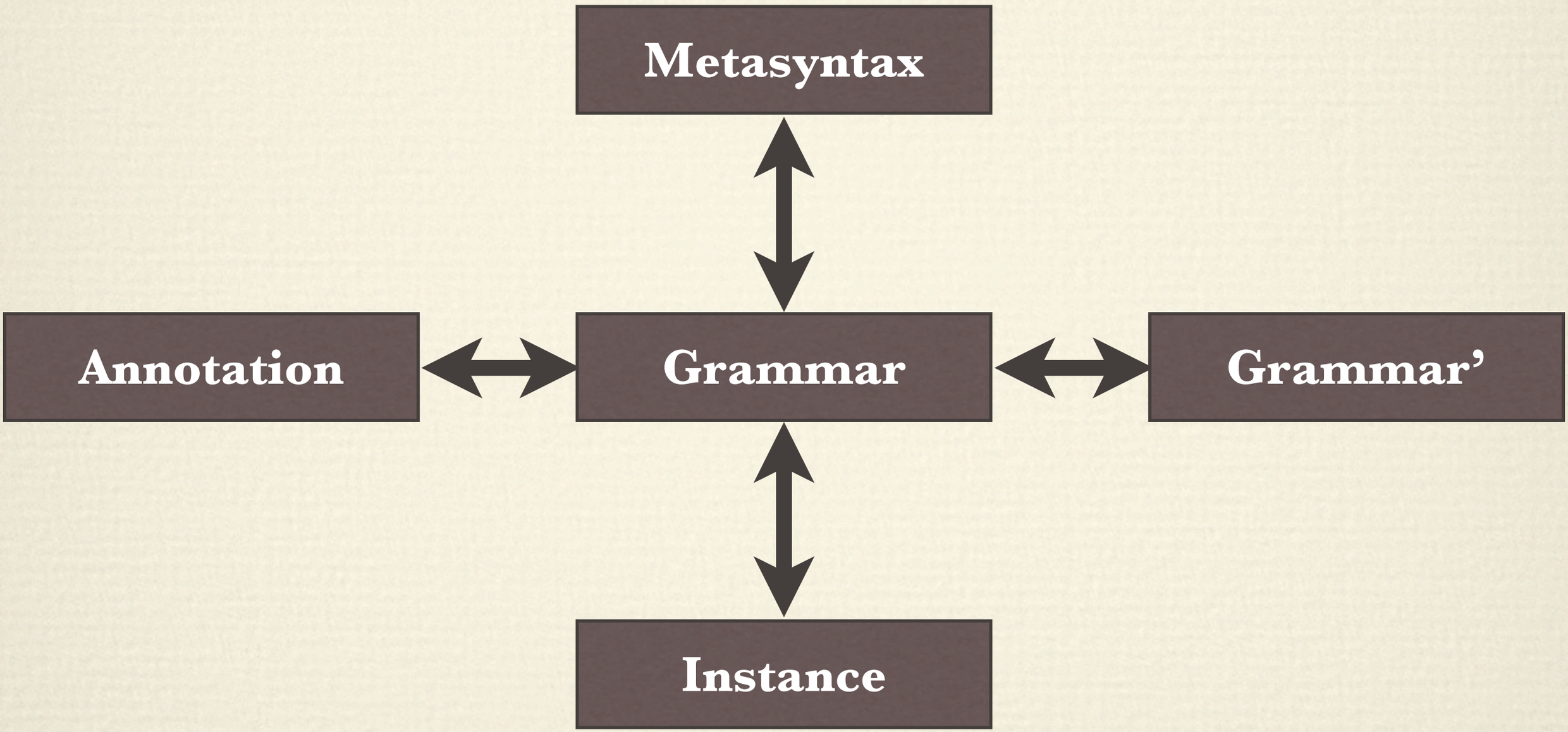
Grammar world: transformations



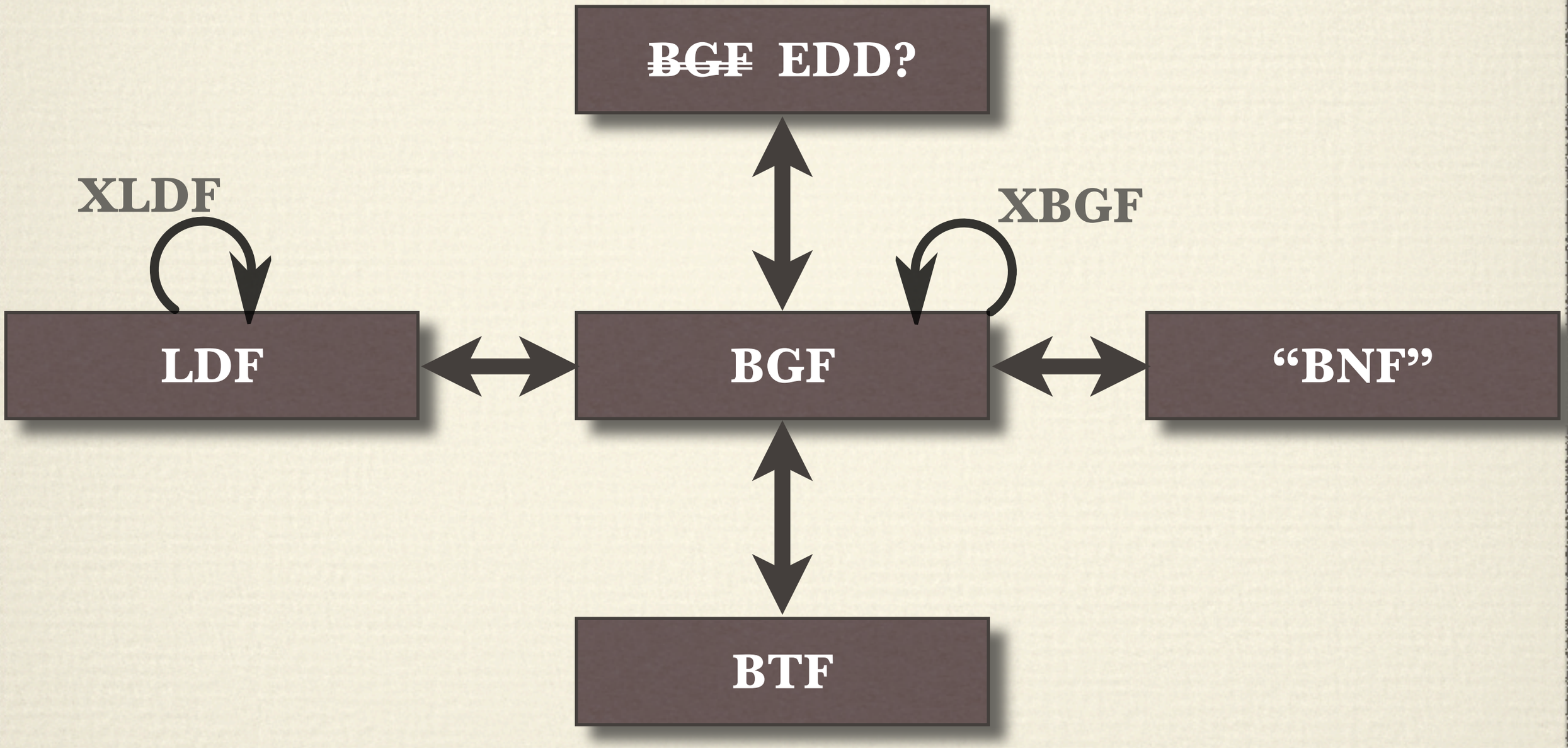
Grammar world: synchronisations



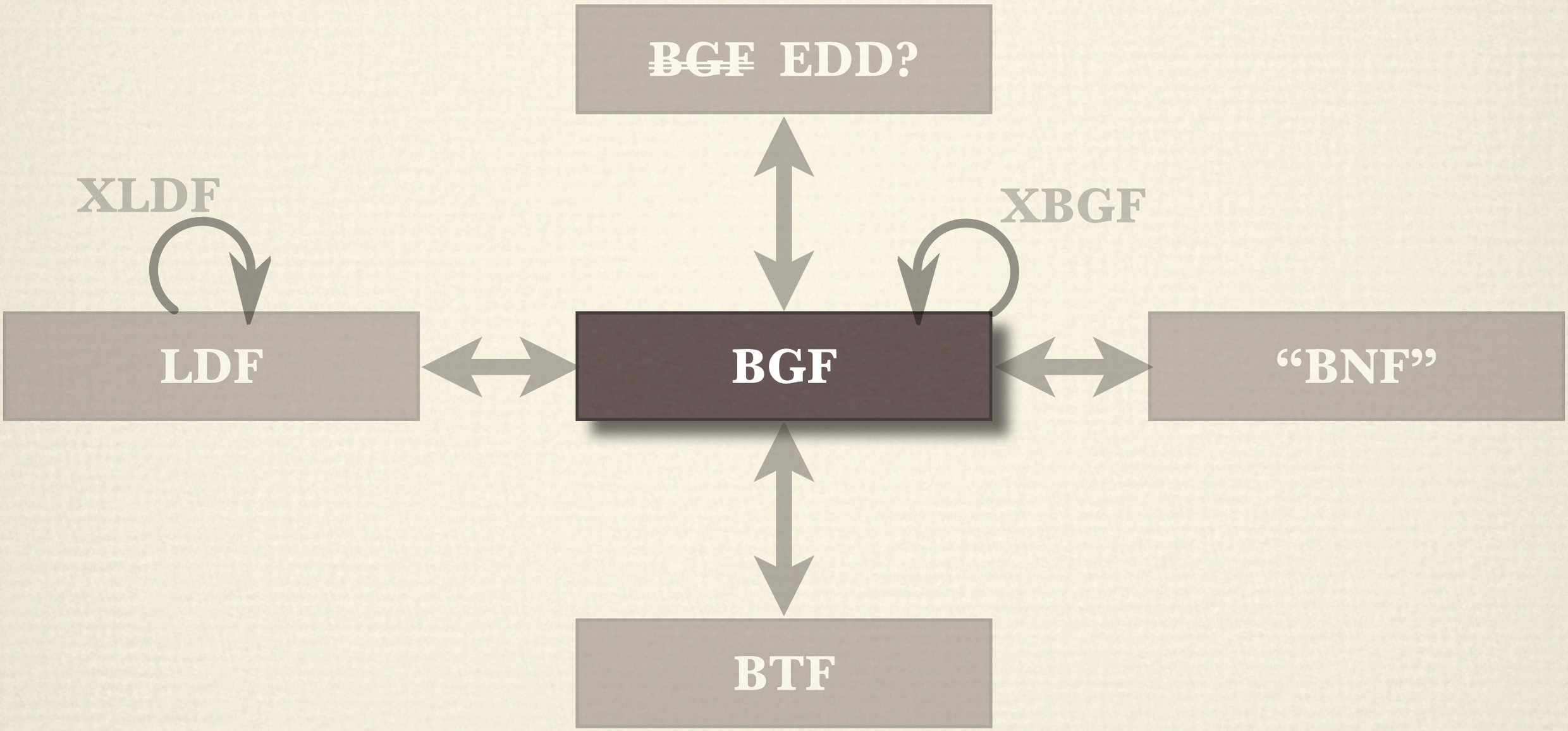
Domain



Domain ⇒ DSLs



What is BGF and what can it do?



What is BGF and what can it do?

- ★ BNF: symbols, composition
- ★ EBNF: *, +, ?
- ★ Production labels
- ★ Expression selectors
- ★ Universal type
- ★ Namespaces

"BNF"

BNF-like Grammar Format

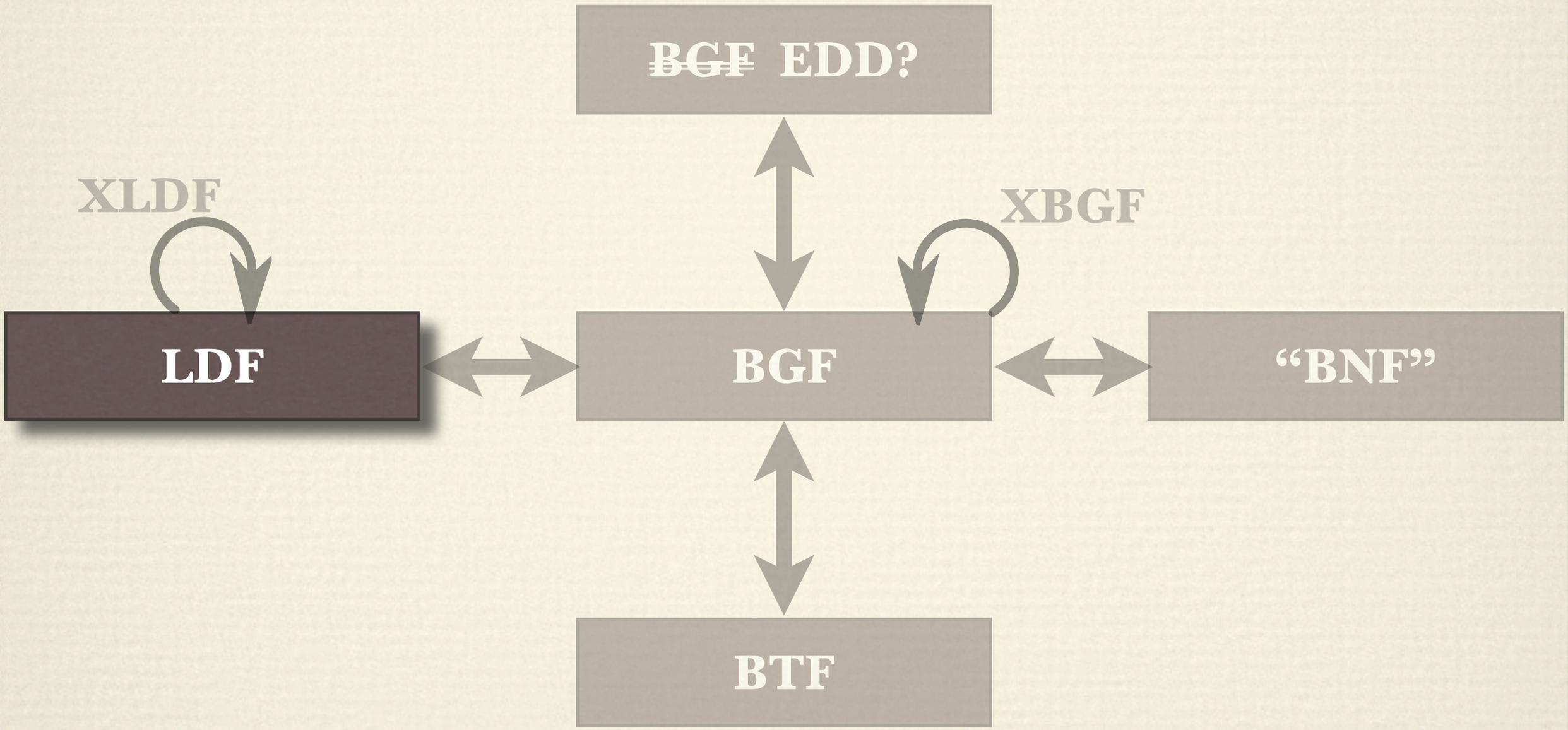
```
<?xml version="1.0" encoding="UTF-8"?>
<bgf:grammar xmlns:bgf="http://planet-sl.org/bgf">
        <root>grammar</root>
        <bgf:production>
                <nonterminal>grammar</nonterminal>
                <bgf:expression>
                        <sequence>
                                <bgf:expression>
                                        <star>
                                                <bgf:expression>
                                                        <selectable>
                                                                <selector>root</selector>
                                                                <bgf:expression>
                                                                        <nonterminal>nonterminal</nonterminal>
                                                                </bgf:expression>
                                                        </selectable>
                                                </bgf:expression>
                                        </star>
                                </bgf:expression>
                                <bgf:expression>
                                        <star>
                                                <bgf:expression>
                                                        <nonterminal>production</nonterminal>
                                                </bgf:expression>
                                        </star>
                                </bgf:expression>
                        </sequence>
                </bgf:expression>
        </bgf:production>
        <bgf:production>
                <nonterminal>production</nonterminal>
                <bgf:expression>
                        <sequence>
                                <bgf:expression>
                                        <optional>
                                                <bgf:expression>
                                                        <selectable>
                                                                <selector>label</selector>
                                                                <bgf:expression>
                                                                        <nonterminal>label</nonterminal>
                                                                </bgf:expression>
                                                        </selectable>
                                                </bgf:expression>
                                        </optional>
                                </bgf:expression>
                                <bgf:expression>
                                        <selectable>
                                                <selector>nonterminal</selector>
                                                <bgf:expression>
                                                        <nonterminal>nonterminal</nonterminal>
                                                </bgf:expression>
                                        </selectable>
                                </bgf:expression>
                                <bgf:expression>
                                        <nonterminal>expression</nonterminal>
                                </bgf:expression>
                        </sequence>
                </bgf:expression>
```

BNF-like Grammar Format

```
grammar:
    root::nonterminal* production*
production:
    label::label? nonterminal::nonterminal expression
expression:
    epsilon::EPSILON
    empty::EPSILON
    value::value
    any::EPSILON
    terminal::terminal
    nonterminal::nonterminal
    selectable::(selector::selector expression)
    sequence::(expression+)
    marked::expression
    choice::(expression+)
    optional::expression
    plus::expression
    star::expression
value:
    int::EPSILON
    string::EPSILON
```

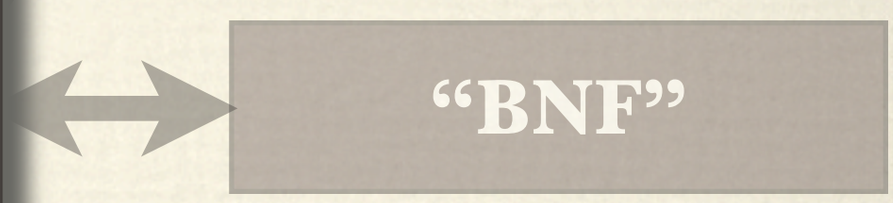

<?xml version="1.0" encoding="UTF-8"?>
<bgf:grammar xmlns:bgf="http://planet-sl.org/bgf">
<root>grammar</root>
<bgf:production>
<nonterminal>gram
<bgf:expression>
<sequenc
</sequen
</bgf:expression>
</bgf:production>
<bgf:production>
<nonterminal>prod
<bgf:expression>
<sequenc
<bgf:expression>
<nonterminal>expression</nonterminal>
</bgf:expression>
</sequence>
</bgf:expression>

What is LDF and what can it do?

