Recovery, Convergence and Documentation of Languages

Doctoral defence of
Drs. ir. Vadim V. Zaytsev

PROGRAMMER'S REFERENCE MANUAL

Fortran

AUTOMATIC CODING SYSTEM

FOR THE IBM 704

Recovery, Convergence and Documentation of Languages

by
Vadim Zaytsev

September 14, 2010

promotoren: prof.dr. R. Lämmel
prof.dr. C. Verhoef

Dit onderzoek werd ondersteund door de Nederlandse Organisatie voor Wetenschappelijk Onderzoek via:
This research has been sponsored by the Dutch Organisation of Scientific Research via:

NWO 612.063.304 *LPPR: Language-Parametric Program Restructuring*

Acknowledgements

Working on a PhD is supposed to be an endeavour completed in seclusion, but in practice one cannot survive without the help and support from others, fruitful scientific discussions, collaborative development of tools and papers and valuable pieces of advice.

My work was supervised by Prof. Dr. Ralf Lämmel and Prof. Dr. Chris Verhoef, who often believed in me more than I did and were always open to questions and ready to give expert advice. They have made my development possible.

LPPR colleagues — Jan Heering, Prof. Dr. Paul Klint, Prof. Dr. Mark van den Brand — have been a rare yet useful source of new research ideas.

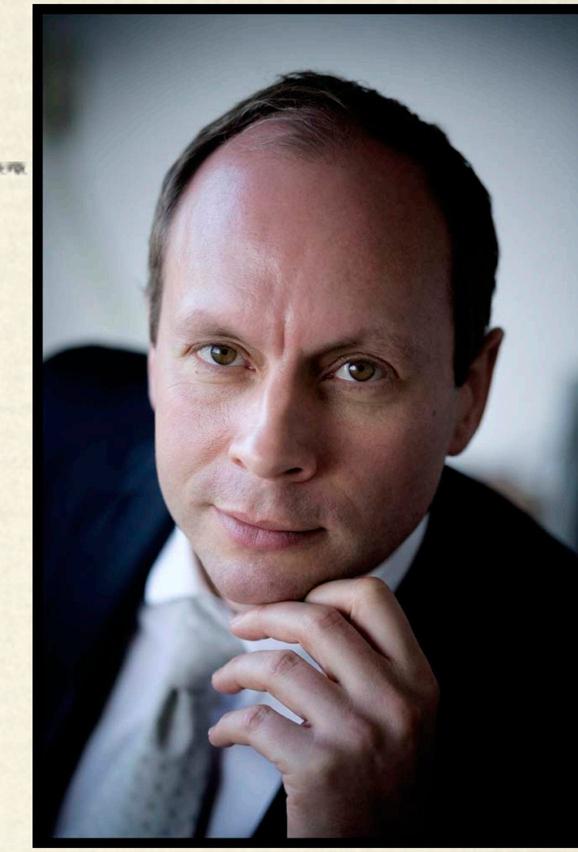
All thesis reading committee members have dedicated a lot of attention to my work and delivered exceptionally useful feedback on the late stage of the research: Prof. Dr. Jean Bézivin, Dr. Jean-Marie Favre, Prof. Dr. Willem Jan Fokkink, Prof. Dr. Paul Klint, Dr. Steven Klusener. I am also grateful for Cor-Paul Bezemer and Toon Verwaest who provided proofreading and correcting services for the Dutch part of this thesis. There have been a lot of insightful discussions in the rooms and hallways of the Vrije Universiteit with Dr. Niels Veerman, Ernst-Jan Verhoeven, Łukasz Kwiatkowski and Johan Vincent de Vries.

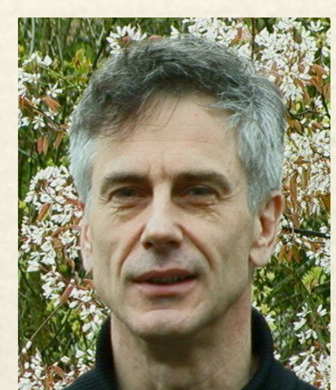
I would like to thank my family that backed me up with complete support and encouragement through the years of research, especially my mother, Dr.ir. Liudmila Zaytseva; my grandmother, Dr. Svetlana Bocheva; my grandfather, Prof. Dr.ir. Alexander Bochev; my uncle, Dr. Michael Bochev and my godfather, Prof. Dr. Yuri Bashmakov, MD.

My close friends' understanding, respect and interest in my work was also among the most important things that kept me going: Dr. Alexander Gufan, Dr. Stanislav Tsykavy and Stanislav Rezhabek.

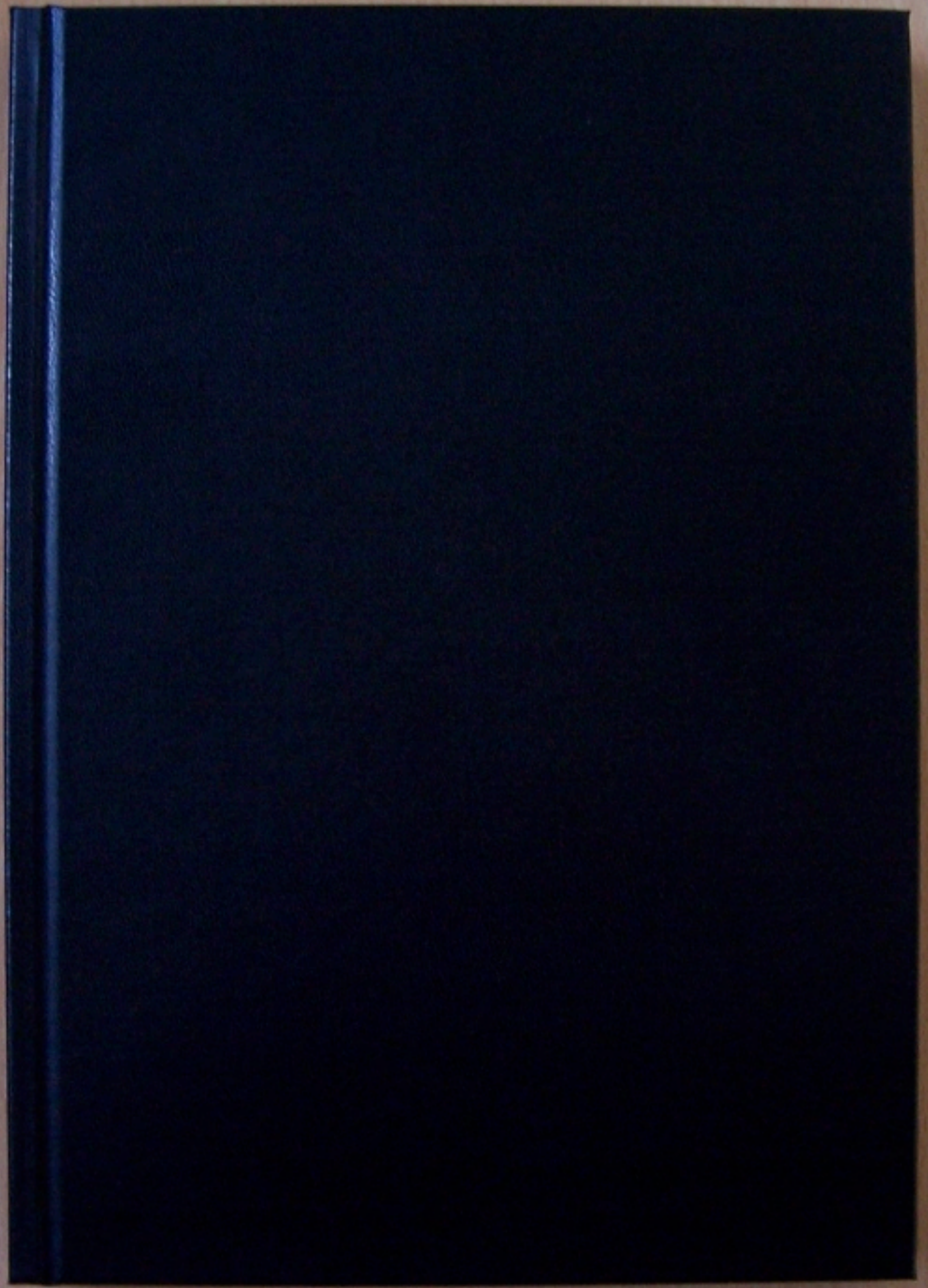
I have also been saved many times from depression and writer's blocks by good music. I cannot name all the artists responsible for that, but the most credit goes to Huddie Ledbetter, William Broonzy, Fulton Allen, Thomas McClennan and Bruce Springsteen.

Acknowledgements





?

