Chapter 8

Conclusion

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.
Language documentation. We reverse engineered a large library of language specifications and designed a general document schema that describes a format of a typical language standard or manual. We presented a description of an infrastructure that needs to evolve around this schema to be integrated into the language evolution process. The transformational language and all the tools have been successfully prototyped, although the full scale application of this methodology to industrial standards is beyond the scope of the thesis and was left for future work.

8.2 Research contributions

Scientific contributions of the thesis are summarised in the following list:

Deeper insight into grammar recovery field. The case study we have done for C# has shown that the problems encountered in Cobol and C# manuals are quite similar since similar methods turned out to be applicable to such different cases. Published as [257].

Grammar convergence methodology. We presented a lightweight verification method for transforming grammars until they become identical. We proposed to treat the resulting transformation chains as relationships among the source grammars. A case study converging six related Java grammars was also performed successfully. Published as [166, 167].

Principles of automated grammar extraction. The rule-based structure of the extractor that we used in the convergence case study, proved to be a viable grammar recovery infrastructure. By generalising every problem and lifting it to the level of a pattern applicable to a class of similar problems, we automated a significant part of the error recovery process. The incremental development process of such an extractor is repeatable for other notations and sources.

Detailed analysis of existing language documents. More than 40 language specifications and reference manuals were analysed and compared for their notations and structure. Despite the diversity of notation for syntactic definitions, most EBNF dialects have the same basic principles and the same expressivity, so automated migration from one to another does not pose any technical challenge. However, the ways language documents are organised and structured, are sometimes fundamentally different.

Unified language documentation data model. Based on our analysis, we proposed a general schema that is useful for many language documents. Specific benefits of its application are straightforward grammar extraction, solid separation of concerns, advanced opportunities for testing and document querying.

Language documentation infrastructure prototype. We formulated the principles of an integrated approach to language documentation design, evolution and maintenance. We also proposed a supporting infrastructure to implement it. Published as [143, 258].
Operator suite for grammar transformation. After using grammar transformation in several convergence scenarios, we designed an advanced and detailed grammar transformation operator suite. Having a separate operator for each specific case fits the domain closer by allowing better applicability conditioning and property proving. Having a wide choice of well-advocated operators allows grammar engineers to limit themselves to refactoring steps whenever possible and not backslide to local replacements as is common in many frameworks. As a result, sequences of grammar transformation steps are easier to comprehend.

8.3 Engineering deliverables

We did not limit ourselves to exploring ideas about grammar recovery, transformation and convergence. All aspects were worked out up to the point of having a practical prototype, an reusable tool, a language definition or a useful documentation. This approach yielded several primarily engineering deliverables that are useful and reusable for a wide range of tasks.

The following list describes all types of engineering deliverables, and each item has a subsection associated with it.

Grammars. Designed or recovered, grammars are useful to parse code or for further investigations into their properties.

Languages. The domain-specific languages form the backbone of our proposed framework.

Documents. Each of the language documents both serves as a manual and exemplifies language documentation facilities.

Grammar relationships. Grammar convergence case studies directly result in established relationships among existing grammars.

Tools. We have developed mappers between existing notations and our DSLs.

8.3.1 Grammars

The thesis delivered the following grammars that are electronically accessible through the internet at the SLPS Grammar Zoo (http://slps.sf.net/zoo).

C#. We applied grammar recovery skills to the international standard of C# by ECMA [225] and ISO [114]. The recovering transformations were performed with the Grammar Deployment Kit [149], a previously available framework. The resulting grammar is available in Syntax Definition Formalism [86] as well as in a browsable form generated from it.

Java: JDK 1.0 “Oak”, J2SE 1.2 “Playground”, J2SE 5.0 “Tiger”. The case study that we completed with our own infrastructure delivered three Java grammars recovered from three editions of the Java Language Specification [77, 78, 79]. The
extractor was designed to automatically correct presentation inconsistencies found in the language documents.