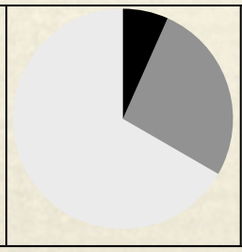
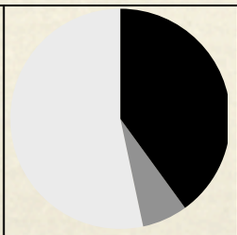
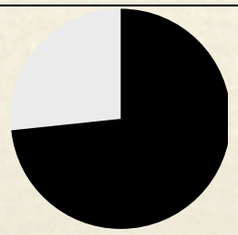
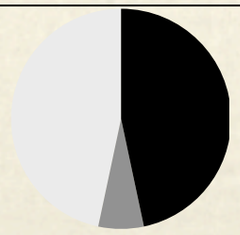
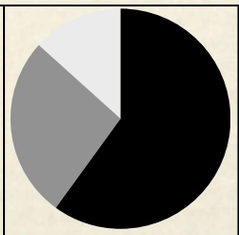
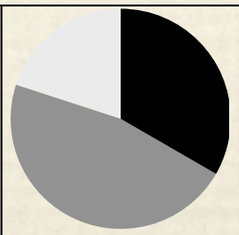
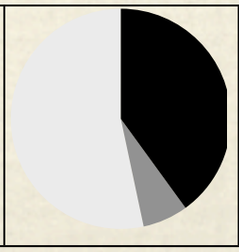
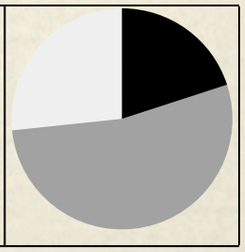


What is LDF and what can it do?

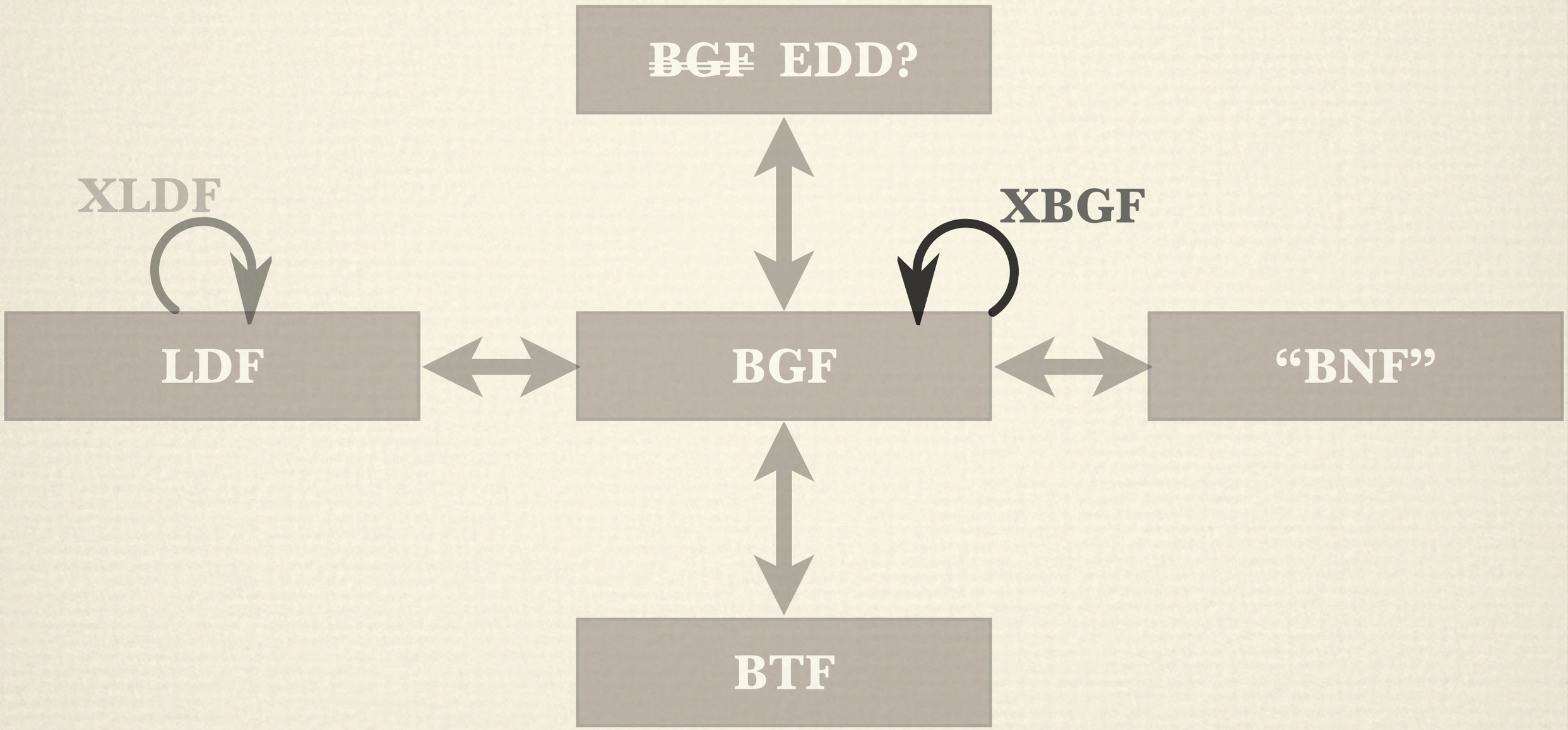
- ★ A unified format for language document engineering
- ★ Derived from real language documentation analysis
- ★ Integrated with current research & infrastructure
- ★ Published, reported
- ★ Awaiting a case study



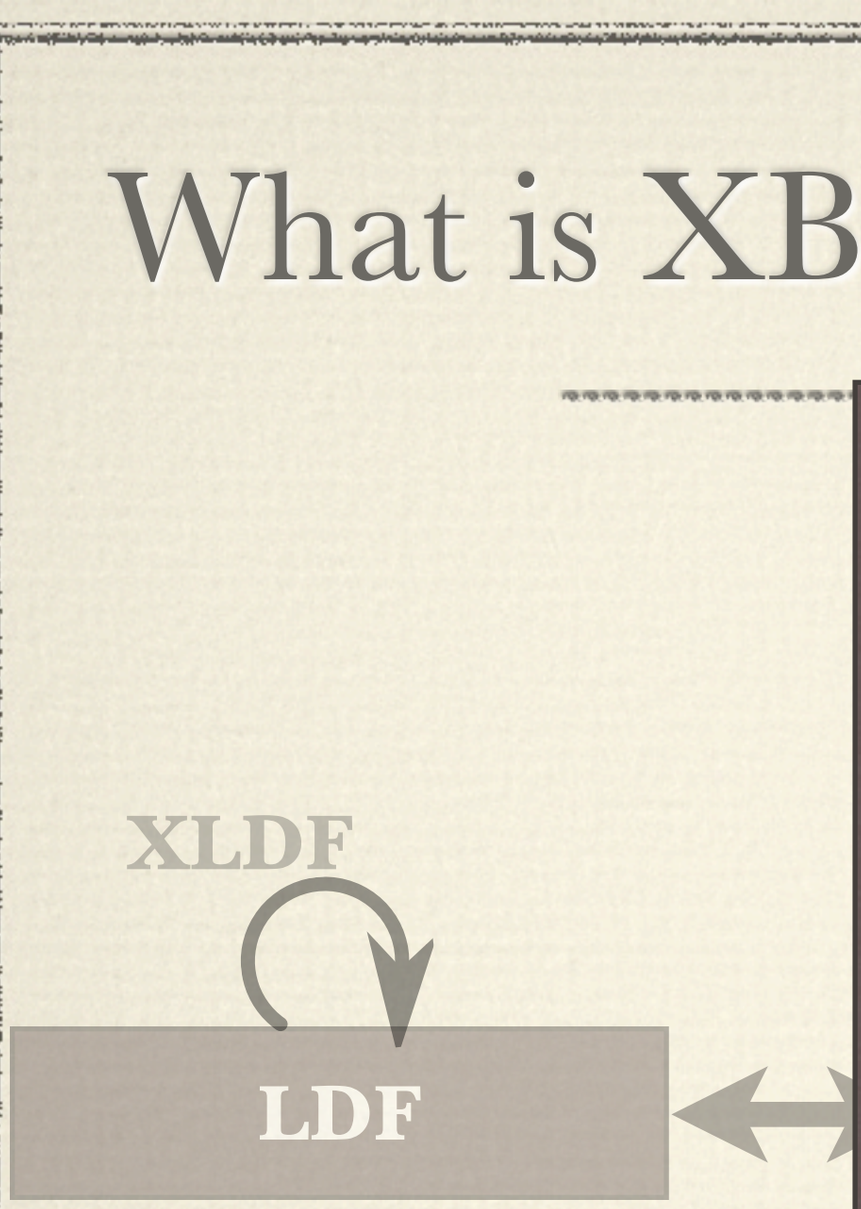
Mapping to LDF

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								

What is XBGF and what can it do?



What is XBGF and what can it do?



★ Advanced operator suite for grammar transformation

★ Based on earlier work

★ Groups of operators with respect to semantics preservation

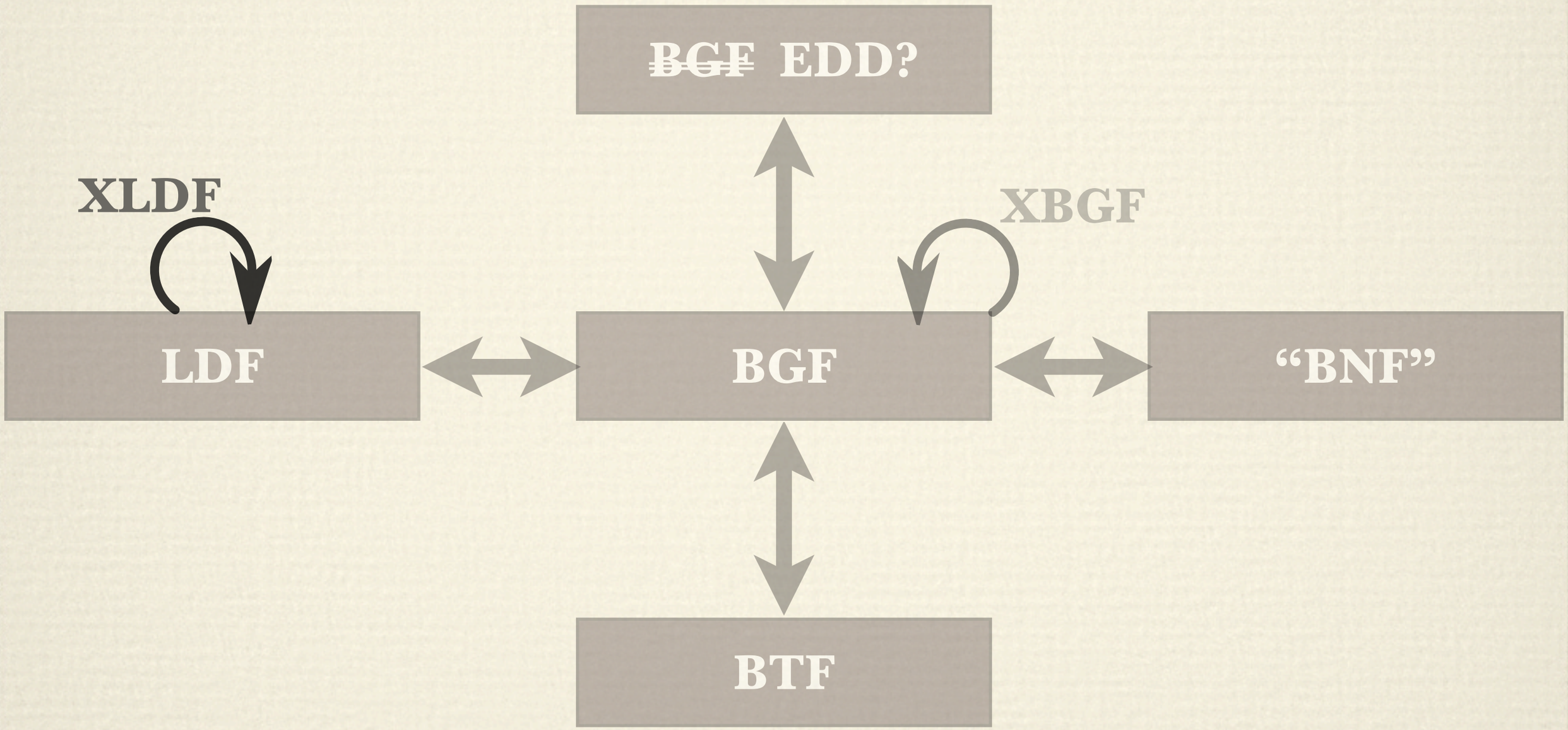
★ …to be continued…

	VSC2	FST	GDK	GRK	XBGF
add a definition for a bottom nonterminal	resolve	resolve	resolve		define
add a new definition	introduce	introduce	introduce		introduce
add a new definition & fold it	extract			extract	extract
add a production to any nonterminal definition	include	include	include		addV
add a production to the grammar	add	add		add	
add alternatives to a choice					addH
change the order in a sequence	permute			permute	permute
do nothing	id	id			
give a production a label					designate
give a subexpression a selectable name					deanonymize
inject a nillable symbol					appear
inject a terminal symbol					concretize
inject symbols to a sequence					inject
inline a chain production					unchain
introduce a chain production					chain
introduce a reflexive chain production					detour
introduce several possibly interconnected definitions					import
merge two nonterminals					unite
merge two nonterminals if their definitions are equal		equate			equate
merge two nonterminals, one of which is bottom	unify	unify	unify		unite*
move a production across modules/sections		move			
perform factoring transformation			preserve*		factor
perform folding transformation	fold	fold	fold	fold	fold
perform massaging transformation	preserve	simplify	preserve		massage
perform narrowing transformation (as in x? to x)	restrict*	restrict*	restrict*		narrow
perform specialized automated factoring transformation					distribute

merge two nonterminals if their definitions are equal		equate			equate
merge two nonterminals, one of which is bottom	unify	unify	unify		unite*
move a production across modules/sections		move			
perform factoring transformation			preserve*		factor
perform folding transformation	fold	fold	fold	fold	fold
perform massaging transformation	preserve	simplify	preserve		massage
perform narrowing transformation (as in x? to x)	restrict*	restrict*	restrict*		narrow
perform specialized automated factoring transformation					distribute
perform unfolding transformation	unfold	unfold	unfold	unfold	unfold
perform widening transformation (as in x+ to x*)	generalise	generalise	generalize		widen
project a nillable symbol	restrict*	restrict*	restrict*		disappear
project a terminal symbol					abstractize
project symbols from a sequence					project
remove a definition of a possibly used nonterminal	reject	reject	reject		undefine
remove a label from a production					unlabel
remove a production from the grammar	sub	sub			
remove a reflexive chain production					abridge
remove a selector in a subexpression					anonymize
remove alternatives from a choice					removeH
remove any part of a grammar	reset	reset			
remove unused definition	eliminate	eliminate	eliminate	reject	eliminate
removes one production of a nonterminal (not the last one)	exclude	exclude	exclude		removeV
rename a label					renameL
rename a nonterminal	rename	rename	rename		renameN
rename a nonterminal in a limited scope	substitute	substitute			replace*
rename a selector					renameS
replace a nonterminal with one of its definitions					downgrade