Outline

Recovery,
Convergence
and
Documentation
of Languages

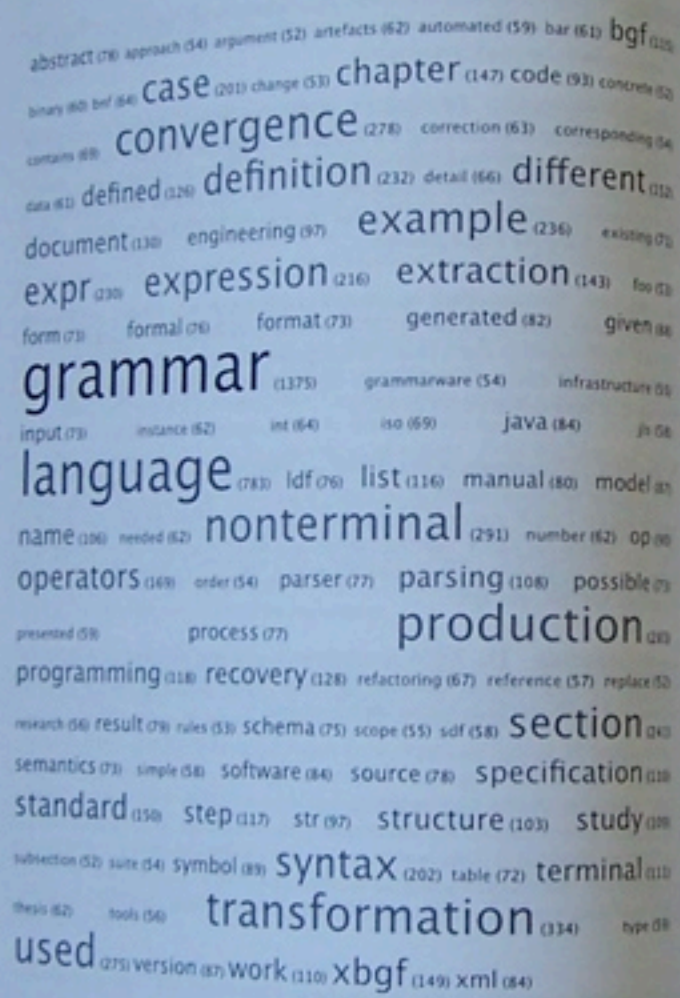


Figure 1.1: Tag cloud of the text of this thesis.

Chapter 2

Additional background

The previous chapter, especially section 1.2, has already introduced the most important background concepts. However, we need to provide additional, more detailed information on the research context. This chapter will present the notions used throughout the thesis, explain the existing methods of grammar engineering and define our view on them. Both purpose and importance of grammar transformation, recovery, convergence and documentation should become clear, enabling the remaining chapters to focus directly on the contributions.

2.1 Terminology

There are notions that will be used extensively in this chapter yet could have remained unclear from the previous sections:

A **grammar** is a strict and precise definition of a language in its formal sense (as a set of allowed words). Hence, the grammar defines the structure of a piece of source code. Grammars for mainstream languages used in industry are big, they are not supposed to be read by humans and be manually checked for completeness, correctness and other properties. Instead, an automated approach is taken with an infrastructure accepting a formal grammar as an input and producing a parser, a transformational tool or other grammarware as an output.

The words "schema", "ontology" or "metamodel" are used instead of the word "grammar" in different areas. Schemata and data models are notions related to grammars in database and data manipulation research, although not all data models can be easily mapped to grammars. XML also calls its grammars schemata, whether they conform to XML Schema [75, 208], RELAX NG [34], DTD [20] or any other standard. Ontologies are used in complex matters such as semantic web, business process modelling or artificial intelligence [221]. They mostly fall outside the scope of this thesis because of their complicate nature. Grammar domain is smaller, simpler and does not face the kind of challenges that are typical for ontology alignments.

Grammar definition formalism is any kind of notation for modelling the syntax of a language. It can be textual with only a few basic features for denoting terminals,

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) 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) replace (52) 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)