**Factorial Language.** Nine small grammars in different forms were developed as examples when prototyping the method of grammar convergence. The author of this thesis designed four FL grammars in ANTLR [202], TXL [39], ASF+SDF [22] and Ecore [59]. Two other FL grammars were derived automatically by existing model transformation or grammar transformation tools.

**C: ISO/IEC 9899:1999(E), 9899:TC2, 9899:TC3.** Since grammar notations used in JLS and in [117, 118, 119] are very similar, it was possible to re-use the principles of the Java grammar extractor for two different ISO standards containing grammars of C.

**C++: ISO/IEC 14882:1998(E), SC22/WG21 N2723.** Grammar notations used in JLS and ISO C standards are similar also to [111] and [120]. We re-used exactly the same tool that was employed for extracting ISO C grammars, for the extraction of ISO C++ grammars. Both C and C++ grammars will be used by SLPS contributors in future work on language convergence, and are already being used by others as a stable extraction source1.

These grammars differ at a level of possible practical use. Technically speaking, the C# grammar is a Level 3 grammar, i.e., it is complete enough for a parser to be generated from it and it was tested on a small codebase. The three Java grammars are Level 2 grammars, i.e., they only define language syntax and lack the lexical parts. All nine FL grammars are at Level 5, i.e., they have been directly derived from compiler sources, parser definitions, schemata and similar artefacts.

### 8.3.2 Languages

During this research, we designed, developed and implemented six new languages specific for grammarware domain. They are used for storing grammars, documents, transformations, configurations and parse trees, and are generally useful and applicable to many adjacent topics of grammar engineering, re-engineering and reverse engineering. We chose to express these DSLs in XML Schema.

**BGF, BNF-like Grammar Format.** This language emerged as a compromise for storing grammar knowledge that is traditionally expressed in BNF, EBNF and dialects thereof. BGF provides the usual functionality of specifying terminal and nonterminal symbols and combining them sequentially or alternatively, plus extra features like selectors, i.e., named sub-expressions for which we found use in many extraction and transformation activities.

1E.g., Lukas Renggli (University of Bern): [http://twitter.com/renggli/status/17401122412](http://twitter.com/renggli/status/17401122412).
8.3 Engineering deliverables

**XBGF, BGF Transformations.** This is the basic transformation language used in grammar convergence, we utilise it to express the sequences of calls to grammar transformation operators. Whenever an operator requires a grammar production or a grammar expression as a parameter, BGF is re-used.

**BTF, BGF Tree Format.** We used this language for coupled transformations, when we needed to specify not only grammars as such, but also their instances, i.e., parse trees. The trees contain the original productions that were used for their generation.

**LDF, Language Document Format.** After analysing >4 language documents of different nature, we inferred a general metamodel to cover all observed variations. This metamodel is expressed and exploited in the form of LDF. The language defines domain concepts like code samples of syntax description sections, as well as the rules for combining them.

**XLDF, LDF Transformations.** We extended XBGF to work on language documents: each XLDF command represents a language evolution step (such as adding a new language construct) or a language documentation evolution step (such as rearranging sections).

**LCF, LCI Configuration Format.** Language convergence infrastructure needed a separate DSL to represent the starting configuration and the desired outcome of a convergence scenario. LCF concepts include the notions of an extractor, a generator, a branch of convergence, a convergence phase and others.

### 8.3.3 Language documents

The thesis delivered the following language documents:

**XBGF.** This manual served as a showcase for our language documentation framework for language documentation life cycle, i.e., storage, evolution, transformation, testing and convergence. Its core was extracted in a completely automated manner from the XML Schema definition of the language to form the starting LDF document. Then it was transformed by a chain of XLDF commands that completed the text with side remarks and inserted pretty-printed test cases. The resulting LDF document was used directly to generate the grammar transformation chapter of the thesis (see chapter 7 that is completely automatically generated using our framework). The online hyperlinked version of the XBGF manual was generated from the same LDF document as well.

**XLDF.** The XML Schema definition of XLDF was used as an extraction source for the XLDF manual, and several XLDF scripts were used to refactor the resulting LDF document to its final form. The corresponding thesis section was generated automatically from this final LDF document using the general LDF to EPSTX pretty-printer.