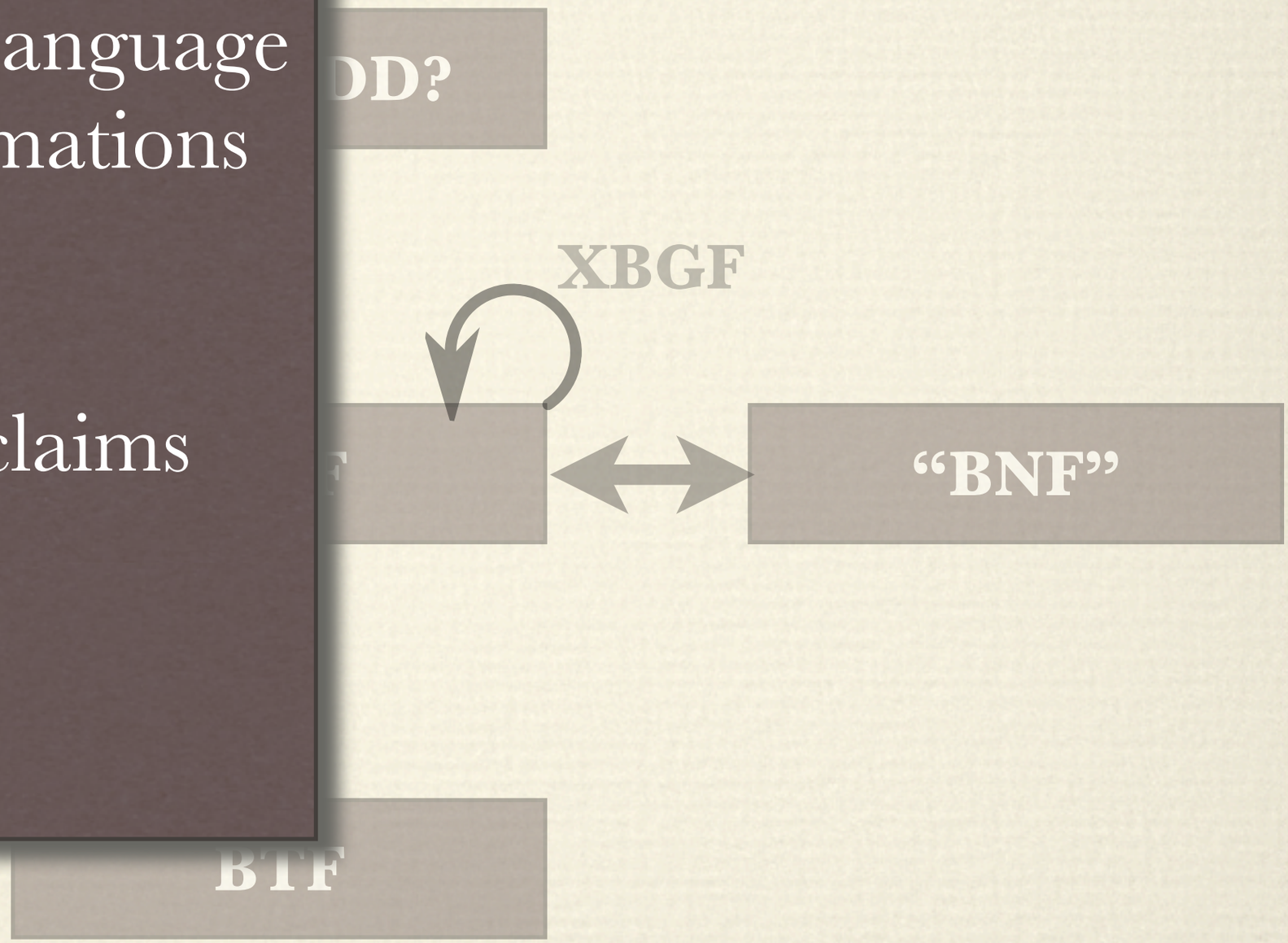
remove a production from the grammar	sub	sub			
remove a reflexive chain production					abridge
remove a selector in a subexpression					anonymize
remove alternatives from a choice					removeH
remove any part of a grammar	reset	reset			
remove unused definition	eliminate	eliminate	eliminate	reject	eliminate
removes one production of a nonterminal (not the last one)	exclude	exclude	exclude		removeV
rename a label					renameL
rename a nonterminal	rename	rename	rename		renameN
rename a nonterminal in a limited scope	substitute	substitute			replace*
rename a selector					renameS
replace a nonterminal with one of its definitions					downgrade
replace a nonterminal with ε	delete		delete		replace*
replace a terminal with another terminal					renameT
replace an expression by a nonterminal that can be evaluated to it					upgrade
replace any expression with another expression	replace	replace		replace	replace
replace iteration with left-associative equivalent					lassoc
replace iteration with recursion			preserve*		yaccify
replace iteration with right-associative equivalent					rassoc
replace recursion with iteration			preserve*		deyaccify
replace the current definition by a new one			redefine		redefine
separate one nonterminal into several (reverse of merge)	separate	seperate	separate		
terminate transformation sequence	fail	fail	write		dump
transpose a multi-production definition to the one with top-level choices					horizontal
transpose top-level choices to multiple productions					vertical
unfold & eliminate					inline

What is XLDF and what can it do?



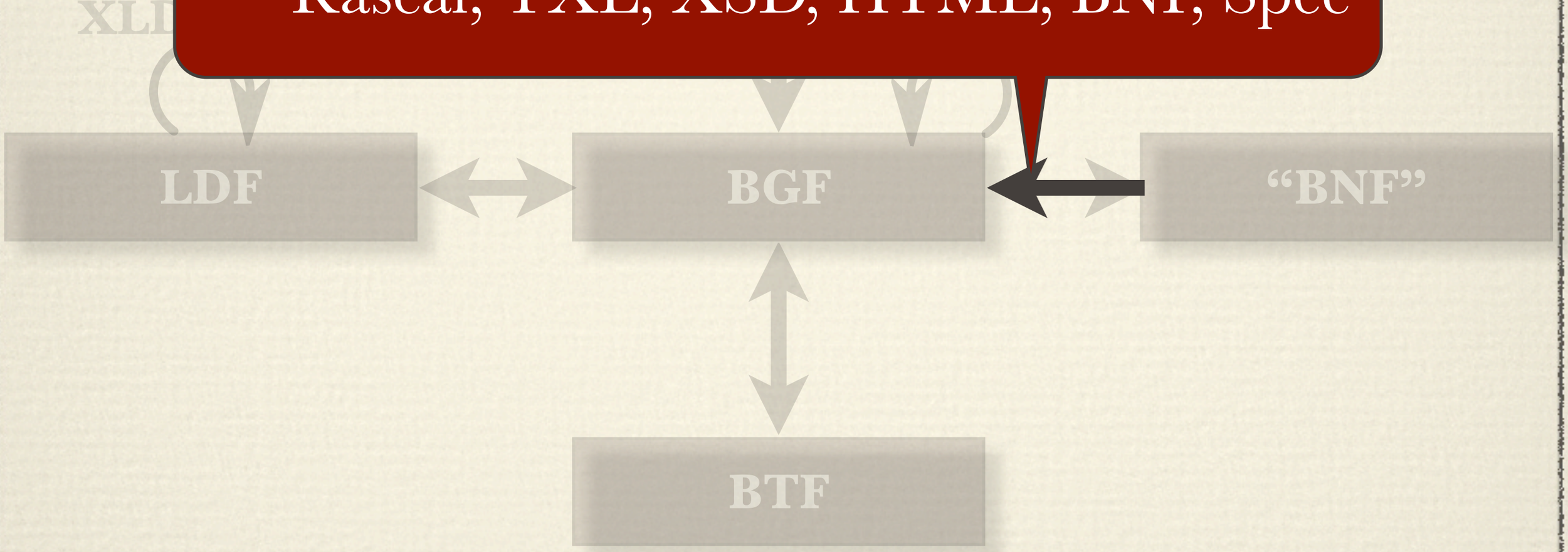
What is XLDF and what can it do?

- ★ Operator suite for language document transformations
- ★ Working prototype
- ★ No expressiveness claims
- ★ No previous work
- ★ Future work

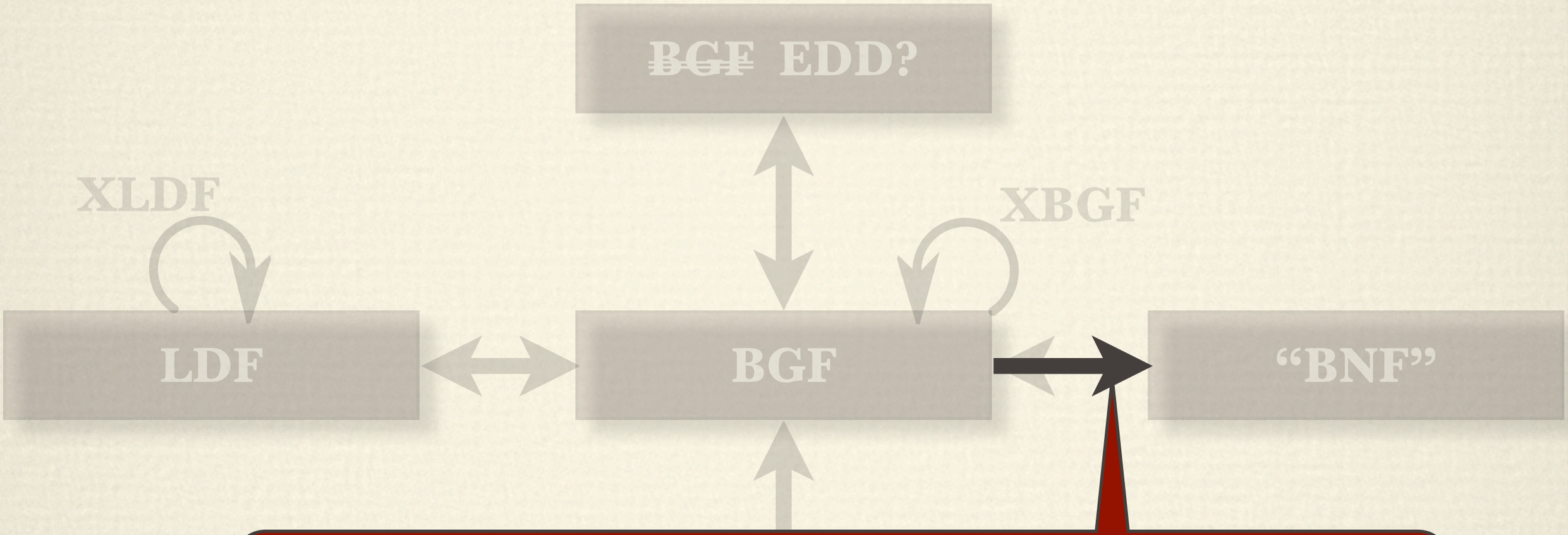


What else can SLPS do?

★ Numerous extractors:
ANTLR, DCG, Ecore, Java, SDF, PDF,
Rascal, TXL, XSD, HTML, BNF, Spec



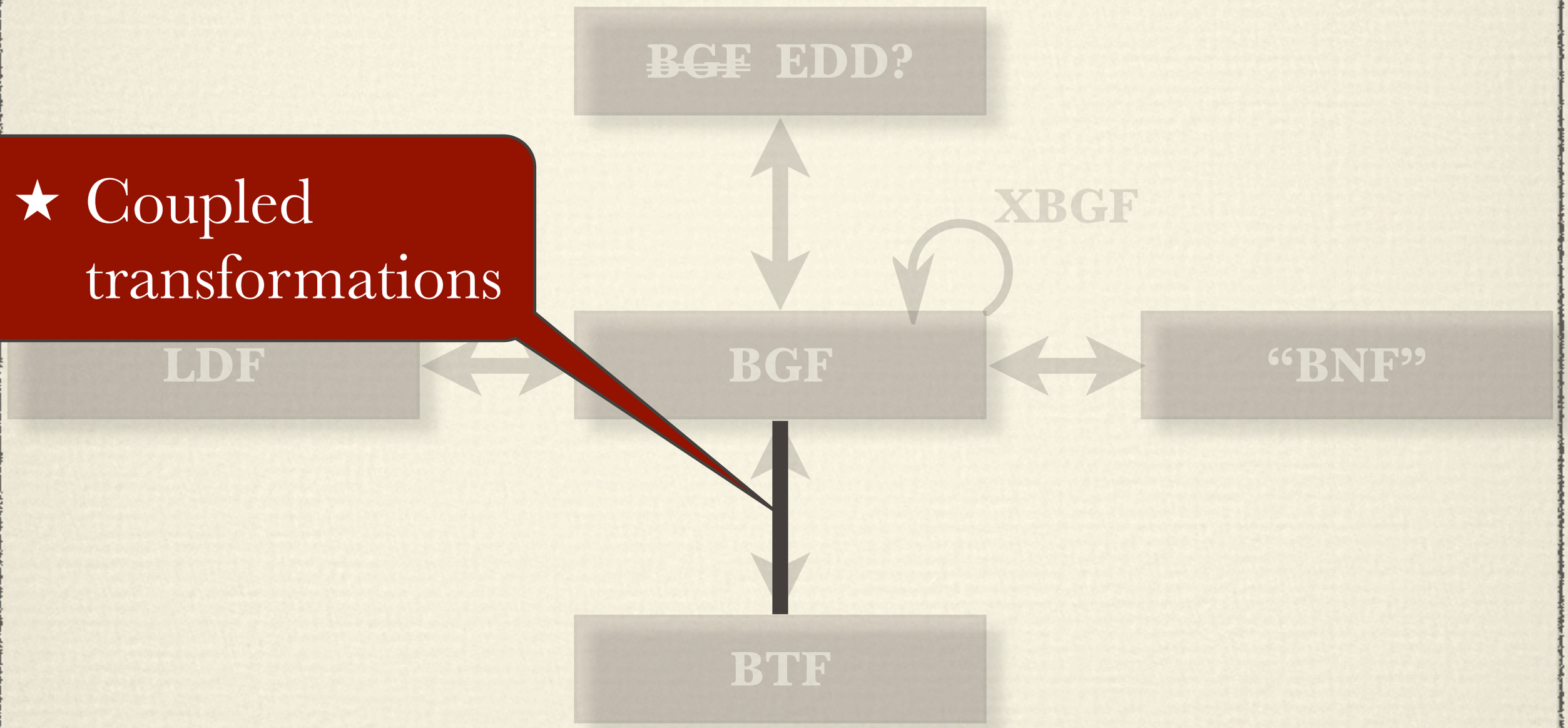
What else can SLPS do?



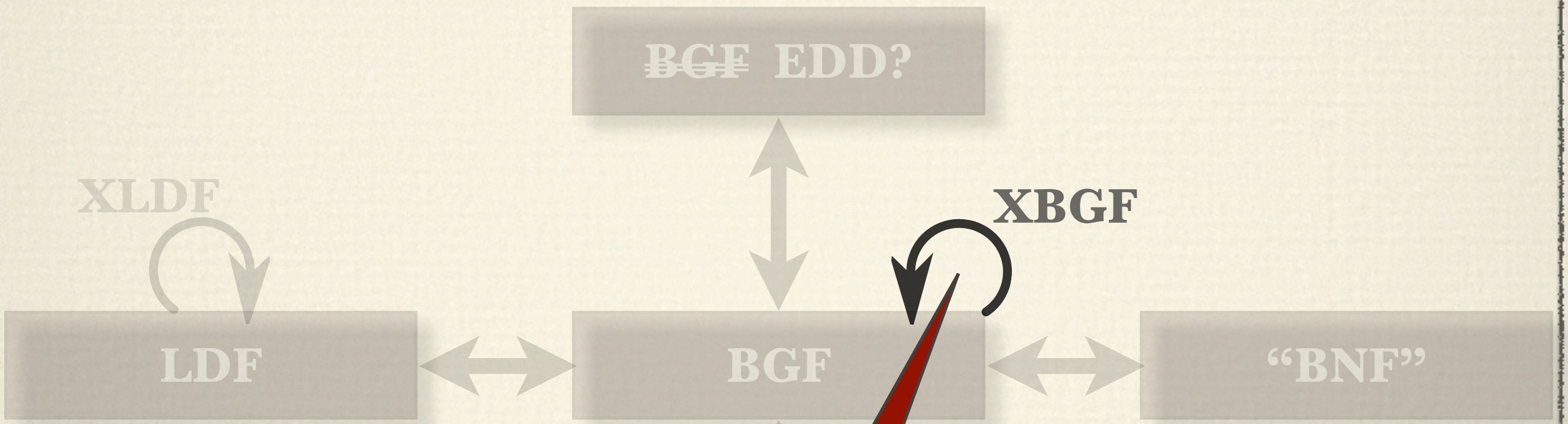
★ Numerous exporters & pretty-printers:
EBNF, SDF, TXL, TEX, …

What else can SLPS do?