Outline

Recovery, Convergence and Documentation of Languages

Language: Java

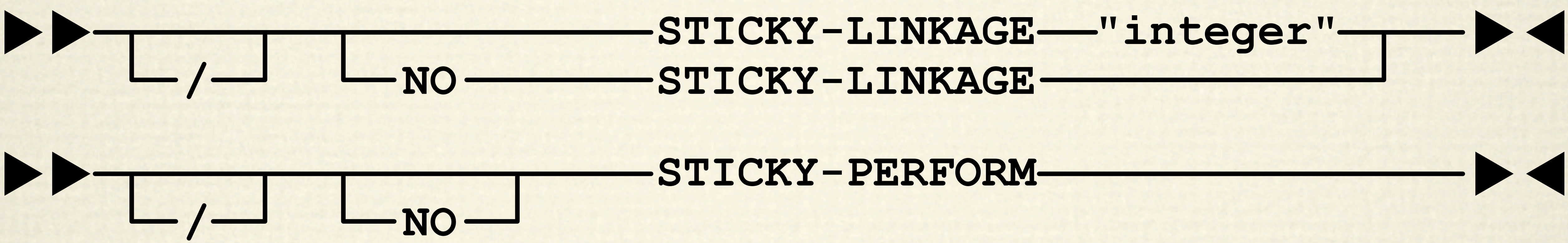
```
import types.*;
import org.antlr.runtime.*;
import java.io.*;

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

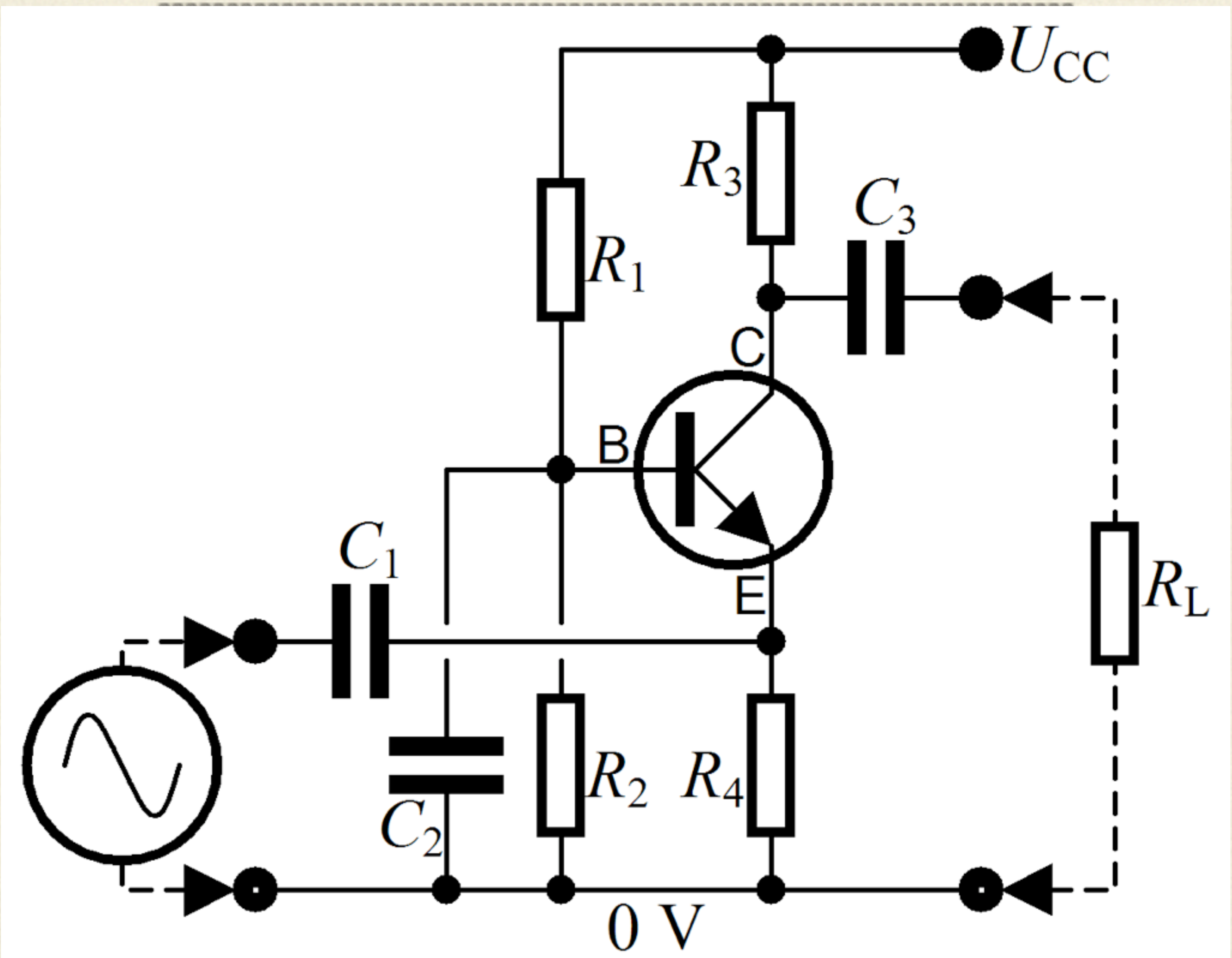
Language: XML (BGF)

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

Language: syntax diagram



Also a language



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

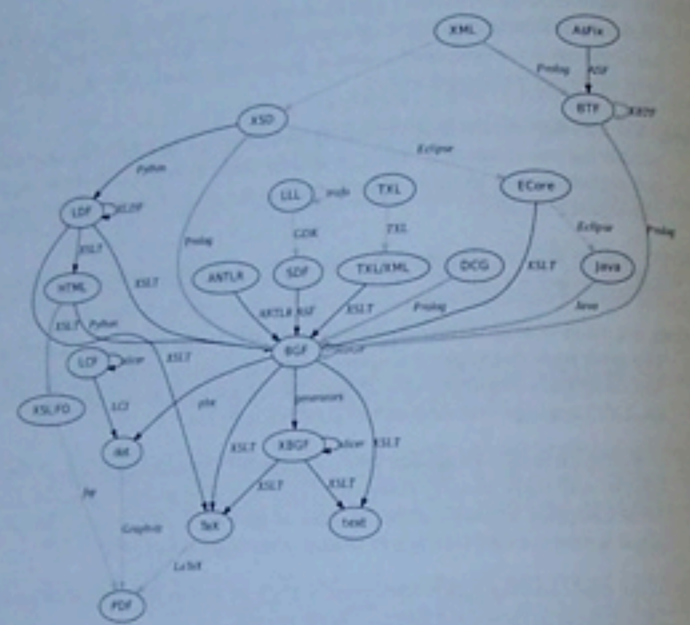


Figure 2.2: Dependency graph showing all language transformations. Thin grey lines denote tools present prior to this research. Thick grey edges are for co-developed transformations.

XPath dialect [173] was used internally by all Python scripts that were working with XML (in particular, Language Convergence Infrastructure), and classic XPath as a standalone tool for analyses, metrics and presentation.

XQuery scripts were generating output when XPath was not able to express the designed benchmarks.

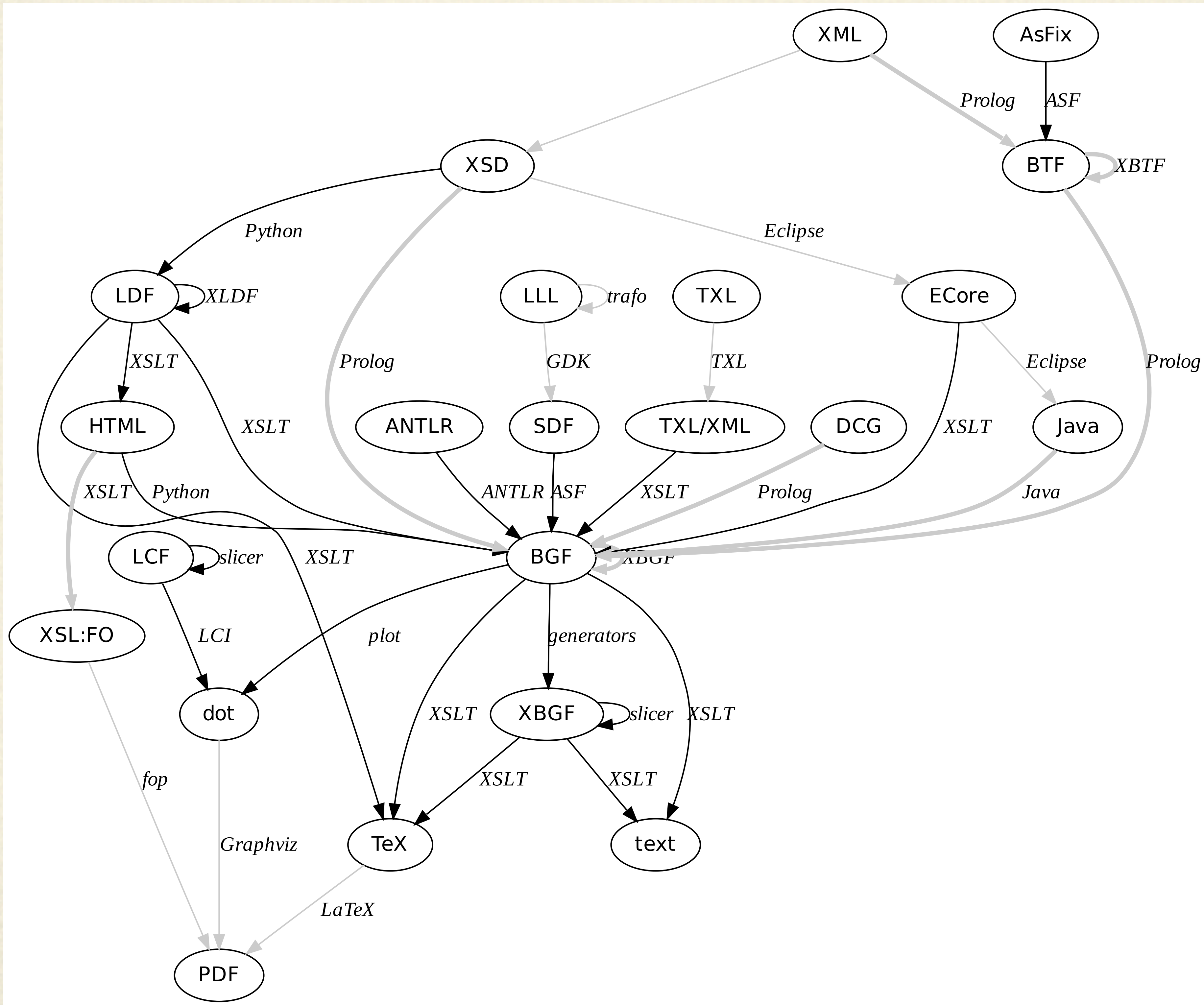
2.13 Transformations used in the thesis

Figure 2.2 shows a dependency graph of all languages and other artefacts of grammar knowledge used in this work. Every node in this graph demonstrates a format, a form or a view of a language or its part. Every edge is a transformation. All solid black edges were designed and implemented as a part of this research project. Thin grey edges

represent external transformation facilities such as those provided by Eclipse Modelling Framework [59], Graphviz [74] or LaTeX. Thick grey edges are for transformations that were co-developed or taken from third parties. We list them below:

- XBGF is a collaboration effort with Prof. Dr. Ralf Lämmel.
- Java2BGF, XSD2BGF, XML2BTF, DCG2BGF and BTF2BGF are courtesy of Prof. Dr. Ralf Lämmel.
- HTML2XSLFO is courtesy of Antenna House, Inc., use permission granted.

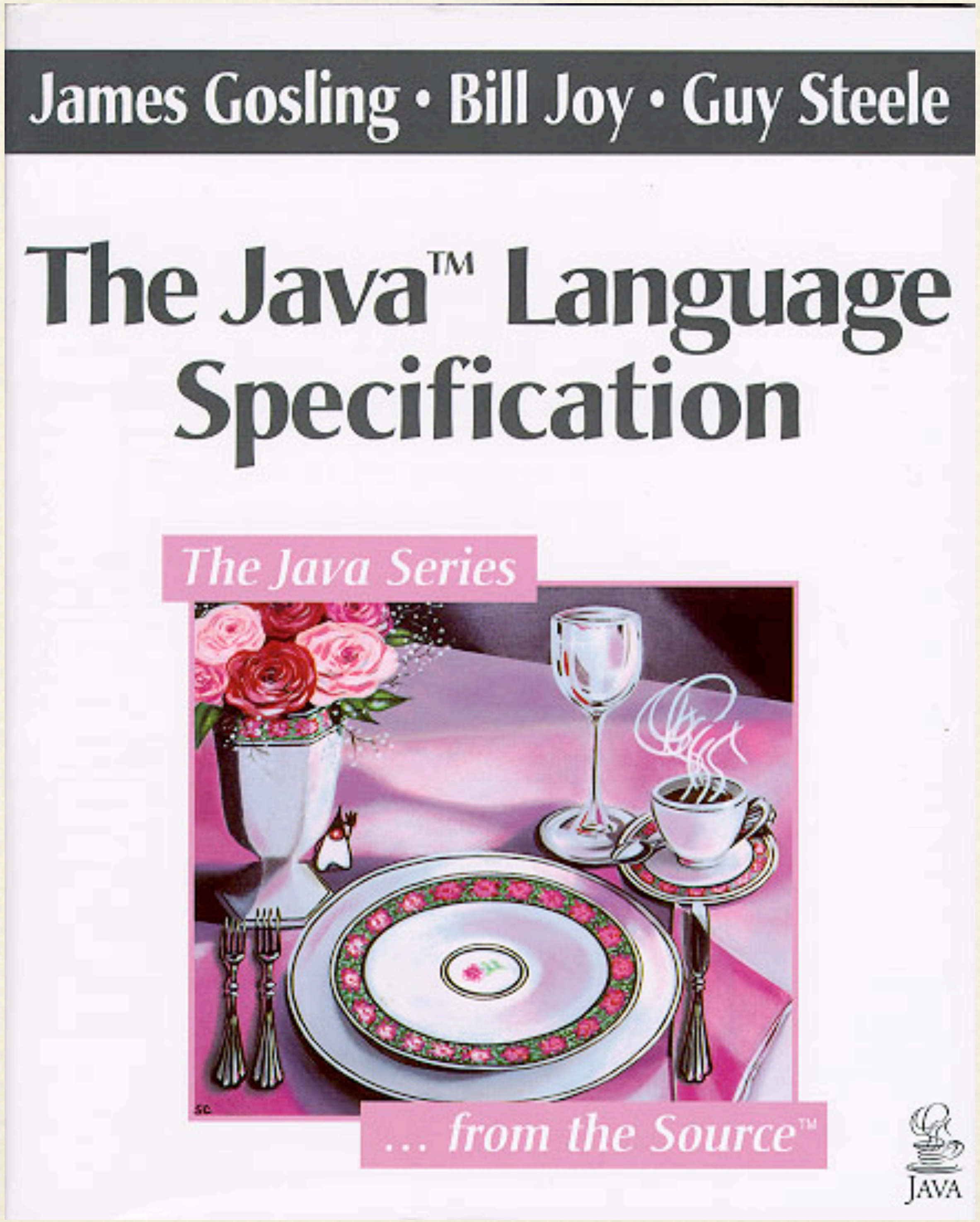
Languages and transformations



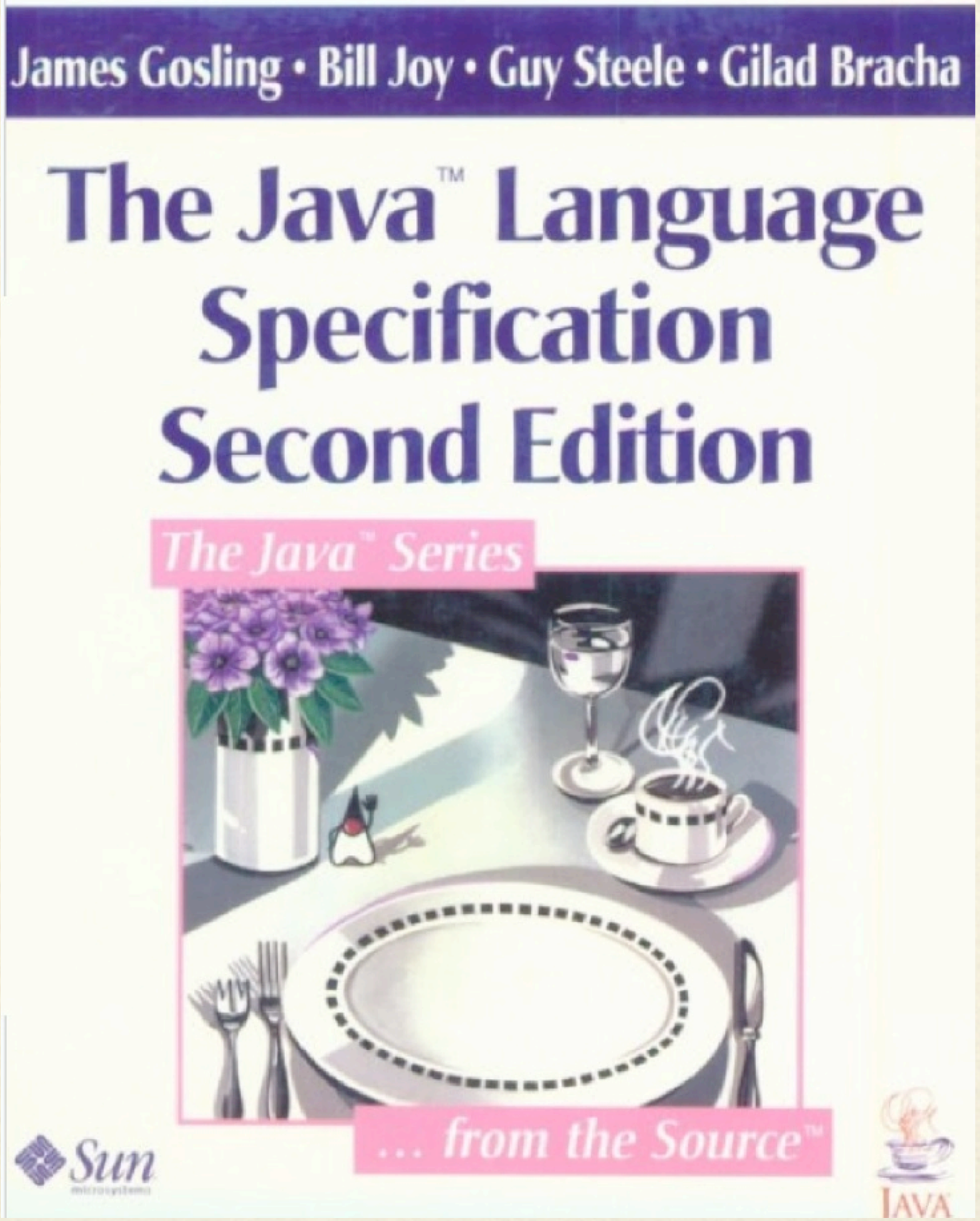
Outline

Recovery,
Convergence
and
Documentation
of Languages

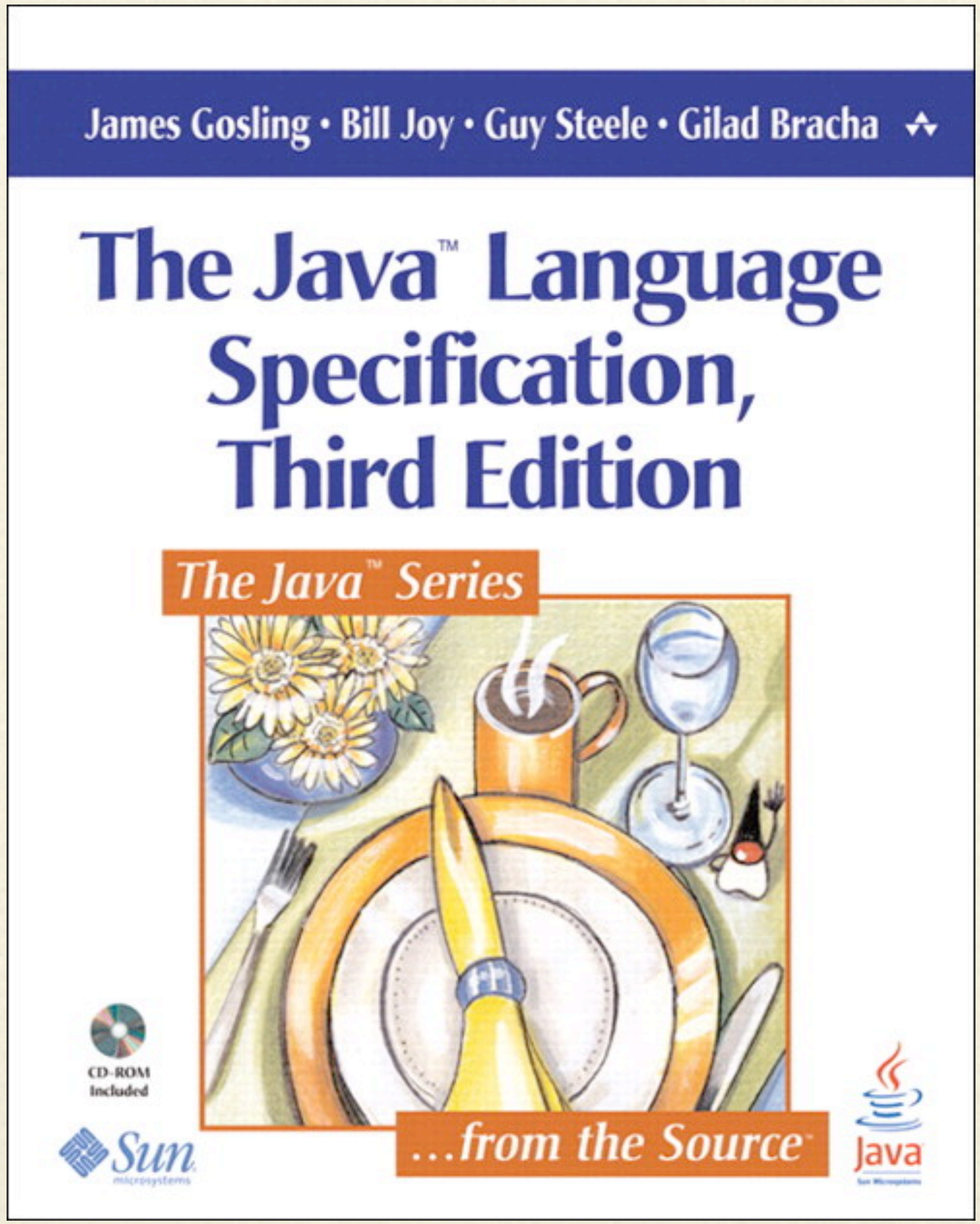
Language documentation



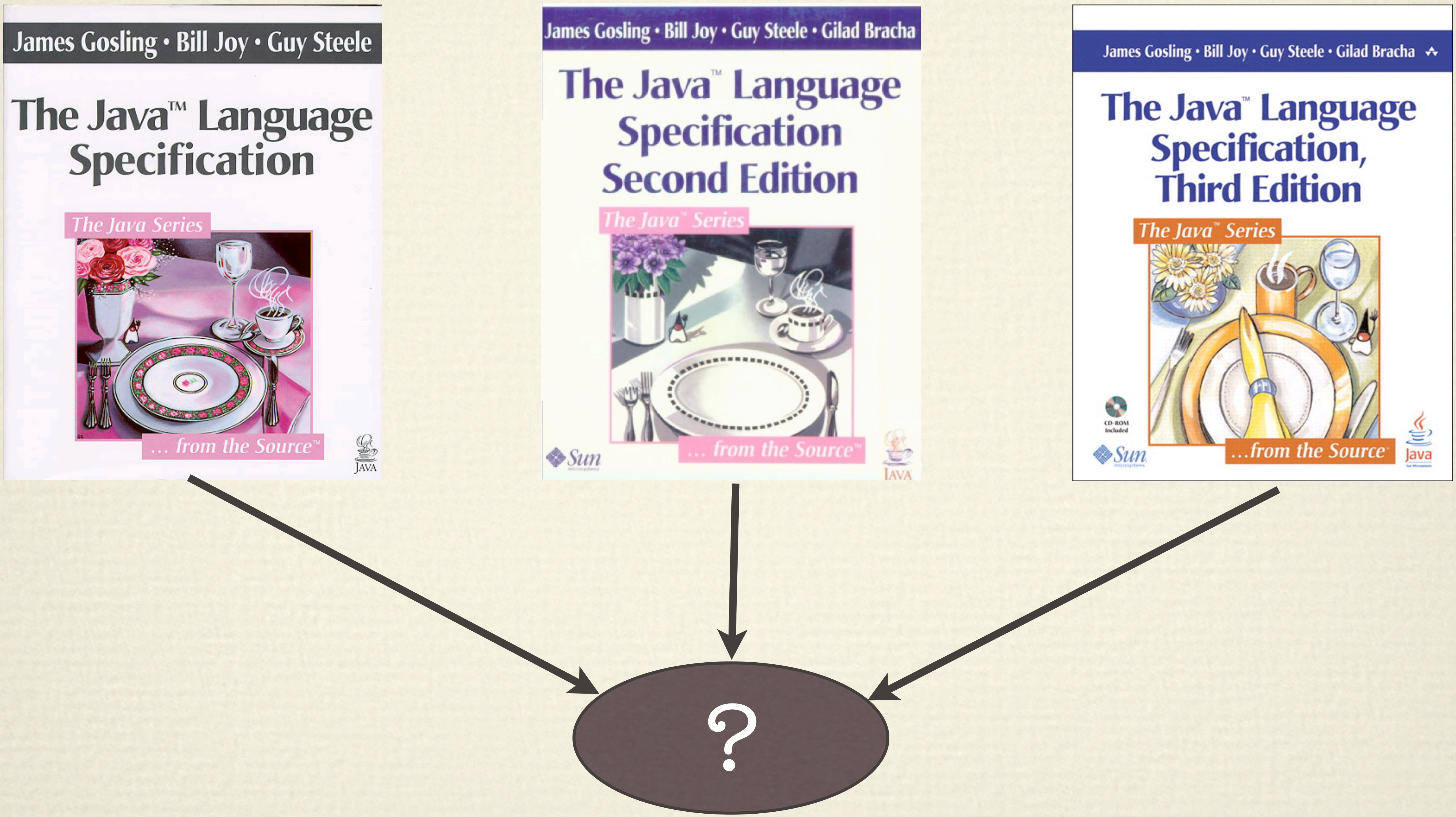
Language documentation



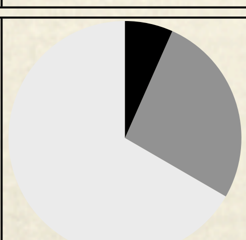
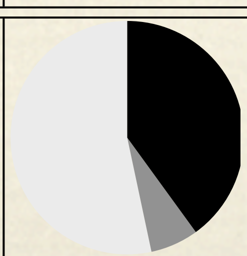
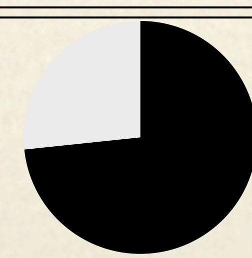
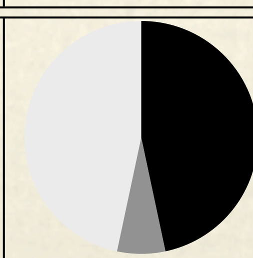
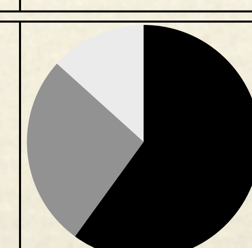
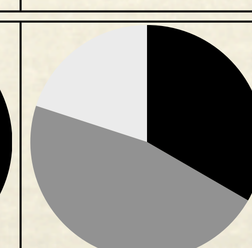
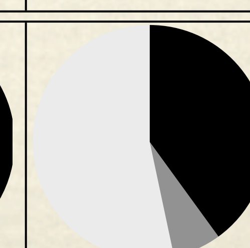
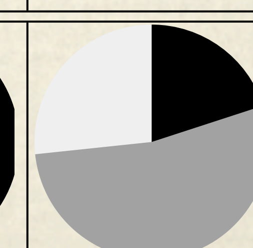
Language documentation



Language documentation