**LDF.** The self-definition of LDF in LDF was generated from the XML Schema definition of LDF, and several XLDF scripts were used to refactor the resulting LDF document...
to its final form. The corresponding thesis section was generated automatically from this final LDF document using the general LDF to \texttt{\LaTeX} pretty-printer.

**LCF.** The LCF manual focuses on the domain of configuration of the overall infrastructure and mainly on the semantics of the language constructs. For instance, if a tool is defined with a particular tag, the manual documents when it will be called and with which parameters. The LCF manual is extracted from a corresponding XML Schema definition, beautified by transformations and pretty-printed to \texttt{\LaTeX}. It contains considerably less code fragments than the other manuals and more information about informally described semantics.

### 8.3.4 Grammar relationships

The following grammar relationships were recovered by performing the corresponding case studies:

**Factorial Language.** Nine different grammars of the same language were converged, extracted from ANTLR, DCG, SDF, TXL, Ecore, XSD and Java sources. The transformation steps representing the relationships uncovered all the peculiarities of each of the source formats.

**Java Language Specification.** Six different grammars of the same language were converged, extracted from three versions of Java Language Specification by pairs of a grammar intended for reading and a grammar intended for implementation. The recovered relationships were compared with the explicit and implicit claims such as backward compatibility.

**BNF-like Grammar Format.** The abstract definition in the schema of BGF was converged with its concrete representation used by a pretty-printer. This case study was so small that we could a priori predict exactly the differences that we encountered during convergence. In such a situation, our method was used as complementary framework testing.

### 8.3.5 Tools

We do not list general tools like the transformation engine here, but rather assume that the claims from other sections should be interpreted in a wide sense: thus, “we have a language XBGF” means that there is a document schema for it, as well as the appropriate language processor that can run XBGF commands, as well as libraries for parsing XBGF files. However, we do distinguish between extractors and pretty-printers. **Extractors** establish mappings between certain classes of grammar artefacts and the BGF-driven infrastructure, i.e., they transform other formats to BGF, BTF or LDF. **Pretty-printers** were developed to map the grammars, documents and scripts stored in XML in a disciplined fashion, to other formats, i.e., to generate alternative textual representations of them (mostly plain text, hypertext or \texttt{\LaTeX}). Only those tools attributed to the author of the thesis are listed. All tools are easily executable via wrapper scripts with one or
two necessary parameters, although internally they can consist of multiple calls to various utilities from different frameworks.

Extractors:

**HTML to BGF.** This advanced extractor had to work on a manually and loosely hypertext source. It comprised a set of generalised rules in a pattern form that it tried to apply for automated recovery.

**ANTLR to BGF.** In order to be able to extract grammars from ANTLR parser definitions, we re-used the standard ANTLR grammar for ANTLR grammars by attaching appropriate semantic actions to it.

**SDF to BGF.** We encoded the necessary traversal functions for crawling the parse trees of SDF grammars and producing BGF and reused the SDF module and the XML module from the standard package of the Meta-Environment.

**TXL to BGF.** We re-used the TXL grammar for TXL grammars; the mapping between TXL XML and our XML (i.e., BGF) was straightforwardly encoded in XSLT.

**Ecore to BGF.** Since Ecore models are by default serialised as XMI, we only needed to express the mapping between Ecore and BGF, which was done in XSLT.

**AsFix to BTF.** We encoded the necessary traversal functions for crawling the parse trees of AsFix parse trees and producing BTF and reused the AsFix module and the XML module from the standard package of the Meta-Environment.

**XML Schema to LDF.** The XSD has proven to be a suitable format for prototyping language documentation extraction, since it is capable of expressing a grammar-like structure, as well as annotating functionality. This extractor intertwines the XSD annotations with the extracted grammar productions.

**LDF to BGF.** Since we assume that any LDF document does contain grammar knowledge explicitly, i.e., in BGF, we use a special extractor to take out the BGF bits and compose a grammar from them.