★ Coupled transformations



What else can SLPS do?



- ★ Generators
- ★ Convergence scenarios

Grammar transformation

★ Grammar in, grammar out

★ A set of parametrised operators

★ Programmable transformation steps

★ Best way to approach grammarware engineering

★ Generators: grammar in, transformation out

A fragment of concrete syntax.
What if we want to derive the abstract syntax?

```
expr : ...;
atom : ID | INT  | '(' expr ')';
```

Need to project away “(” and “)”

Need to merge “expr” and “atom”

Alternative needs to go entirely

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;

XBGF Operator Suite

$$L(G_1) = L(G_2)$$

★ Semantics–preserving (refactoring)

- ✦ rename, import, introduce, eliminate
- ✦ fold, unfold, extract, inline, *lift*
- ✦ factor, distribute, horizontal, vertical
- ✦ yaccify, deyaccify, massage
- ✦ designate, unlabel
- ✦ ...

XBGF Operator Suite

★ Semantics–increasing/–decreasing

$$L(G_1) \subseteq L(G_2)$$
$$\vee$$
$$L(G_2) \subseteq L(G_1)$$

- ✦ appear, disappear
- ✦ narrow, widen
- ✦ add, remove
- ✦ upgrade, downgrade
- ✦ unite
- ✦ ...

XBGF Operator Suite

$$L(G_1) \not\subseteq L(G_2) \wedge L(G_2) \not\subseteq L(G_1)$$

★ Semantics–revising

- ✦ undefine, define
- ✦ inject, project
- ✦ permute
- ✦ abstractize, concretize
- ✦ replace, redefine

Grammar refactoring example

BGF (*read2*)

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

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

XBGF (*grammar refactoring*)

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

Grammar extension example

BGF (*read2*)

ClassModifier:
"public"
"protected"
"private"
"abstract"
"static"
"final"
"strictfp"

FieldModifier:
"public"
"protected"
"private"
"static"
"final"
"transient"
"volatile"

MethodModifier:
"public"
"protected"
"private"
"abstract"
"static"
"final"
"synchronized"
"native"
"strictfp"

XBGF (grammar optimisation)

```
unite(InterfaceModifier, Modifier);
unite(ConstructorModifier, Modifier);
unite(MethodModifier, Modifier);
unite(FieldModifier, Modifier);
... ... ...
```

Grammar revision example

BGF (*impl2*, *impl3*)

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

XBGF (*grammar correction*)

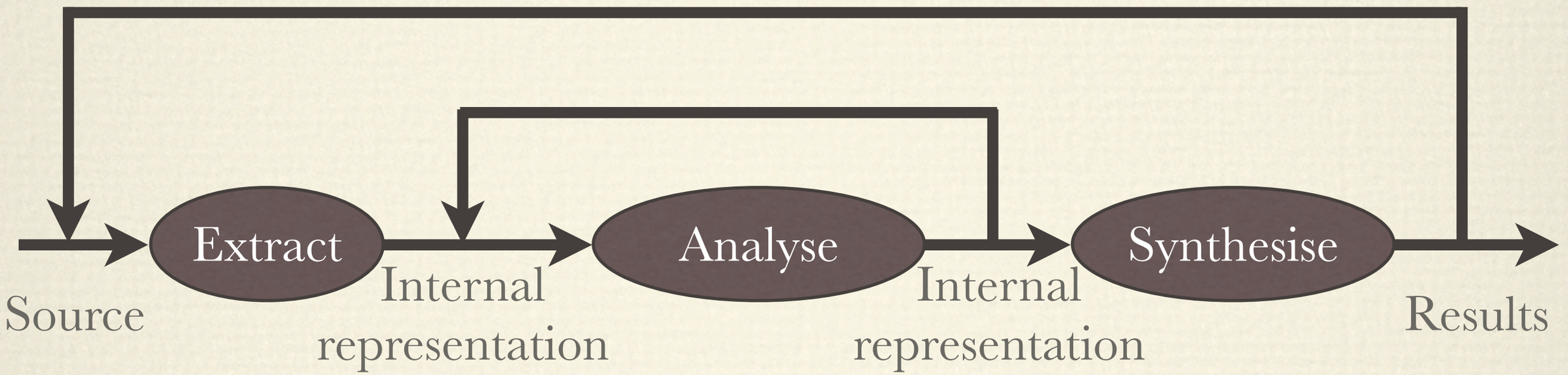
```
project(
 Expression2Rest:
    < Expression3 > "instanceof" Type
);
```

Methodology revealed

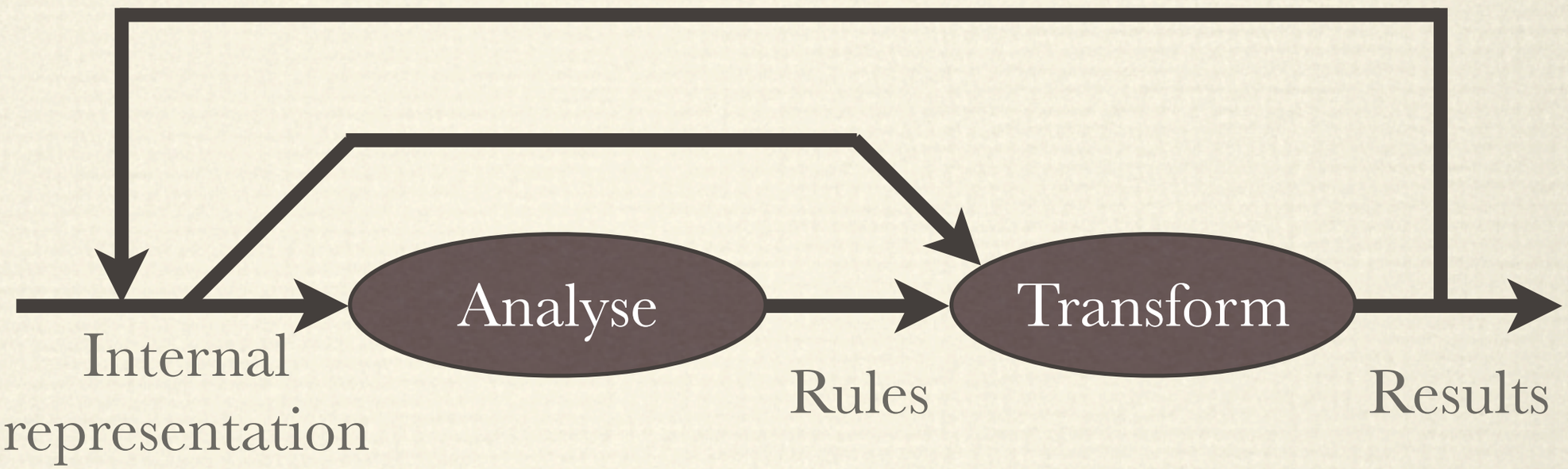
- ★ Recovery of languages
- ★ Convergence of languages
- ★ Documentation of languages
- ★ Also featured:
  - ★ Extraction
  - ★ Transformation
  - ★ Comparison
  - ★ Measurement