Unified model for language docs

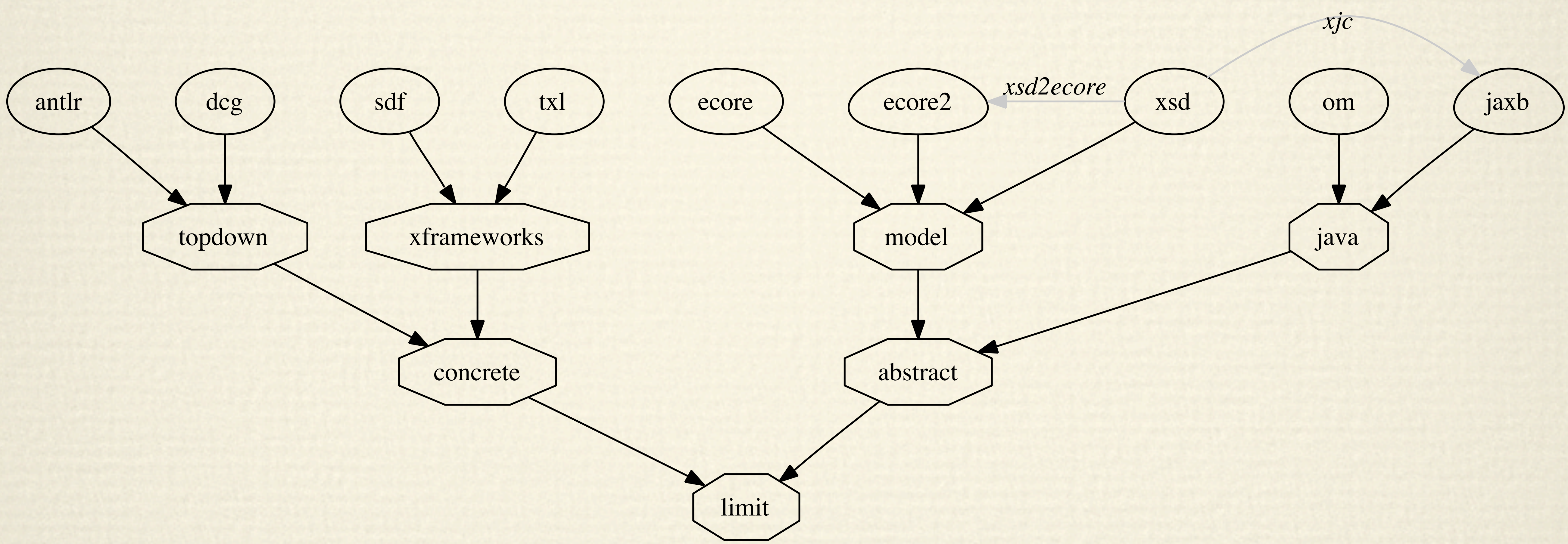
Domain concept	IAL [Bac60]	Jovial [MIL84]	Design Patterns [GHJV95]	Smalltalk [Sha97]	Informix [IBM03]	C# [Sta06]	MOF [MOF06]	XPath [BBC+07]
synopsis	—	~	intent	synopsis	~	~	~	—
description	~	—	motivation	definition	usage	~	—	~
syntax	—[a]	syntax	structure	~	~	~	—	[NN][b]
constraints	—	constraints	applicability	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								

Outline

Recovery,
Convergence
and
Documentation
of Languages

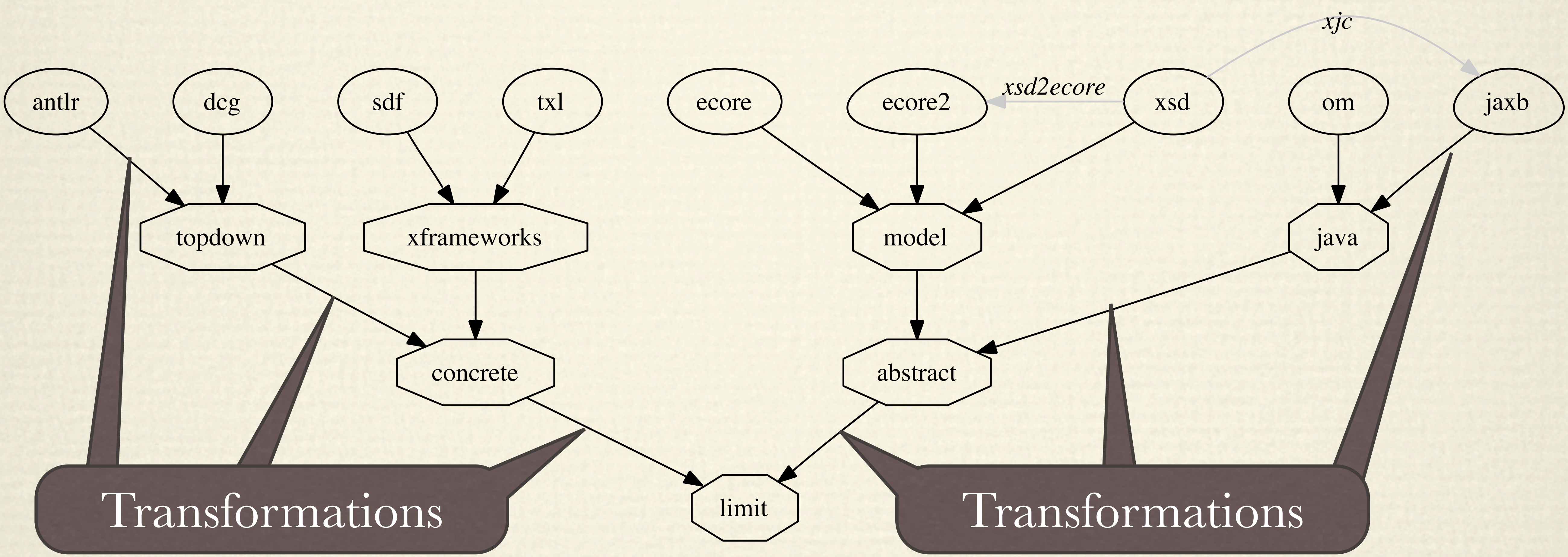
Relationships between languages

Different versions of the same language

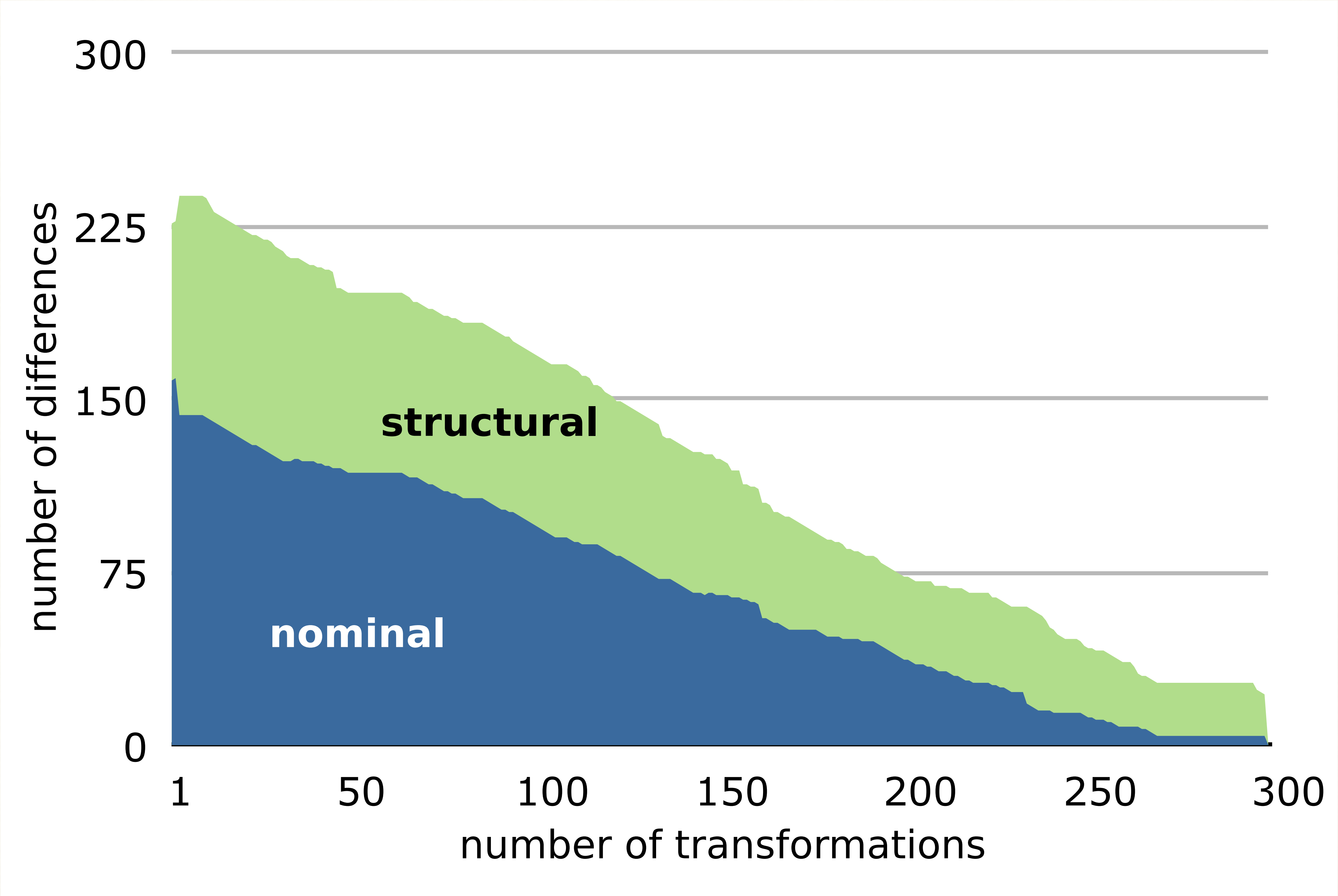


Relationships between languages

Different versions of the same language



Grammar convergence



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

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

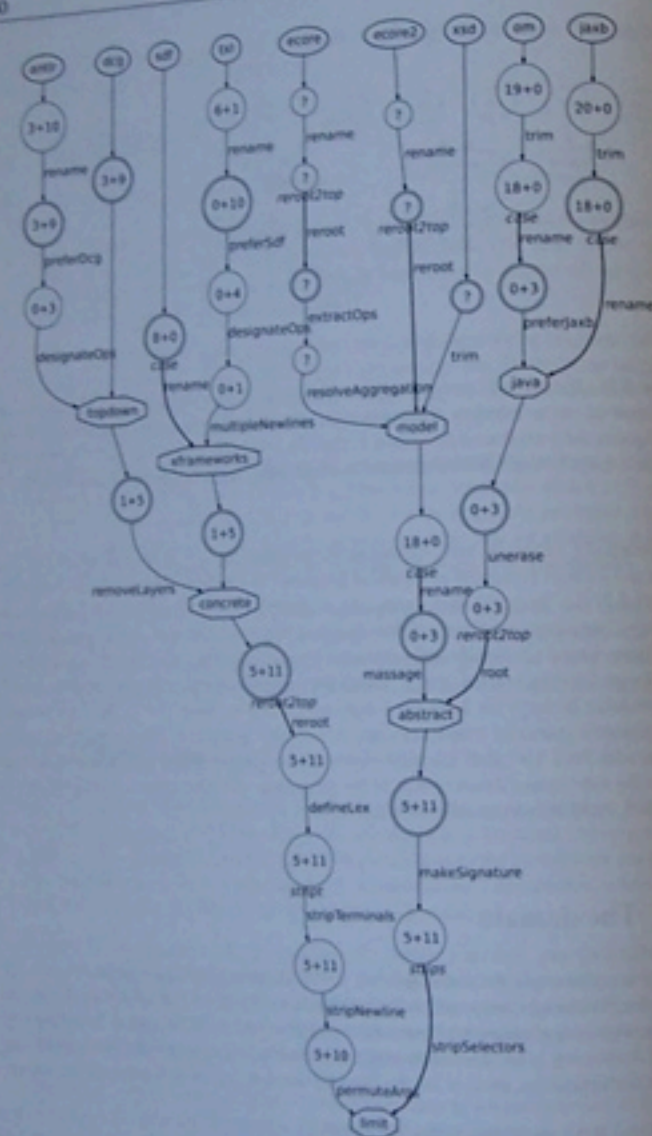


Figure 4.2: The detailed convergence graph for the Factorial Language. The numbers in each bubble are the number of nominal differences plus the number of structural differences. Edges that correspond to automated actions are bolder, with the generator's name in italics. The target *model* has been split in two in order to apply the metrics (otherwise it would be impossible to make a branch choice for synchronisation).

4.3.1 Sources of convergence

Figure 4.1 shows the sketch of a convergence tree for some of the existing FL implementations. The *leaves* of the tree (at the top of the figure) denote different *sources* for FL. We use the term source here to mean "software artefact containing grammar knowledge". Here is short description of the sources for FL:

antlr This is a parser description in the input language language of ANTLR [202]. Semantic actions (in Java) are intertwined with EBNF-like productions.

dcg This is a logic program written in the style of definite clause grammars; see Listing 4.2.

sdf This is a concrete syntax definition in the notation of SDF (Syntax Definition Formalism [86, 239]). It is parser description based on the SGLR implementation for SDF (Scannerless Generalised LR Parsing); see Listing 4.1.

txl This is another concrete syntax definition in the notation of TXL (Turing eXtender Language) transformational framework [39, 42, 43]. Unlike SDF, this framework uses a combination of pattern matching and term rewriting.

ecore This is an Ecore model [197], created manually in Eclipse [59] and represented in XMI; see Listing 4.3.