Pretty-printers:

**BGF to text.** The main pretty-printer is the one that takes any BGF file and generates a purely textual serialisation of it suitable to be read by a human. Strictly speaking, this pretty-printer re-invents the concrete syntax for BGF.

**BGF to \LaTeX.** In order to use BGF more extensively and more efficiently for examples in this thesis and in all related articles, we developed a separate pretty-printer that wraps the textual syntax of BGF in a proper \LaTeX environment.

**LDF to \LaTeX.** Given a proper LDF, this pretty-printer generates a compilable standalone \LaTeX document. All textual sections are named properly, and all BGF productions included in the LDF are pretty-printed by re-using the BGF to text mapper.
LDF to PDF. We have two competing implementations that map LDF to PDF: one generates a \texttt{LATEX} file and runs \texttt{pdflatex}, while the other one generates X\texttt{HTML}, transforms it to X\texttt{SLT} FO by re-using an openly distributed X\texttt{SLT} sheet, and runs \texttt{fop}. Apparently, the \texttt{LATEX}-based path produces more appealing PDFs.

LDF to HTML. It is easy to represent an LDF document in a hypertext form. Due to the nature of X\texttt{SLT}, the transformation always produces a well-formed X\texttt{HTML} document (claim that is hard to make for LDF to \texttt{LATEX} mapper). The X\texttt{BGF} manual intended for online distribution was created using this pretty-printer.

X\texttt{BGF} to text. We developed a BNF-resembling concrete syntax that we use for representing the transformation scripts in a textual form. The productions and expressions of \texttt{BGF} occur as parameters to \texttt{XSGF} commands.

X\texttt{BGF} to \texttt{LATEX}. The appropriately parametrised \texttt{LATEX} package \texttt{listings} allows to present X\texttt{BGF} scripts in an appealing presentation format. The textual component is pretty-printed by re-using the X\texttt{BGF} to text mapping.

All pretty-printers were implemented in X\texttt{SLT}: either in X\texttt{SLT} or in X\texttt{SLT} FO.

8.4 Future work

Automated grammar recovery. For this project we have designed, developed, implemented and presented a heuristic rules-based grammar extractor. It was used for the Java Language Specification case study, where it successfully extracted six grammars from differently formatted HTML documents. However, it must be possible to generalise the rules further and make them universally applicable to a broad class of hypertext grammar descriptions. The development of robust extractors capable of solving extraction problems in a generic way is one of the important directions of future research and future contributions to the Software Language Processing Suite.

Dialect-parametric parsing of EBNF. During the research we have seen dozens of different dialects of BNF and EBNF and completed their study with concise conclusions about how the variations relate to the ISO EBNF standard. In the future, we plan to design a DSL for defining EBNF dialects and implement a generic parser based on it. By combining disciplined dialect definition with automated grammar extraction we will reach the milestone of treating all (E)BNF-based language specifications the same for the purposes of language recovery.

Suggestive convergence metrics. At its current state the method of grammar convergence relies on a set of guidelines that the language engineer follows, and on a set of comparison-based metrics that measure the convergence progress. The grammar comparator at this stage displays only apparent differences between the two grammars under consideration. One can suggest experimenting with aggressive forms of normalisation, search automation for specific constructs that are known to
be refactoring material, name matching automation algorithms. The boundaries for effectiveness and automation in this area have not been reached yet.

**Inference of converging transformations.** Our method of grammar convergence is based on automation of small atomic grammar transformation operators. All operator calls and parameters are performed by a human expert with the aid of metrics. It is also known that the opposite extreme is possible, where all transformation steps are derived automatically give the source and the target grammar. However, the inference algorithms used in such a scenario use a limited set of possible transformations (cf. the Levenshtein distance: deletion, insertion, substitution). The question of inferring transformation chains for the purpose of grammar convergence based on an advanced operator suite such as XBGF is an interesting research topic.

**Generality of XBGF.** At this point we cannot make a useful generality statement about the XBGF grammar transformation operator suite. Proving that all possible grammar refactorings are expressible in XBGF, or that any transformation step except error correction is expressible without substitution, is an interesting future research topic.