Grammarware metamodel
outdated

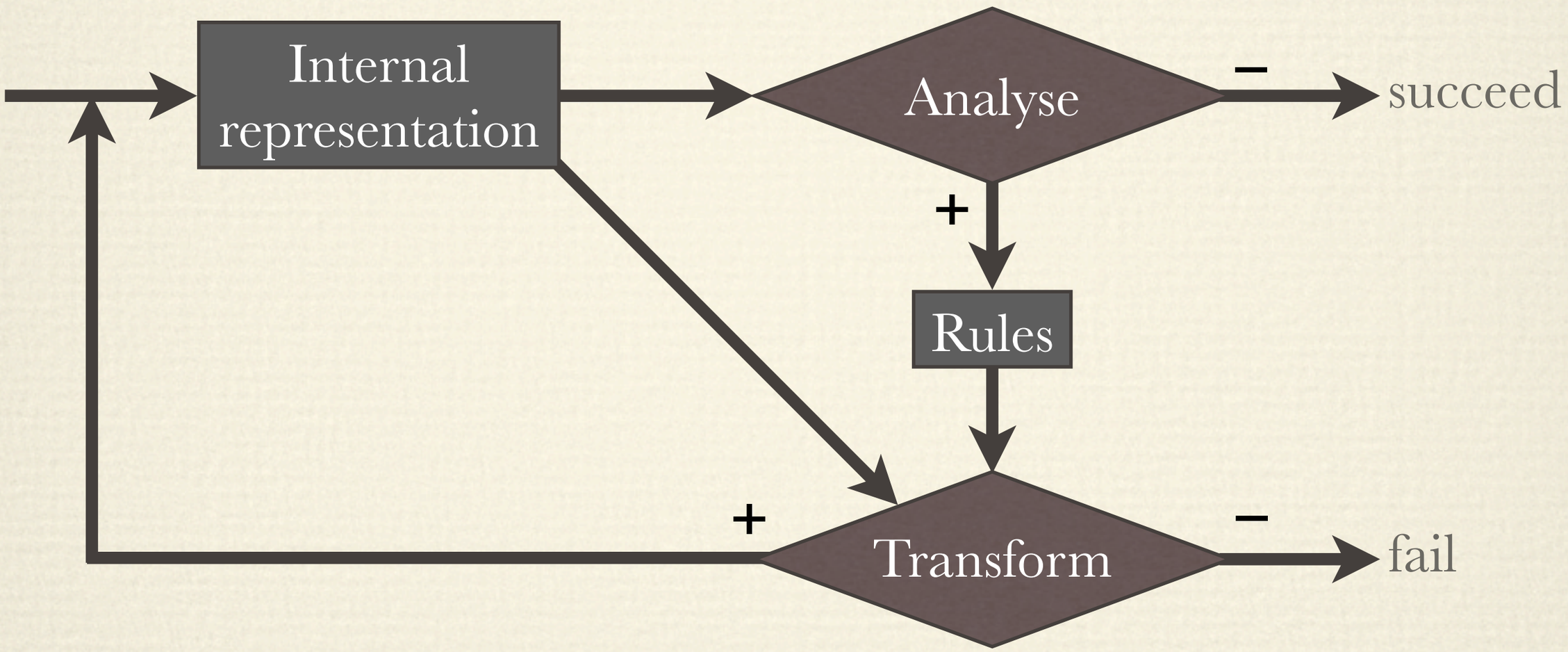
Grammarware metamodel
revisited



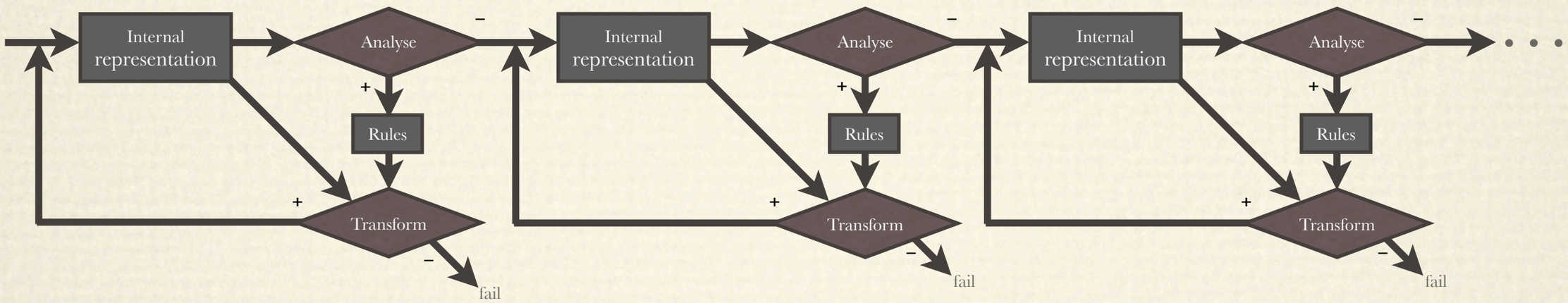
Grammarware metamodel
generalised



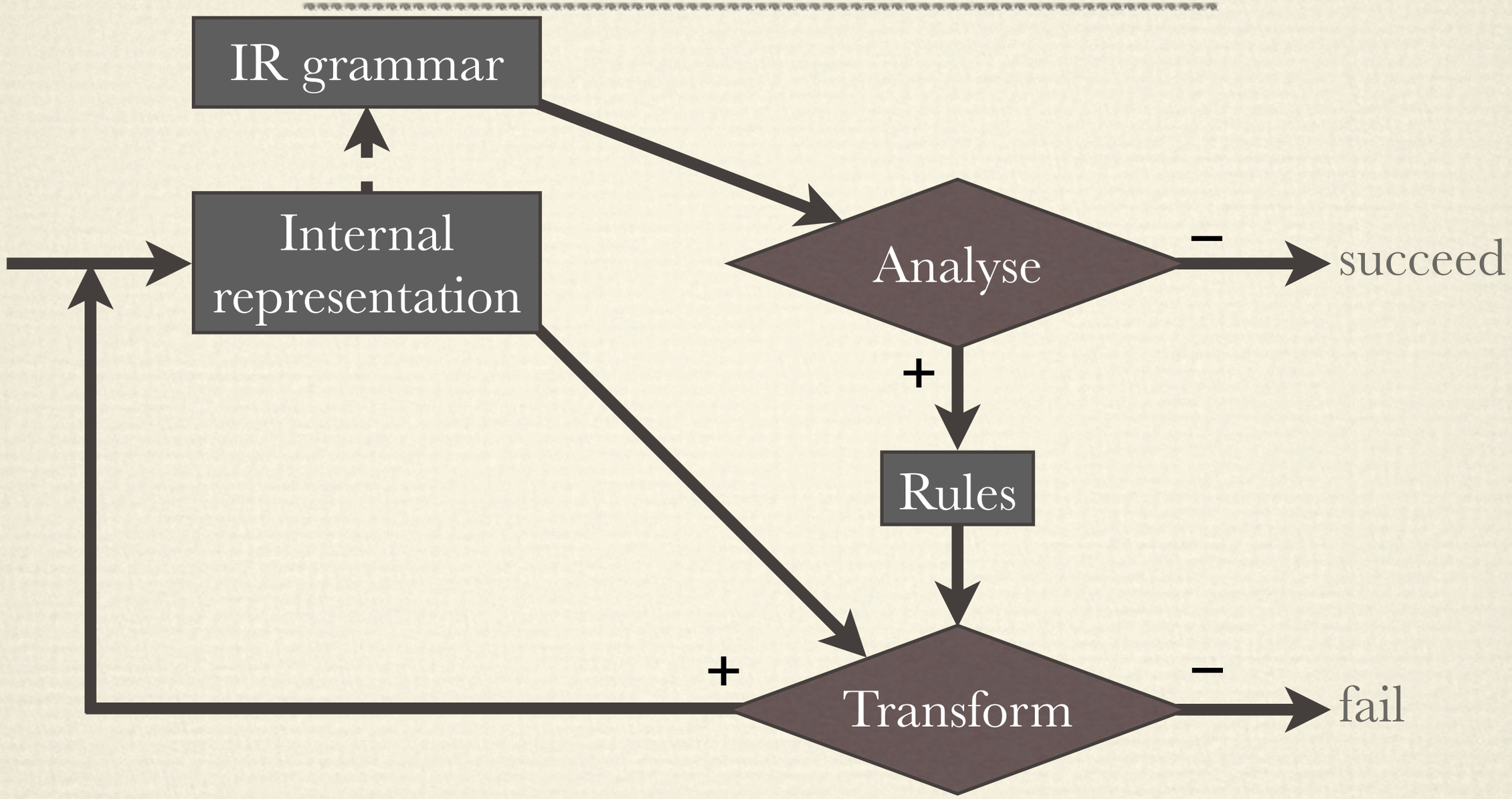
Grammarware metamodel
perfect



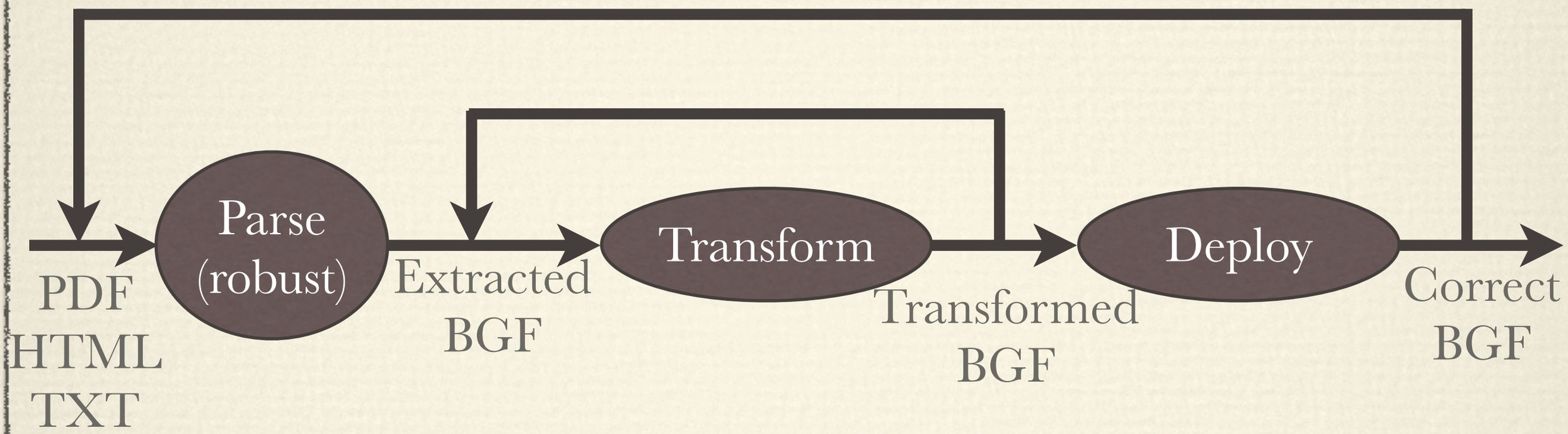
Grammarware metamodel
tiled



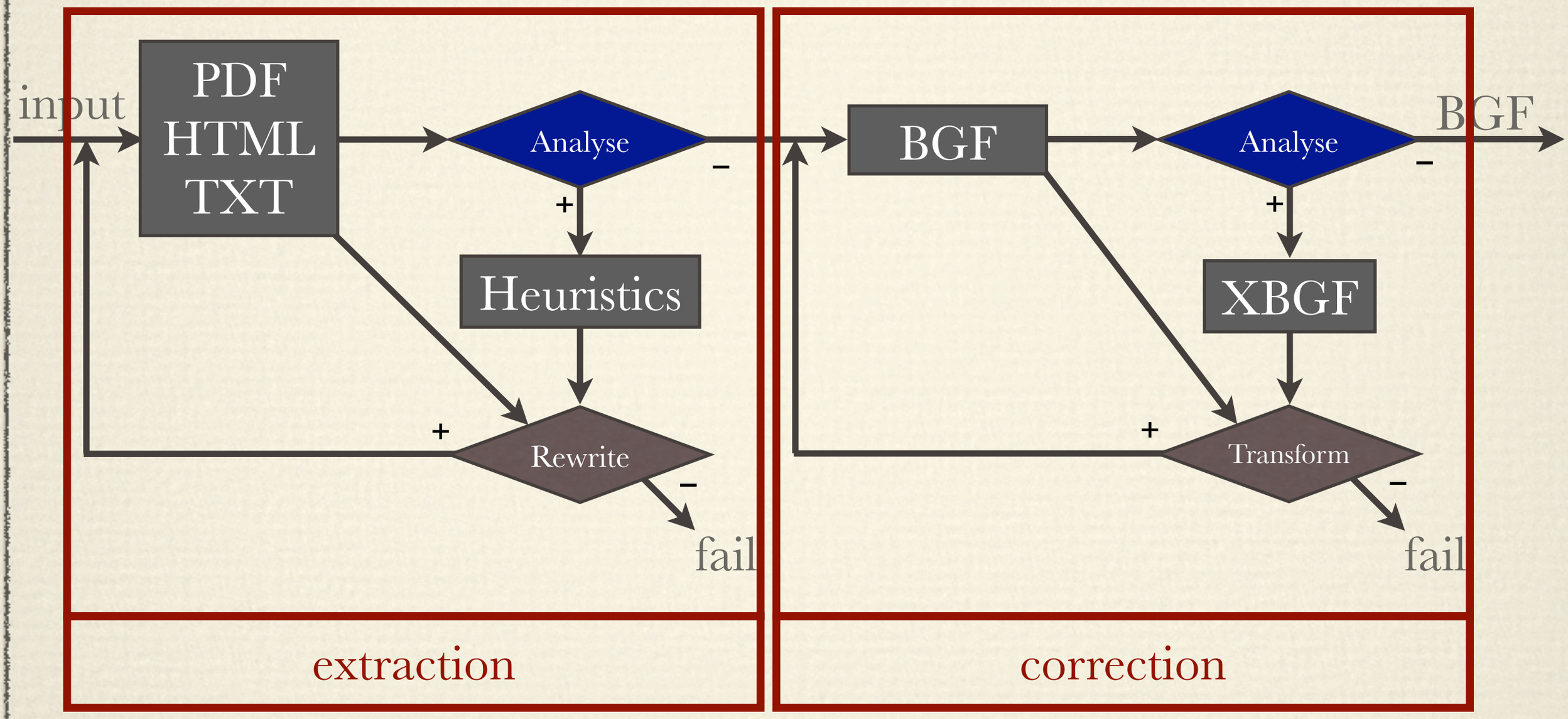
Grammar**ware** metamodel



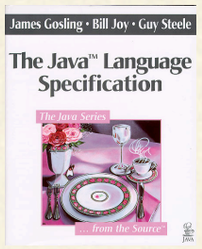
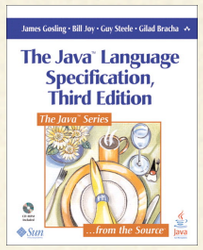
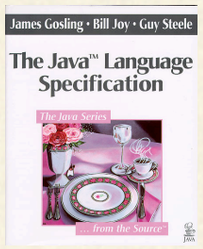
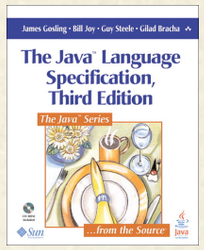
Grammar recovery
EASY



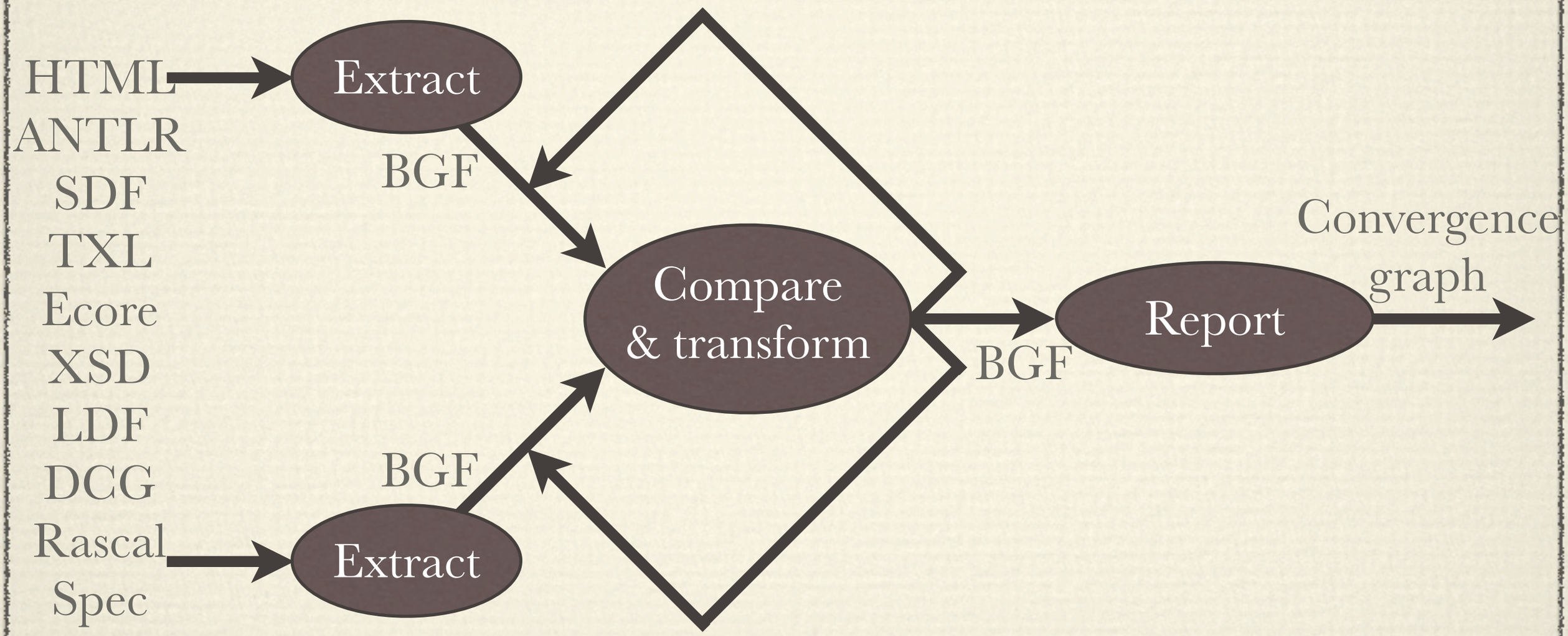
Grammar recovery
ART



Extraction of Java grammars

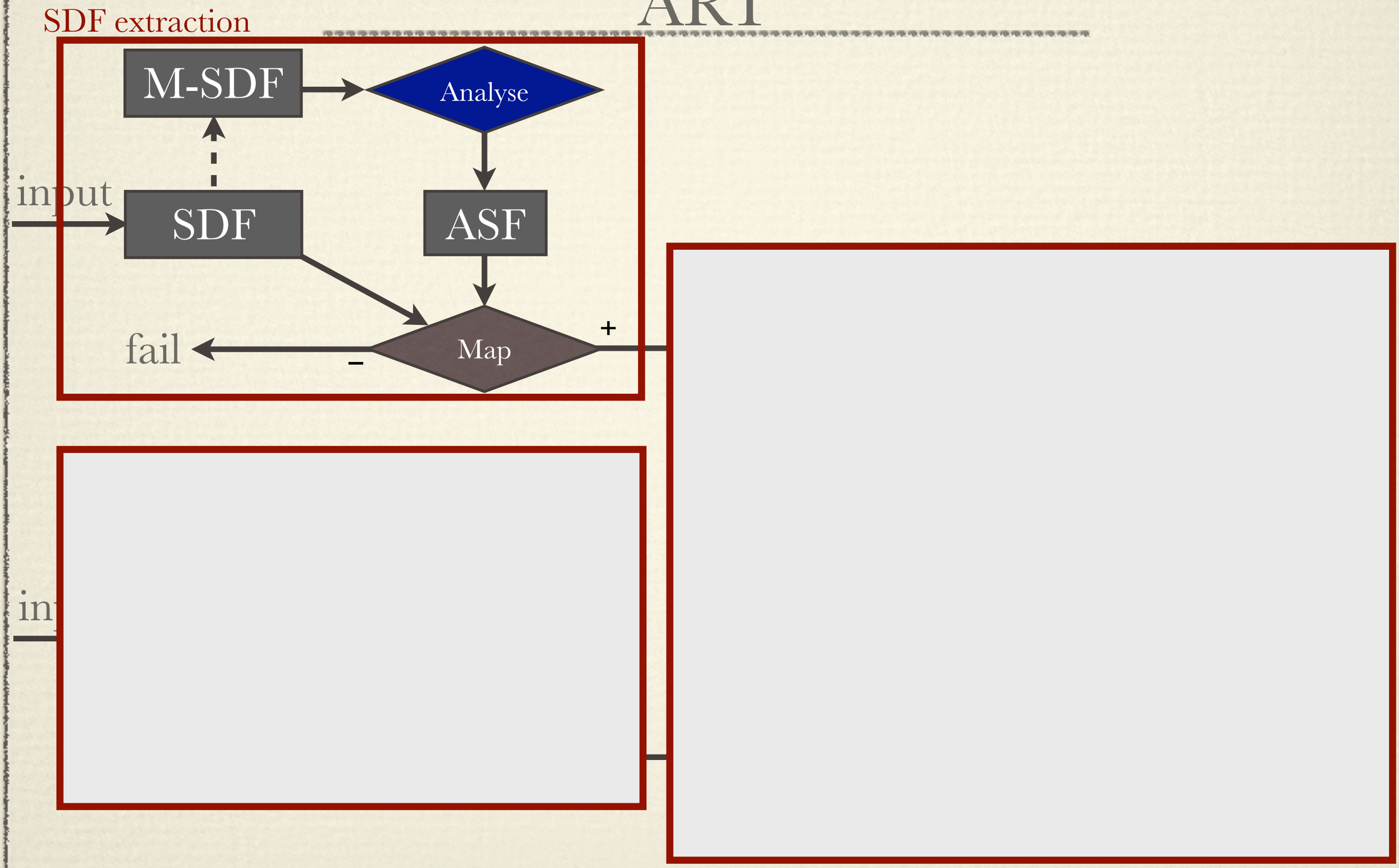
	 **impl1**	 **impl2**	 **impl3**	 **read1**	 **read2**	 **read3**	**Total**
Arbitrary lexical decisions	2	109	60	1	90	161	423
Well-formedness violations	5	0	7	4	11	4	31
Indentation violations	1	2	7	1	4	8	23
Recovery rules	3	12	18	2	59	47	141
◦ Match parentheses	0	3	6	0	0	0	9
◦ Metasymbol to terminal	0	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

Grammar convergence
EASY



Grammar convergence

ART



SDF extractor

★Use SDF of SDF

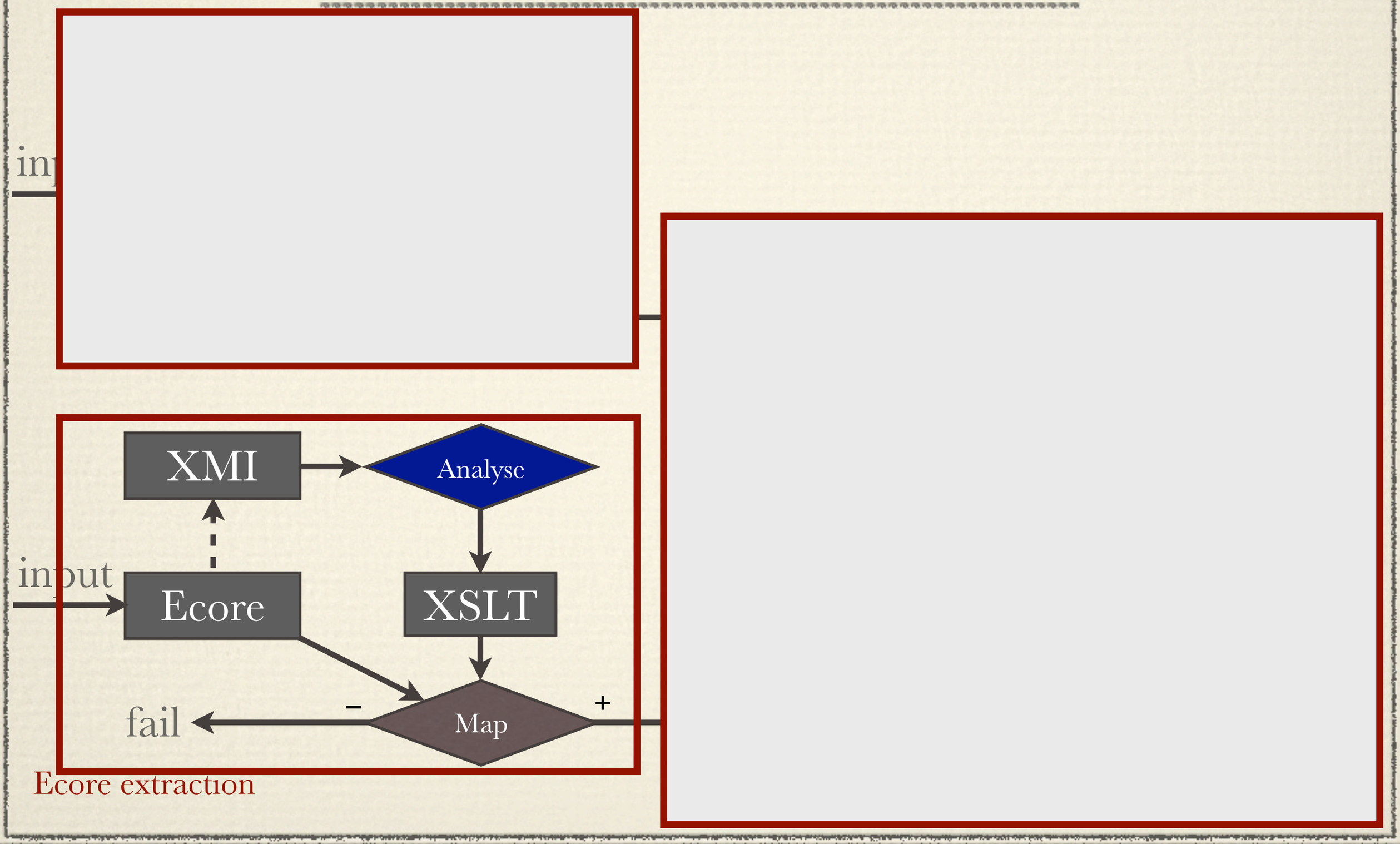
★Use ASF over it

★Construct BGF via XML

```
[transform-a-production]
&C1 := sort2chardata(&N1),
&E2 := trafoSymbols(&Ss1)
=======================
trafoProd ( &Ss1 -> &N1 &As1 ) =
<bgf:production>
 <nonterminal>&C1</nonterminal>
 &E2
</bgf:production>

[transform-empty-definition-of-nonterminal]
trafoSymbols() =
<bgf:expression>
 <epsilon/>
</bgf:expression>

[transform-definition-that-is-not-a-sequence]
trafoSymbols(&S1) = trafoSymbol(&S1)

[transform-a-definition-that-is-a-nontrivial-sequence]
&S1 &S2 &S*3 := &Ss1,
&C*1 := mapTrafoSymbol(&Ss1)
=======================
trafoSymbols(&Ss1) =
<bgf:expression>
 <sequence>
  &C*1
 </sequence>
</bgf:expression>
```