ecore2 This alternative Ecore model was automatically generated by Eclipse from the XML Schema and extracted from XMI [196].

xsd This is an XML schema [75, 208] for the abstract syntax of FL. In fact, this is the schema that served as the input for generating the object model of the *jaxb* source and the Ecore model of the *ecore2* source.

om This is a hand-crafted object model (Java classes) for the abstract syntax of FL. It is used by a Java-based implementation of an FL interpreter.

jaxb This is also an object model, but it was generated by the XML-data binding technology JAXB [126] from an XML schema for FL.

The sources are part of FL language processors, e.g., interpreters and optimisers.

4.3.2 Targets of convergence

Consider again Figure 4.1. The *inner nodes and the root* denote targets of the convergence process. These are grammars that are derived by transformation with the sole purpose of establishing grammar equality. There are the following targets:

topdown The sources *antlr* and *dcg* both involve top-down parsing. Their correspondence can be established by a few simple refactoring steps.

concrete This target converges all concrete syntax definitions. A noteworthy difference is that *sdf* uses one expression nonterminal, whereas *topdown* uses two "layers".

	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

	VNCS	FST	GDK	GRK	XBGF
[illegible]	resolve	resolve	resolve		define
[illegible]	introduce	introduce	introduce		introduce
[illegible]	extract			extract	extract
[illegible]	include	include	include		addV
[illegible]	add	add		add	
[illegible]					addH
[illegible]	permute			permute	permute
[illegible]	id	id			
[illegible]					designate
[illegible]					deanonymize
[illegible]					appear
[illegible]					concretize
[illegible]					inject
[illegible]					unchain
[illegible]					chain
[illegible]					detour
[illegible]					import
[illegible]					unite
[illegible]		equate			equate
[illegible]	unify	unify	unify		unite*
[illegible]		move			
[illegible]			preserve*		factor
[illegible]	fold	fold	fold	fold	fold
[illegible]	preserve	simplify	preserve		massage
[illegible]	restrict*	restrict*	restrict*		narrow
[illegible]					distribute
[illegible]	unfold	unfold	unfold	unfold	unfold
[illegible]	generalise	generalise	generalise		widen
[illegible]	restrict*	restrict*	restrict*		disappear
[illegible]					abstractize
[illegible]					project
[illegible]	reject	reject	reject		undefine
[illegible]					unlabel
[illegible]	sub	sub			
[illegible]					abridge
[illegible]					anonymize
[illegible]					removeV
[illegible]	reset	reset			
[illegible]	eliminate	eliminate	eliminate	reject	eliminate