**Formal proofs for XBGF operator properties.** The properties of grammar transformation operators such as preservation of semantics were explained in an “intuitive” way. For operators dealing with concrete syntax, we also used arguments based on string languages. However, it can also be useful to have formal proofs for them in term algebra. This is a highly nontrivial task: the underlying grammar sources are heterogeneous and are based on different semantics—the meaning of a selector in one formalism can overlap with the meaning of a nonterminal symbol in another. In order to approach a solution to this problem, we intend to enhance the XBGF with annotations and develop bidirectional mapping tools based on our existing grammar extractors. With this, one will not only be able to derive a grammar from a data model or a parser definition, but to consistently transform a data model or a parser definition to a grammar and back without any loss of information. When such mapping is established, we will have the information about the necessary amount of extra annotations, and then we plan to utilise automated theorem proving techniques here.

**Language documentation.** On the pages of this thesis we have proposed an infrastructure for language documentation life cycle. Several case studies were performed to support the prototype. All of them used a “clean” source format for extraction (XML Schema) and the beautification transformations were dictated by trivial differences between LDF and XSD. In the future we will take an existing specification, extract the LDF document from it, perform improving transformations until the acceptable level of usability is reached, examine it with various automated or semi-automated analysis techniques, resolve the inconsistencies found in the process, and regenerate the specification and satellite artefacts such as a test set.

**Language document transformation.** At this stage XLDF, the language for language document transformation, is a minimal one needed to complete the case studies we
have scheduled for this project. Just like with XBGF and even more so, this set of commands will need streamlining in the future, with constructive claims about its completeness and expressive power and verification thereof. There are also some open research questions about the criteria for language document transformations: e.g., will it be truly viable to have one suite for language evolution and language adaptation?

**Reversibility of grammar transformations.** Currently most of XBGF operators are implemented and presented in pairs: **add** and **remove**, **upgrade** and **downgrade**, **project** and **inject**, etc—in each such pair one of the operators is the inverse of the other. In future work, reversibility will be addressed and examined more thoroughly, with guaranteed properties and development of the prototype higher-order tools to generate transformations that reverse the effect of a transformational script given as input.

**EBNF extension for language documentation.** Given the analysis of EBNF dialects that we performed and the general language documentation engineering effort, we can see that a practically aimed, technology independent, modern and powerful formalism for describing everything needed for a modern language definition is still needed. Backus normal form, Wirth syntax notation and their variations were mostly used for expressing grammar knowledge during the last five decades, together with attempts to improve it, successful [112, 252], unsuccessful [31, 123, 250] and domain-specifically successful [175, 201]. We did admit that a standardised grammar definition formalism is needed, and even listed the factors that should affect its design, such as tool support, freedom from a fixed parsing technology, modularity, unambiguity. A possible future work direction is to define such a language and perform case studies with it. The extension would perhaps go even beyond “syntactic sugar”, they can bear something like static semantics to express more structured details than possible with EBNF now.

**Synchronisation points.** We introduced the notion of a synchronisation point for targets with two branches. It should be possible to define a synchronisation point for all targets, possibly by letting the comparator decide which grammar is closer to the one under measurement. By doing so, an important issue of choosing what grammars are to converge first, will be solved.

**Coupled transformation.** Convergence of parse trees is a topic that was not fully developed in this project. We did mention on several occasions that it is important to extend grammar transformation operators’ behaviour to the level of instances. For example, if the research on coupled transformation is completed, it will allow us to converge test sets together with grammars.

**Bridging grammarware and modelware.** Currently there is no consensus about what should count as programming and what as modelling. Together with classic programming languages such as C and classic modelling ones like UML there exist methodologies like EMFText that have both characteristics. Similarly, almost the same problems often arise before grammar experts and metamodel experts. There
are domains in which one or the other of these approaches prevails, and many where their comparison was not performed or turned out indecisive. For convergence, we will need to examine similar methods that compare, converge and calculate the difference between two models. Also a deeper analysis and comparison of grammarware and modelware is needed in order to draw conclusions about overall compatibility, applicability and the advantages of our methodology when applied to this adjacent area.