Grammar convergence

ART



Ecore extractor

★Use XMI representation

★Use XSLT over it

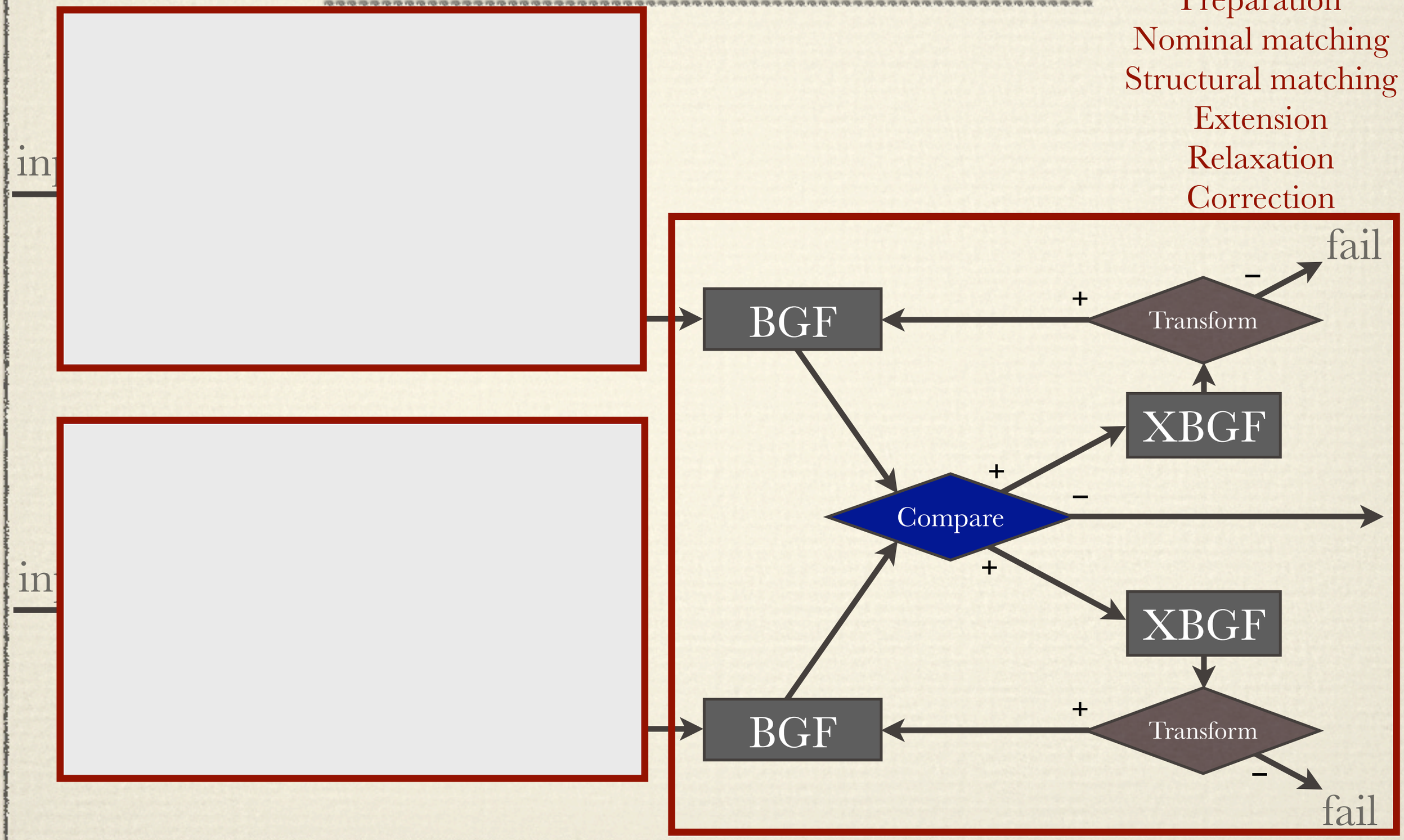
★Map classes to nonterminals, etc

```
<xsl:template match="eClassifiers">
  <xsl:variable name="ourEType" select="concat('#//',./@name)"/>
  <xsl:variable name="ourSuperType" select="substring(@eSuperTypes,4)"/>
  <xsl:choose>
    <xsl:when test="@abstract='true'">
      <bgf:production>
        <nonterminal>
          <xsl:value-of select="./@name"/>
        </nonterminal>
        <xsl:choose>
          <xsl:when test="//eClassifiers[@eSuperTypes=$ourEType]">
            <bgf:expression>
              <choice>
                <xsl:for-each select="//eClassifiers[@eSuperTypes=$ourEType]">
                  <bgf:expression>
                    <nonterminal>
                      <xsl:value-of select="./@name" />
                    </nonterminal>
                  </bgf:expression>
                </xsl:for-each>
              </choice>
            </bgf:expression>
          </xsl:when>
          <xsl:otherwise>
            <bgf:expression>
              <epsilon/>
            </bgf:expression>
          </xsl:otherwise>
        </xsl:choose>
      </bgf:production>
    </xsl:when>
    <xsl:when test="@name='DocumentRoot'"/>
    <xsl:when test="@xsi:type='ecore:EDataType'"/>
```

Grammar convergence

ART

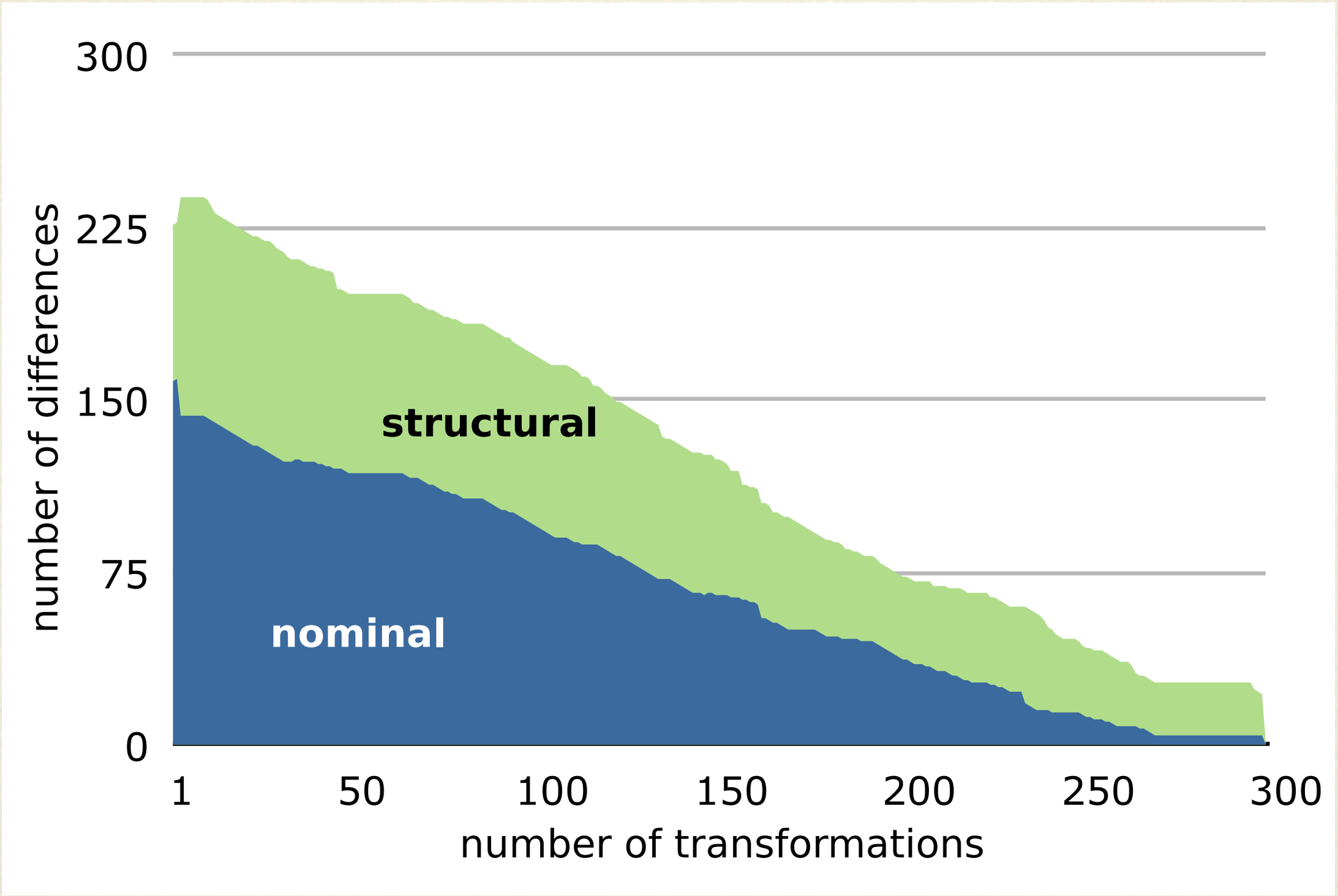
Preparation
Nominal matching
Structural matching
Extension
Relaxation
Correction



Grammar comparison

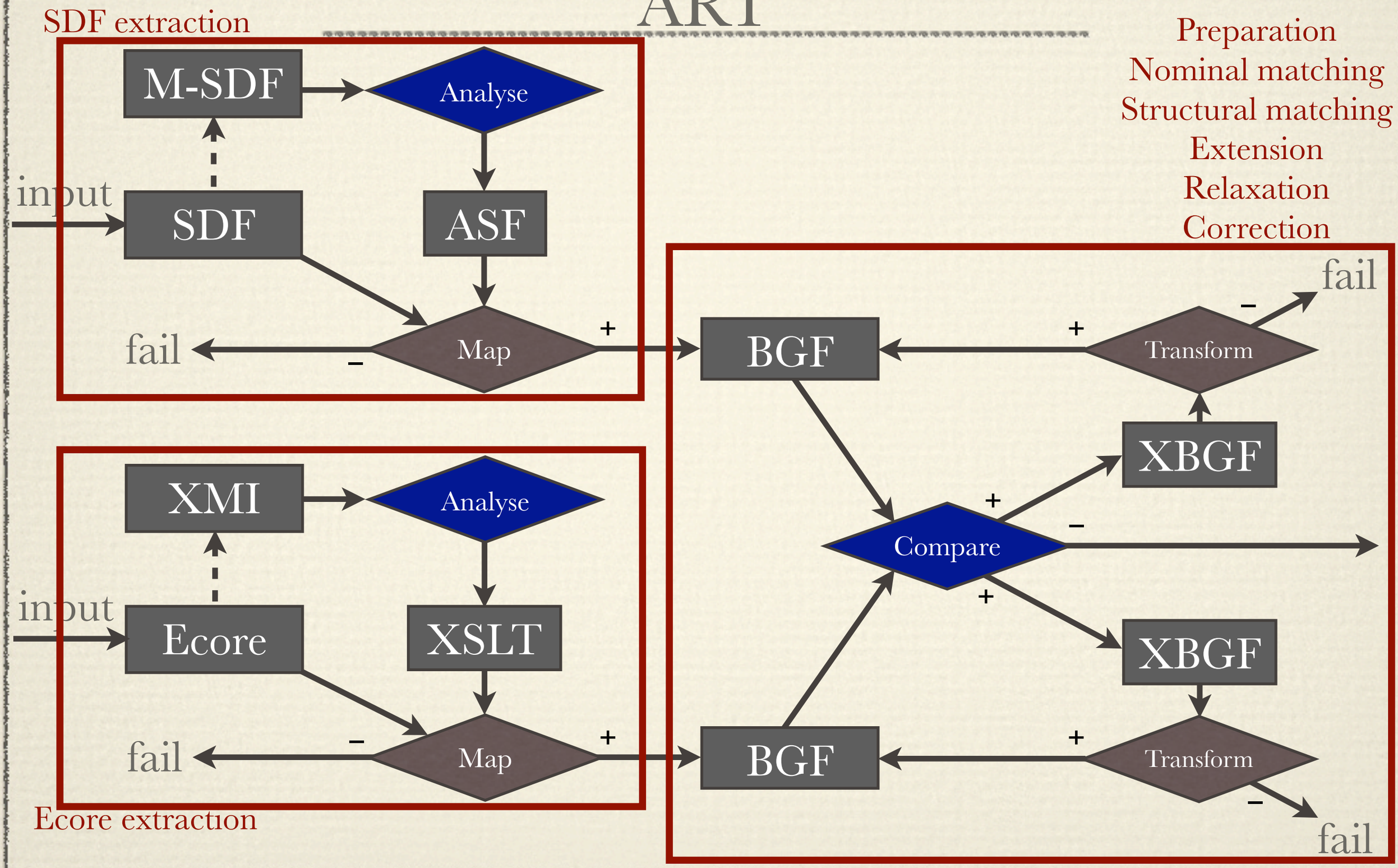
$ gdt ecore.bgf xsd.bgf
Normalizing ecore.bgf.
Normalizing xsd.bgf.
Diffing ecore.bgf and xsd.bgf.
 - Names of defined nonterminals differ.
   - Intersection [Program, Function, Argument].
   - ecore.bgf only: [Exp, LiteralExp, ArgumentExp, IfThenElseExp, ApplyExp, BinaryExp, PlusExp, MinusExp, EqualExp].
   - xsd.bgf only: [Fragment, Expr, Literal, Binary, Ops, IfThenElse, Apply].
 - Comparisons per (common) nonterminal:
   - Ok: Program.
   - Fail (1/1): Function.
     - [], ,([s(name, v(string)), +s(argument, n(Argument)), s(definition, n(Exp))])
     vs.
     - [], ,([s(name, v(string)), +s(arg, v(string)), s(rhs, n(Expr))])
   - Ok: Argument.
 - Roots differ: [] vs. [Program, Fragment].
$