[illegible]	exclude	exclude	exclude		removeH
[illegible]					renameL
[illegible]	rename	rename	rename		renameN
[illegible]	substitute	substitute			replace*
[illegible]					renameS
[illegible]					downgrade
[illegible]	delete		delete		replace*
[illegible]					renameT
[illegible]					upgrade
[illegible]	replace	replace		replace	replace
[illegible]					lassoc
[illegible]			preserve*		yaccify
[illegible]					rassoc
[illegible]			preserve*		deyaccify
[illegible]			redefine		redefine
[illegible]	separate	separate	separate		
[illegible]	fail	fail	write		dump
[illegible]					horizontal
[illegible]					vertical
[illegible]					inline

Table 5.8: Systematic comparison of grammar transformation operators provided by different frameworks

syntax (e.g., **substitute**, **reset**), or it is too low-level in the sense that all major application scenarios are covered by more specialised operators (e.g., **add**, **sub**), or it is not currently implementable (e.g., **move**—modules are not fully supported in our infrastructure), or it was simply not needed and perhaps debated so far (e.g., **delete**, **id**, **separate**, also known as FST's **seperate**; see the table for the typo).

There is generally a tension between the number of transformation operators vs. the achievable precision of a transformational program in terms of expressing intentions, and thereby enabling extra sanity checks by the transformation engine. Consider, for example, the line "add a production to the grammar". This low-level idiom may be used to **include** another production into an existing definition, or to add one or more productions in an effort to **resolve** a missing definition, or to **introduce** a definition for a so-far fresh nonterminal. In GRK, all these idioms are modelled by **add**, and hence no intentions are documented, and no extra checks can be performed automatically. In the case of XBGF, we have indeed tried to separate idioms aggressively. This approach also helps us with predicting the formal properties of each application of transformation operators (i.e., semantics-preserving, -increasing, -decreasing, -revising), and chains thereof.

5.9.3 Grammar engineering

Let us also discuss some additional related work on grammar engineering [141] in a broader sense. We begin with metrics which are used by various recovery approaches and other work on grammar engineering. We want to highlight [6, 58, 135, 150, 174]. Our work leverages simple grammar metrics (numbers of bottom and top nonterminals) and grammar-comparison metrics (numbers of nominal and structural differences) for providing guidance in a grammar convergence context.

An interesting blend of recovery and convergence (or consistency checking) is described in [21] where *precedence rules* are recovered from *multiple* grammars and checked for consistency. At this point, grammar convergence (in our sense) does not cover such sophisticated convergence issues. In fact, our approach is, as yet, oblivious to technology-specific representations of priority rules (as used in, say YACC or SDF). We could potentially detect priority layers in plain grammars, though.

An alternative to grammar recovery is the use of a flexible parsing scheme based on advanced error handling [12, 13, 142], subject to a baseline grammar. Because of flexible parsing, the grammar could also be used to parse a dialect; no precise grammar is needed. Also, code with syntax errors can be handled, which is important in some application areas such as reverse or re-engineering of legacy code.

There are approaches to connect the technical spaces of grammarware and modelware in a manner that can be viewed as a form of grammar convergence. That is, the parser may be obtained from the (meta)model based on appropriate metadata and mapping rules, using a generative approach [134, 192]. We also use the term model-driven parser development for these approaches. The point of grammar convergence is that it provides a very flexible means to represent relationships between grammar-like artefacts from different technical spaces—without enforcing a particular scheme of designing grammar-based artefacts.

Outline

Recovery,
Convergence
and
Documentation
of Languages

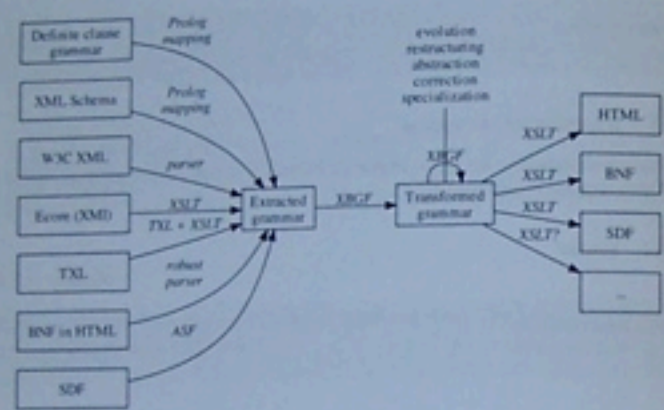


Figure 6.2: The life cycle of a language grammar in the transformational environment: from a grammar knowledge possessing software artefact on the left to the usable working grammar on the right.

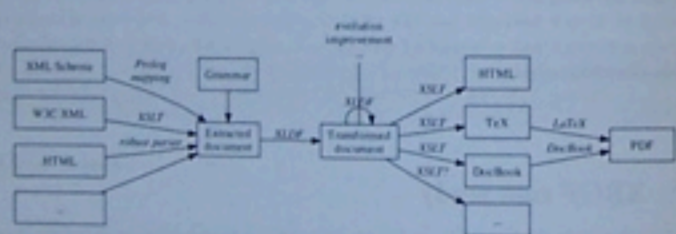


Figure 6.3: The life cycle of a language document in the prototype language document infrastructure: from the structure source on the left to the extracted document gradually transformed to its ultimate form and finally pretty-printed for presentation on the right.

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.

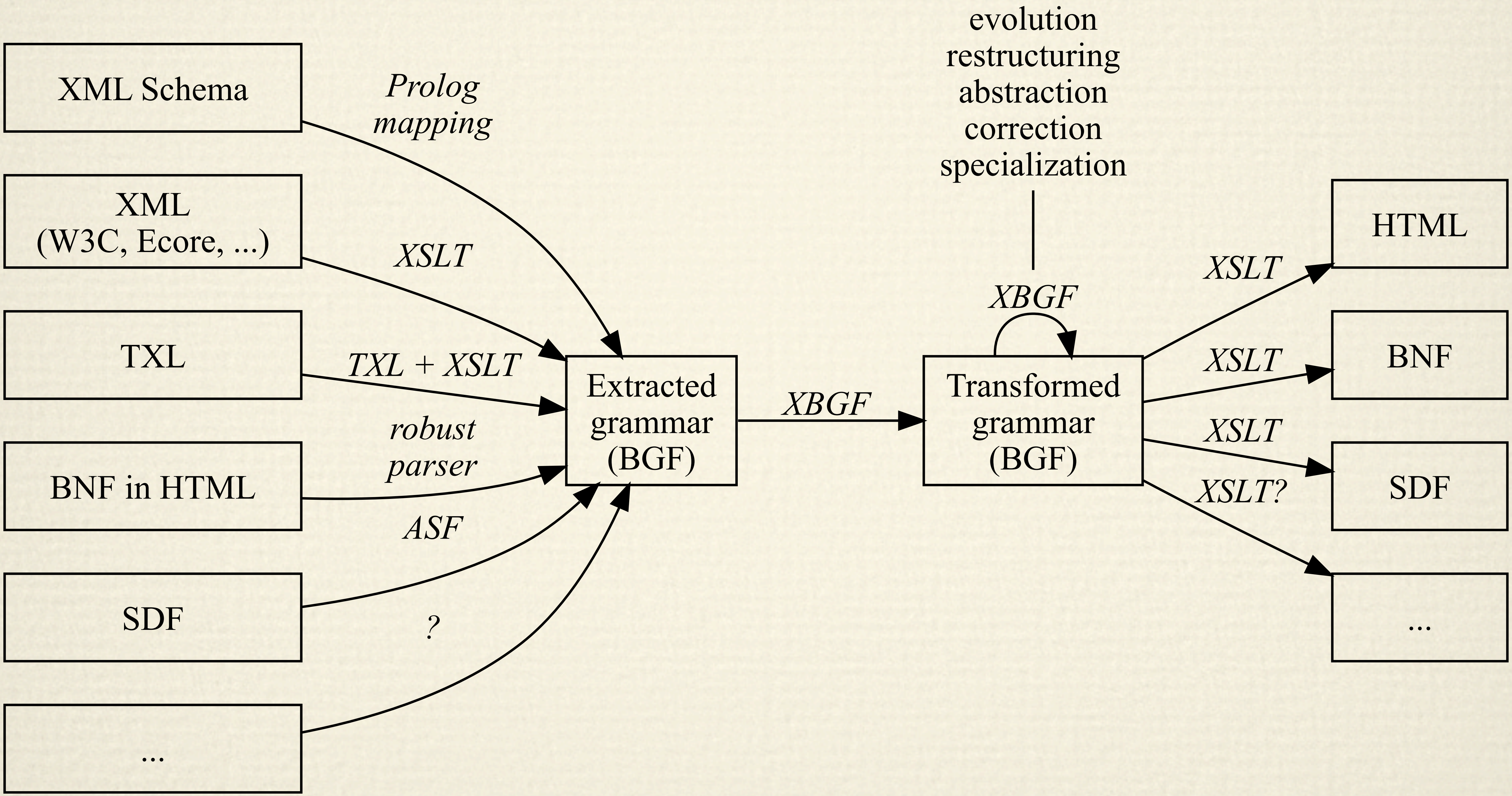
In XSD it is possible to annotate any construct with a piece of text, and that feature was extensively utilised during schema development phase to provide comments for XBGF operators. With `xsd:annotation` and `xsd:documentation` tags we basically inserted typical language documentation information right 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.