Grammar measurement



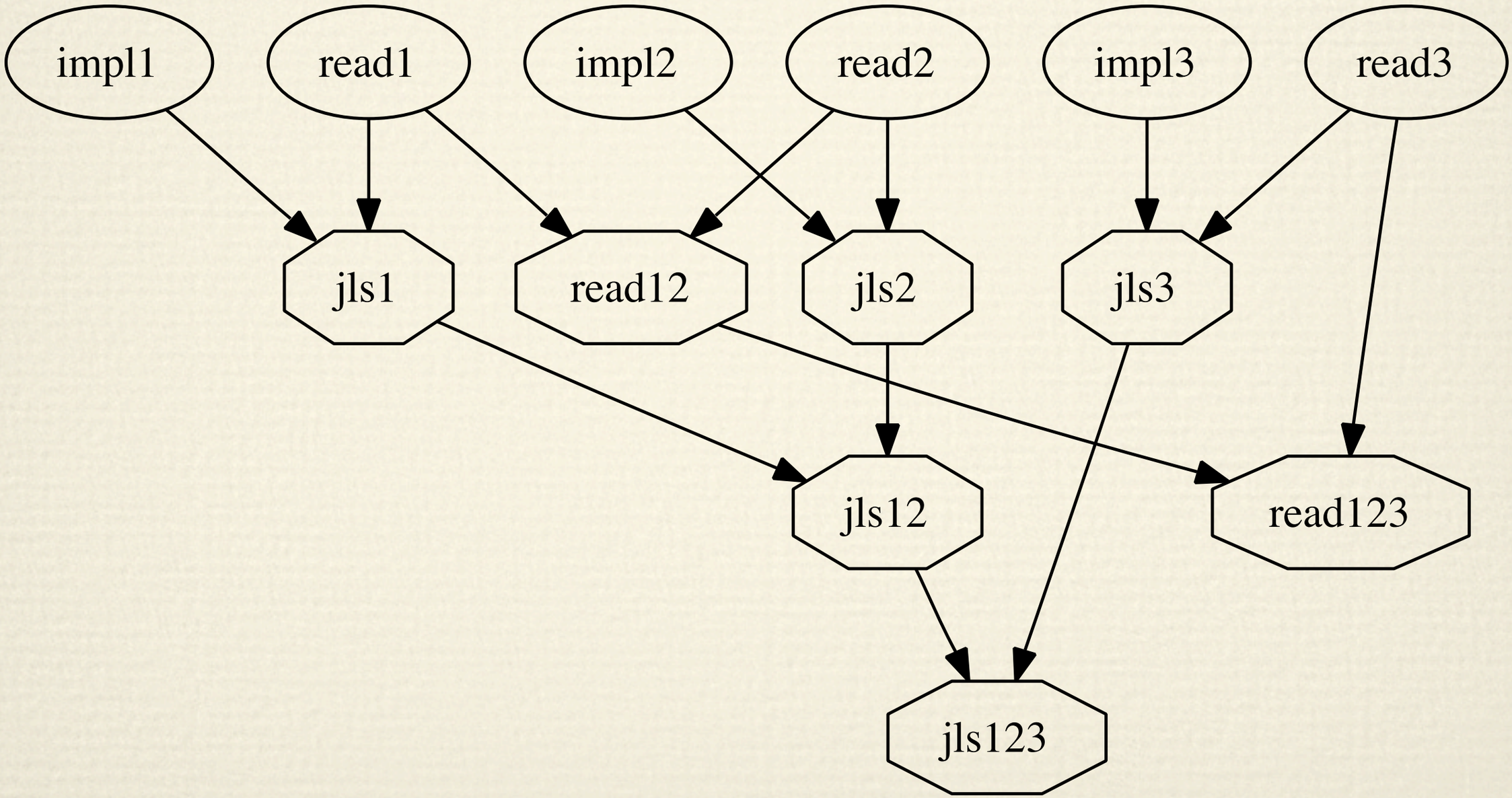
Grammar convergence

ART



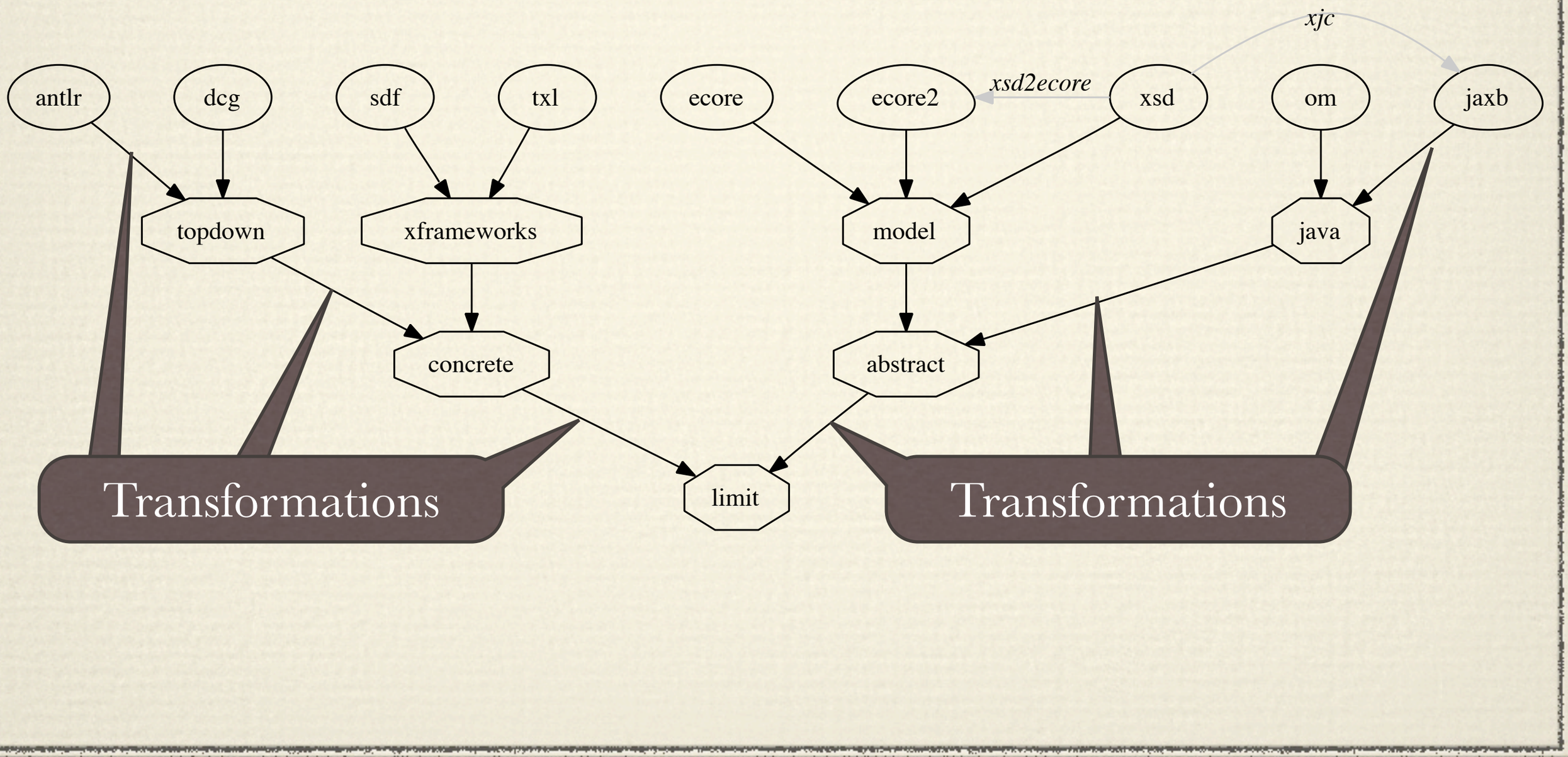
Convergence graph

Different versions of a language as documented by specifications



Relationships between languages

Different versions of the same language

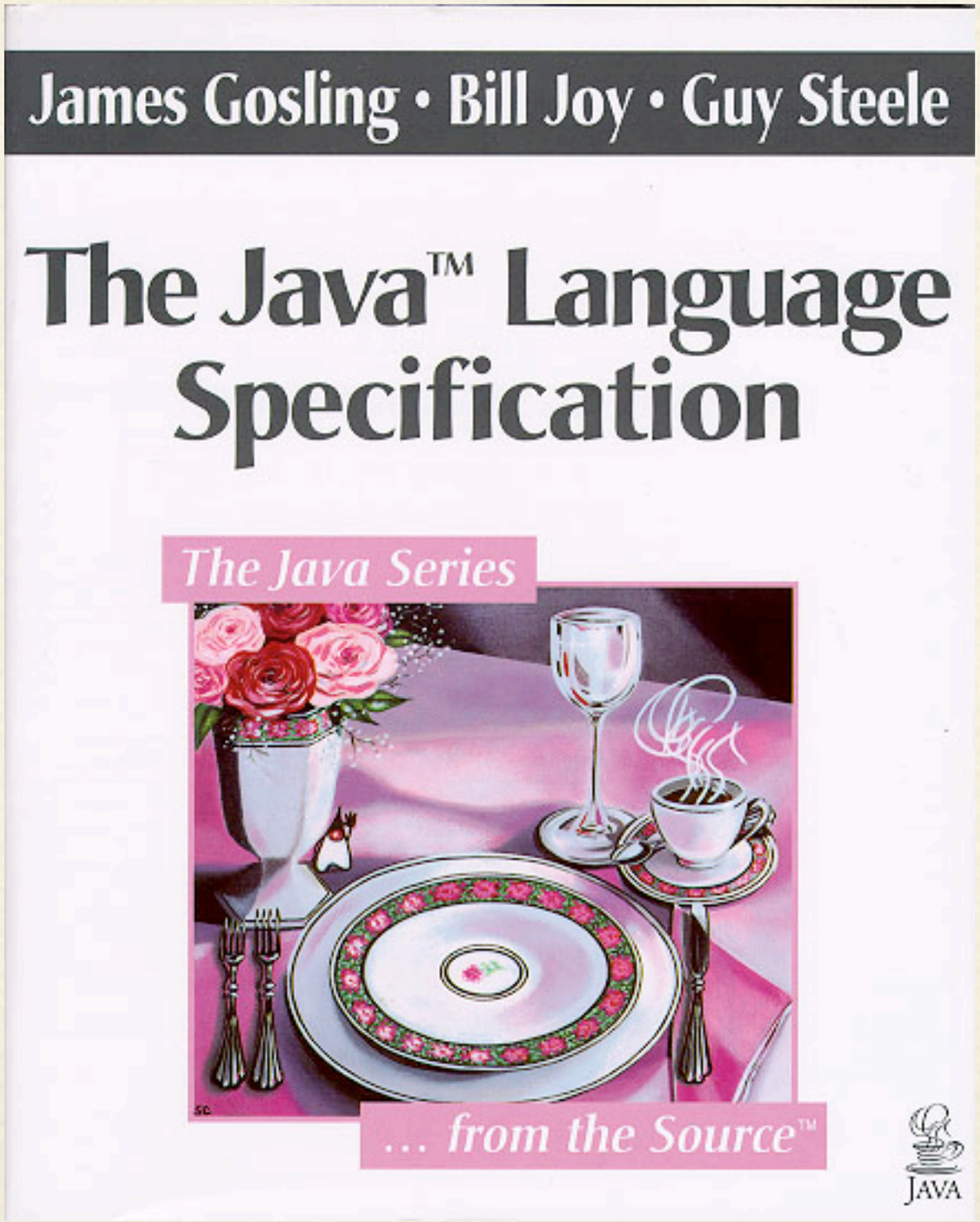


JLS convergence results

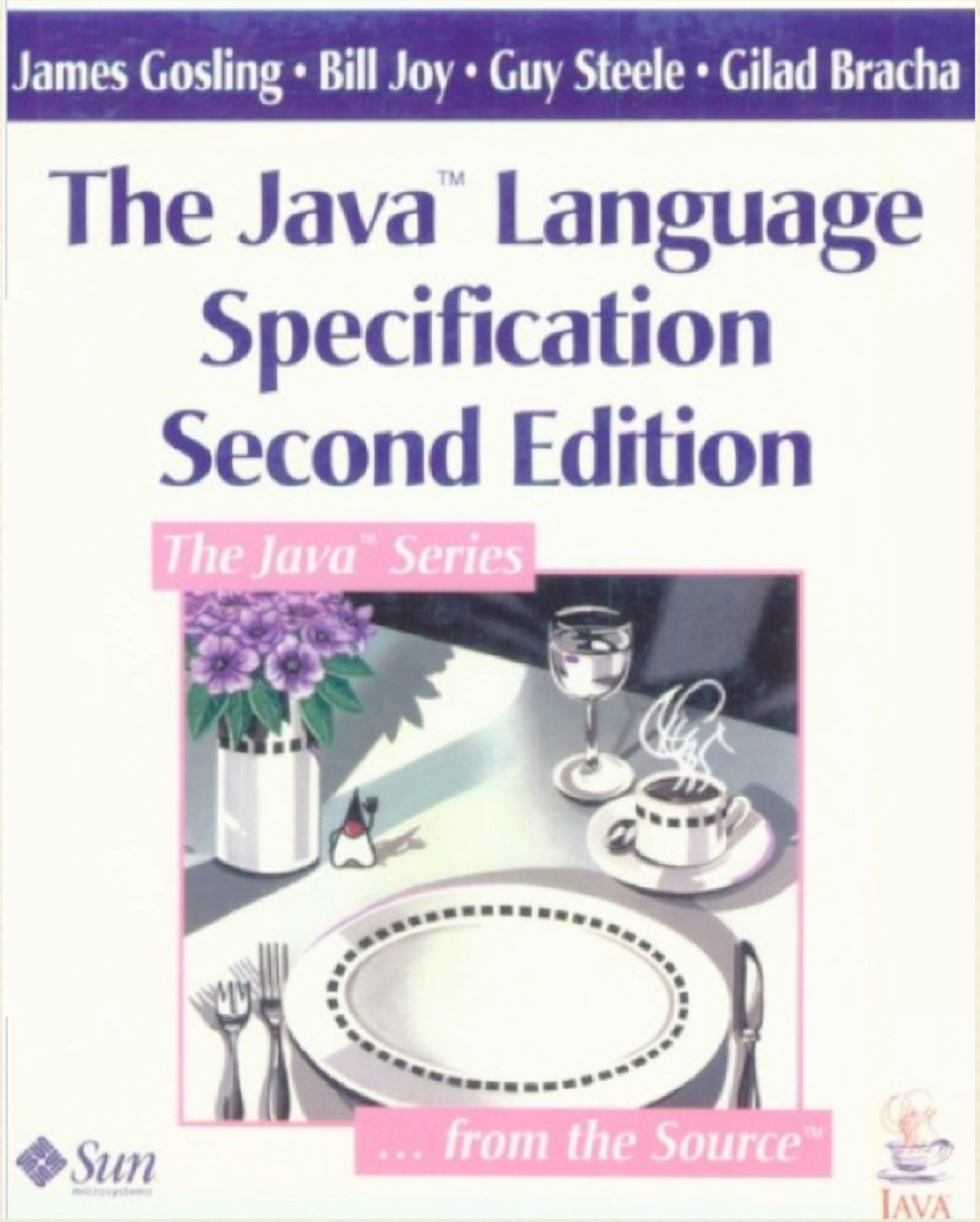
	jls1	**jls12**	**jls123**	**jls2**	**jls3**	**read12**	**read123**	**Total**
Number of lines	682	5114	2847	6774	10721	1639	3082	30859
Number of transformations	67	290	111	387	544	77	135	1611
○ Semantics-preserving (§4.2.2)	45	231	80	275	381	31	78	1121
○ Semantics-increasing/-decreasing	22	58	31	102	150	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

Convergence reveals relationships

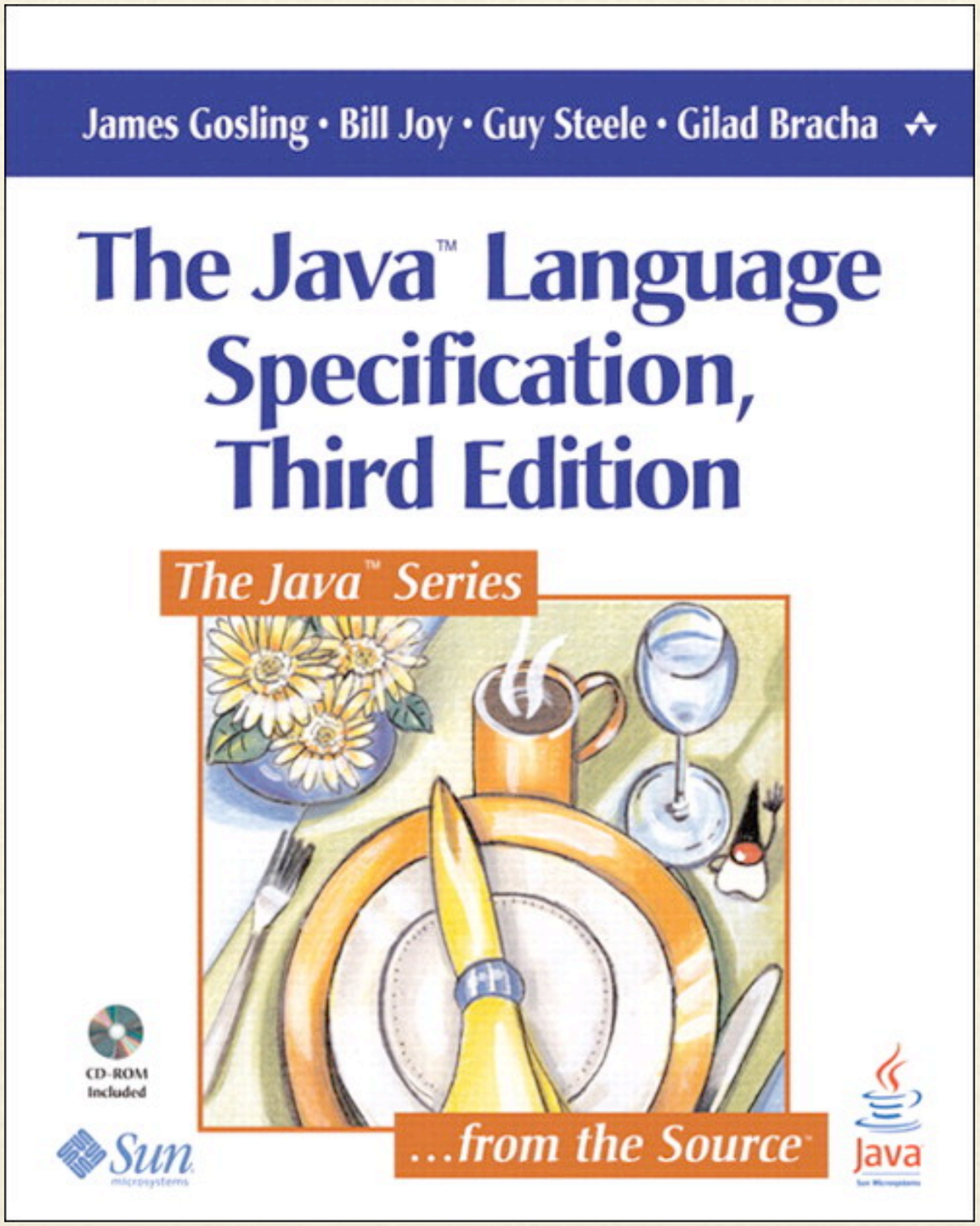
Language documentation



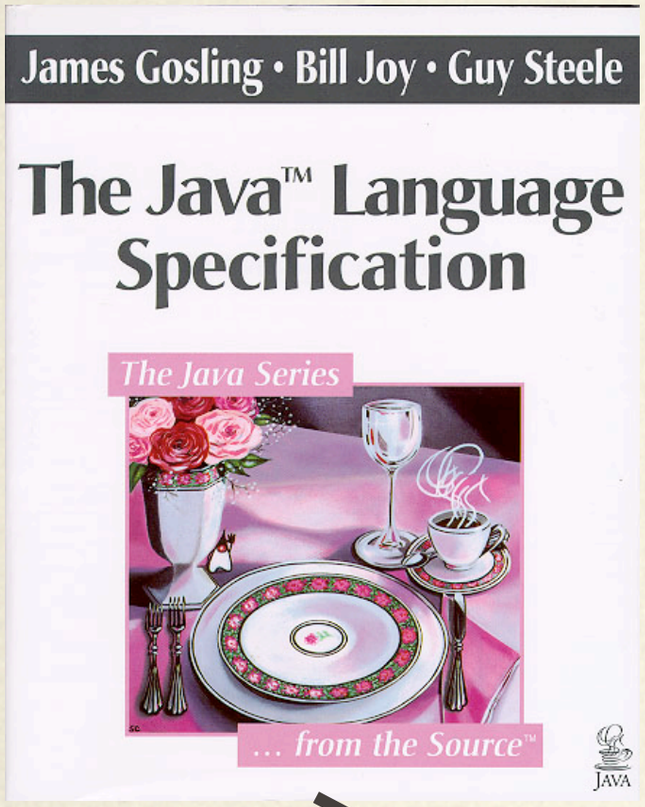
Language documentation

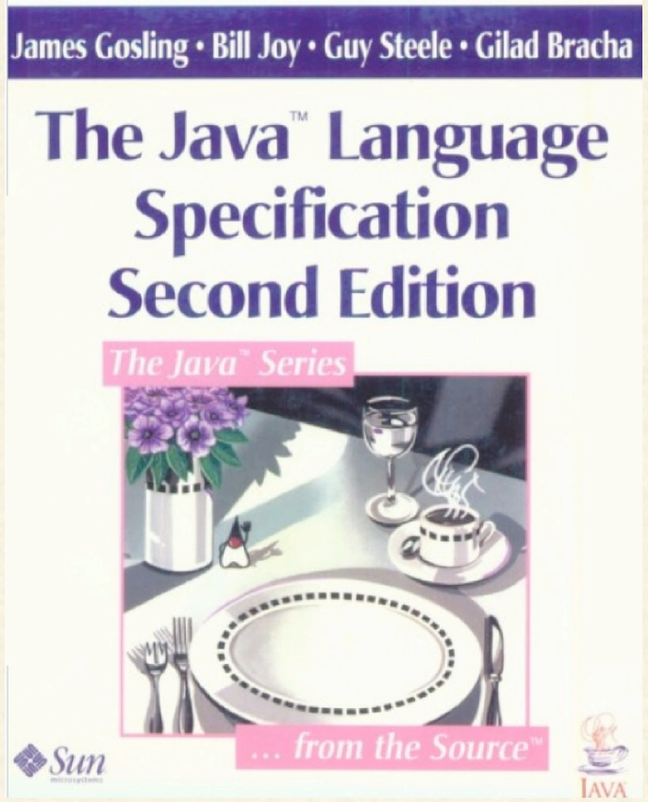


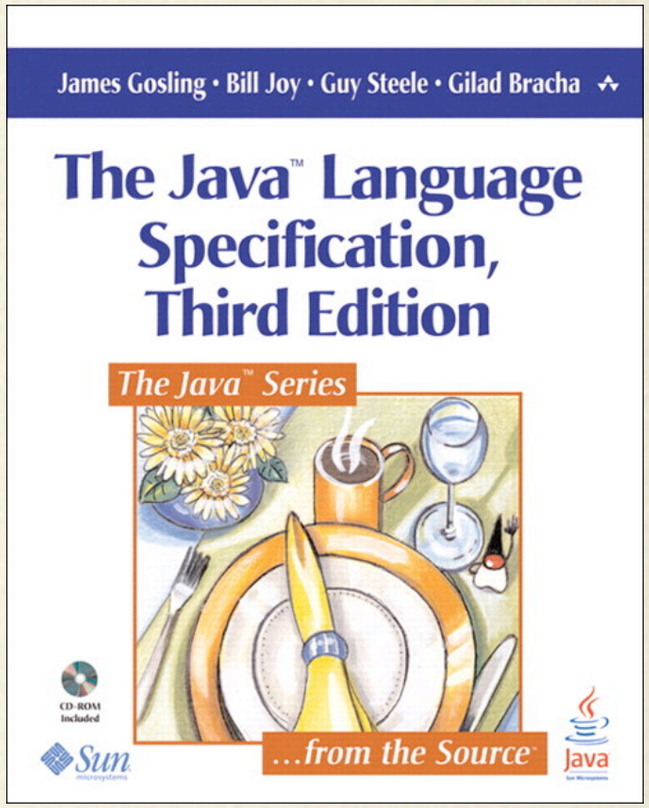
Language documentation

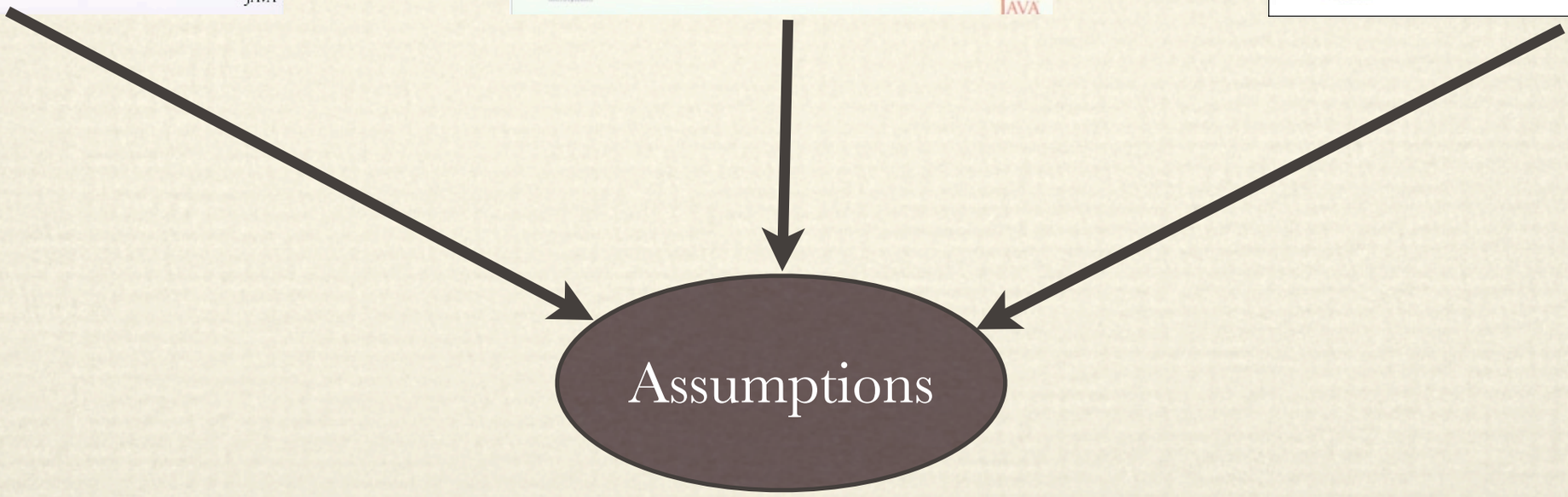


Language documentation

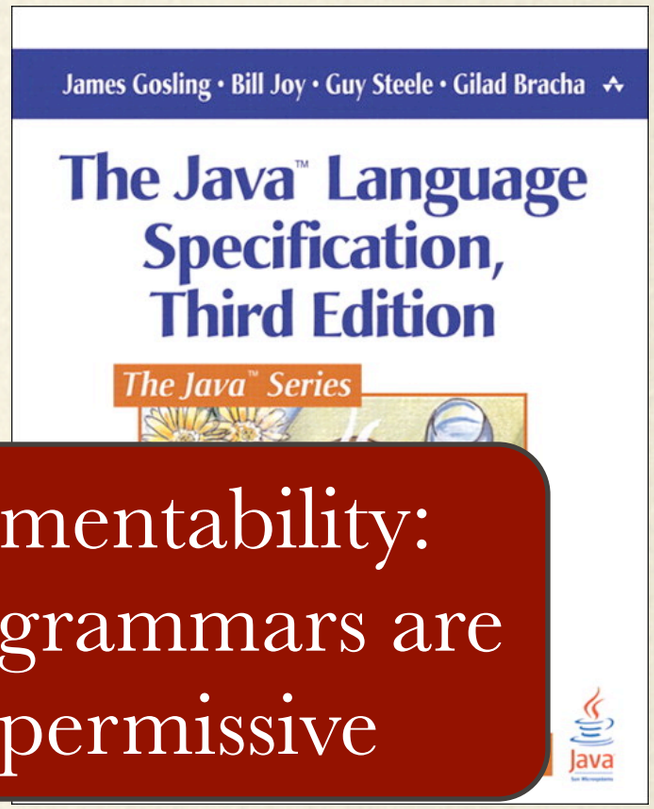








Language documentation



Within one spec: consistent & equivalent content

Backward compatibility: newer grammars include older ones

Implementability: some grammars are more permissive

Operability: the grammars work

Executability: examples run

Assumptions

Language documentation

★ Proper treatment for heterogeneous content

★ Explicit relations among parts of a spec

★ Effortless automated grammar extraction

★ Learn from wikis, eBooks, browsable grammars

★ Use IR and NLG

★ Verification as a stable part of grammar life cycle

Future work

★ Automated grammar recovery (@grammarware)

★ Grammar testing (@reallynotabba)

★ Rascal grammar notation (@jurgenvinju)

★ Analysing SLPS Zoo for ambiguities (@bbasten)

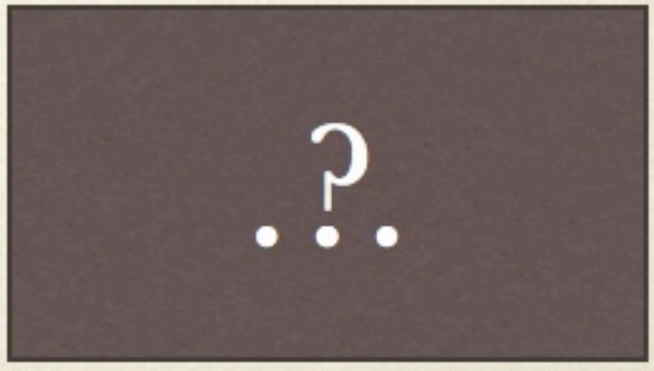
★ CTS DSL (@anyahelene)

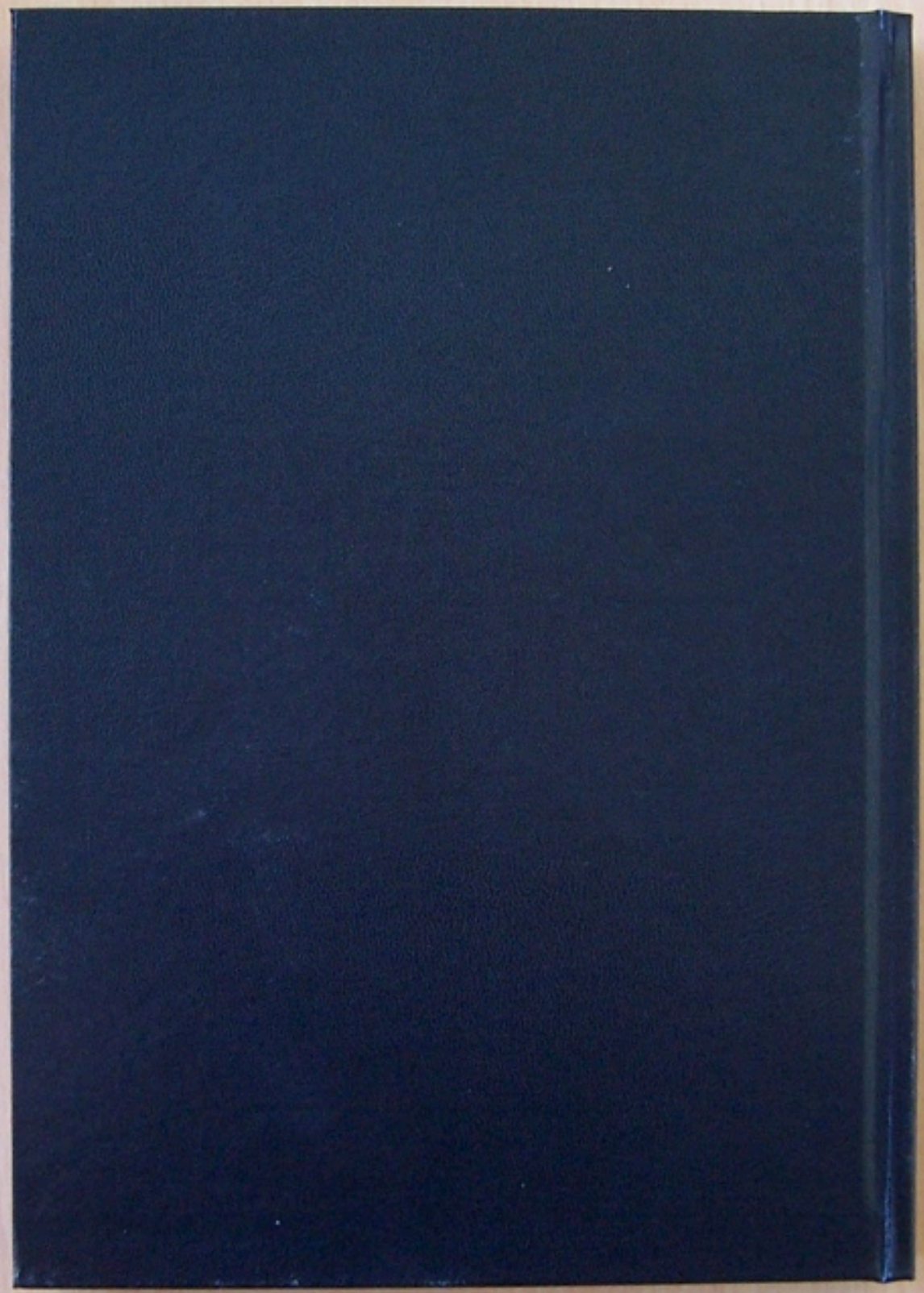
★ … (@…)

Open questions

- ★ Suggestive metrics for convergence
- ★ Inference of converging transformations
- ★ Coupled evolution a.o. of relationships
- ★ Proofs about properties of XBGF
- ★ Reversibility (trafo) & round-tripping (extract-export)
- ★ Metasyntax extentions & metametasyntax
- ★ Bridging grammarware and modelware

Acknowledgements





The End