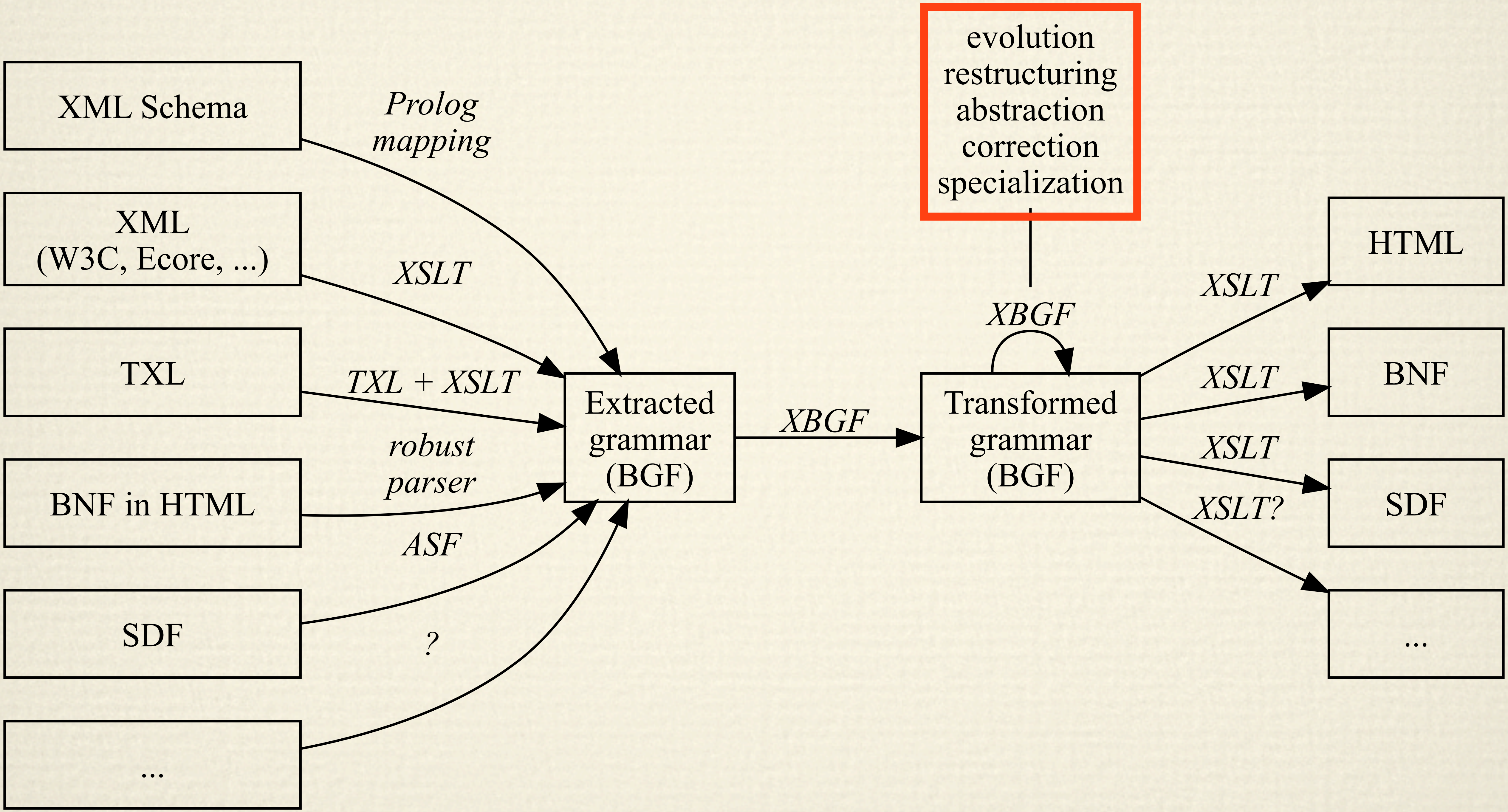
6.7.2 Transformation

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.

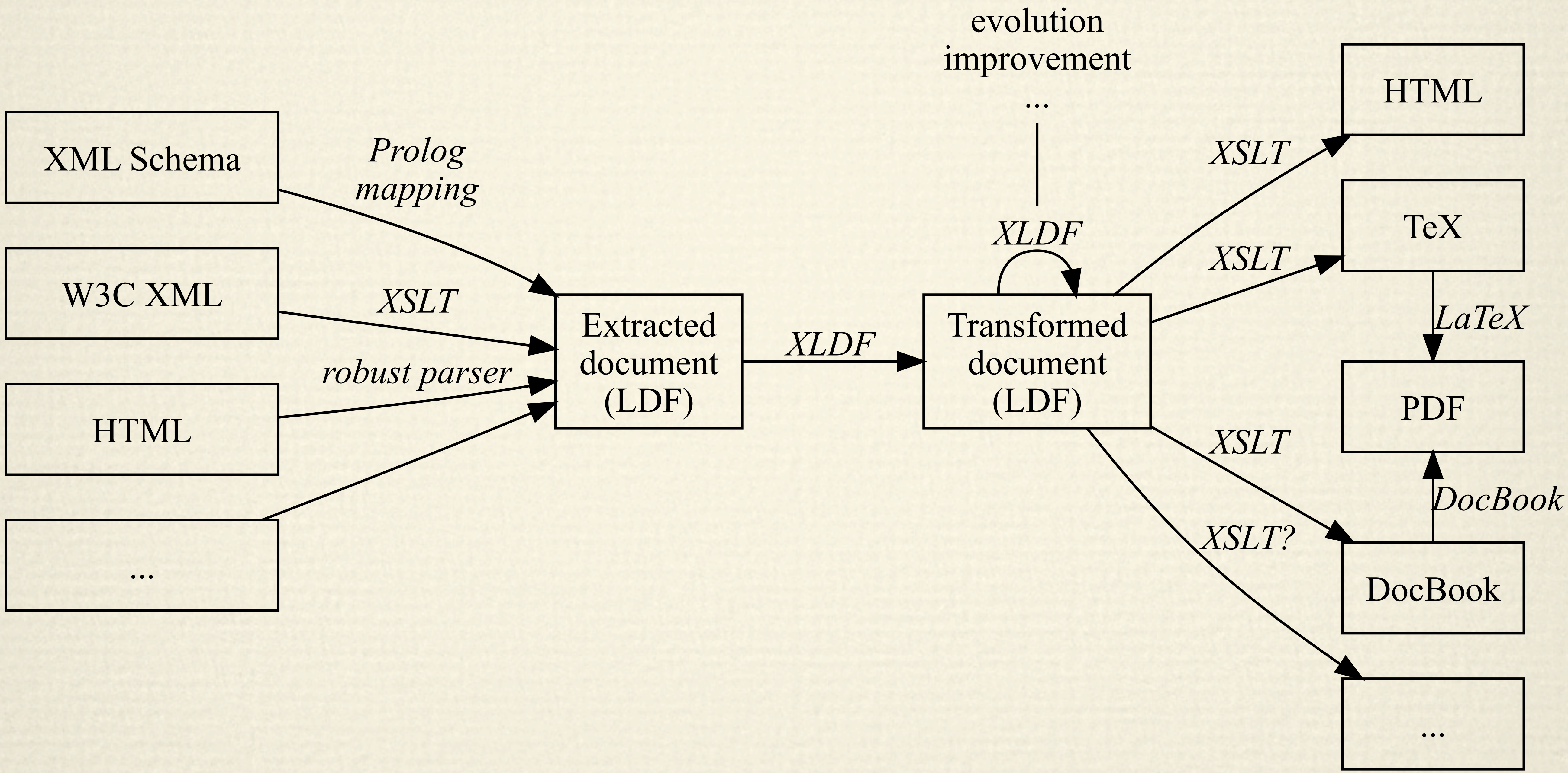
Grammar recovery



Grammar ~~recovery~~ engineering



Language document recovery



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

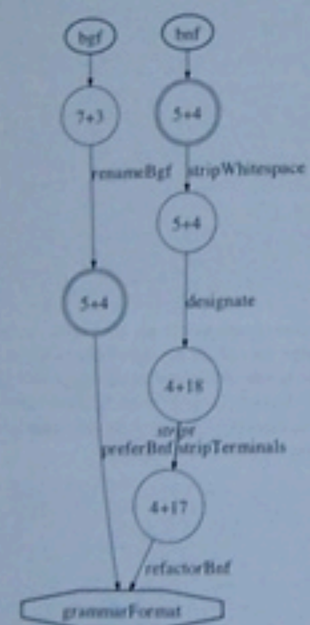


Figure 7.1: Full convergence diagram for BNF and BGF. The top nodes are sources, the bottom node is the target, the arc labels are separate XBGF scripts, the nodes contain numbers of name mismatches and structural mismatches between each step and the synch point (marked as a double circle).

Chapter 8

Conclusion

An ideal world is left as an exercise to the reader.

Paul Graham, 1993 [80]

8.1 Summary

The conceptual contributions of this thesis are listed by the fields of research.

Grammar recovery. A successful endeavour has been made to generalise the steps needed for recovering grammars from real software artefacts with embedded grammar knowledge.

Grammar extraction. The possibility has been shown to automate grammar extraction and to make those extractors so advanced that they operate on a set of rules specified by a language engineer beforehand. Based on such rules, the extractors detect and repair presentation inconsistencies in typical existing language artefacts such as standards that many assume are flawless.

Grammar convergence. We presented the methodology that allows a language engineer to take two or more grammars that are assumed to be related (equal, one covered by another, etc) and by applying a combination of described methods and tools to surface the relationships among them. Such relationships are formally represented by sequences of grammar transformation steps.

Grammar transformation. After careful examination of the existing achievements in this field, an operator suite called XBGF was developed. To the best of our knowledge and experience of working with different transformational frameworks, XBGF surpasses previously existing technology in automation, granularity, maintainability. The proposed set of operators fits the domain of grammar transformation closely, providing separate specialised commands for common use patterns.

Conclusion

- ★ Language recovery steps generalised
- ★ Language convergence methodology proposed
- ★ Language documents analysed
- ★ Transformation languages developed
- ★ All tools and infrastructures prototyped
- ★ Several grammars and relationships delivered


The End