MediaWiki Grammar Recovery

Vadim Zaytsev, vadim@grammarware.net
SWAT, CWI, NL

July 23, 2011

1 Introduction

Wiki is the simplest online database that could possibly work [41]. It usually takes a form of a website or a webpage where the presentation is predefined to some extent, but the content can be edited by a subset of users. The editing ideally does not require any additional software nor extra knowledge, takes place in a browser and utilises a simple notation for markup. Currently there are more than a hundred of such notations, varying slightly in concrete syntax but mostly providing the same set of features for emphasizing fragments of text, making tables, inserting images, etc [10]. The most popular notation of all is the one of MediaWiki engine, it is used on Wikipedia, Wikia and numerous Wikimedia Foundation projects.

In order to facilitate development of new wikiware and to simplify maintenance of existing wikiware, one can rely on methods and tools from software language engineering. It is a field that emerged in recent years, generalising theoretical and practical aspects of programming languages, markup languages, modelling languages, data definition languages, transformation languages, query languages, application programming interfaces, software libraries, etc [15, 23, 25, 70] and believed to be the successor for the object-oriented paradigm [14]. The main instrument of software language engineering is on disciplined creation of new domain specific languages with emphasis on extensive automation. Practice shows that automated software maintenance, analysis, migration and renovation deliver considerable benefits in terms of costs and human effort compared to alternatives (manual changes, legacy rebuild, etc), especially on large scale [11, 61, 65]. However, automated methods do require special foundation for their successful usage.

Wikiware (wiki engines, parsers, bots, etc) is a specific case of grammarware (parsers, compilers, browsers, pretty-printers, analysis and manipulation tools, etc) [25, 75]. The most straightforward definition of grammarware can be of software which input and/or output must belong to a certain language (i.e., can be specified implicitly or explicitly by a formal grammar). An operational grammar is needed to parse the code, to get it from a textual form that the programmers created into a specialised generational and transformational infrastructure that usually utilises a tree-like internal format. In spite

of the fact that the formal grammar theory is quite an established area since 1956 [9], the grammars of mainstream programming languages are rarely freely obtainable, they are complex artefacts that are seen as valuable IT assets, require considerable effort and expertise to compose and therefore are not always readily disclosed to public by those who develop, maintain and reverse engineer them. A syntactic grammar is basically a mere formal description of what can and what cannot be considered valid in a language. The most obvious sources for this kind of information are: language documentation, grammarware source code, international standards, protocol definitions, etc.

However, documentation and specifications are neither ever complete nor error-free [79]. To obtain correct grammars and ensure their quality level, special techniques are needed: grammar adaptation [32], grammar recovery [36], grammar engineering [25], grammar derivation [27], grammar reverse engineering, grammar re-engineering, grammar archaeology [34], grammar extraction [75, §5.4], grammar convergence [37], grammar relationship recovery [39], grammar testing [33], grammar inference [64], grammar correction [75, §5.7], programmable grammar transformation [74], and so on. The current document is mainly a demonstration of application of such techniques to the MediaWiki BNF grammar that was published as [47, 46, 51, 52, 49, 50, 48, 44].

1.1 Objectives

The project reported in this document aims at extraction and initial recovery of the MediaWiki grammar. However, the extracted grammar is not the final goal, but rather a stepping stone to enable the following activities:

Parse wiki pages. The current state of Wikipedia is based on a PHP rewriting system that transforms wiki layout directly into HTML [53]. However, it can not always be utilised in other external wikiware: for example, future plans of Wikimedia Foundation include having an in-browser editor with a WYSIWYG front-end in JavaScript [69]. Having an operational grammar means anyone can parse wiki pages more freely with their own technology of choice, either directly or by deriving tolerant grammars from the baseline grammar [27].

Aid wiki migration. The ability to easily parse and transform wiki pages can deliver considerable benefits when migrating wiki content from one platform to another [78].

Validate existing wiki pages. The current state of MediaWiki parser [53] allows users to submit wiki pages that are essentially incorrect: they may combine wiki notation with bare HTML, contain unbalanced markup, refer to nonexistent templates. This positively affects the user-friendliness of the wiki, but makes some wiki pages possibly problematic. Such pages can be identified and repaired with static code analysis techniques [7].

Test existing wiki parsers. There is considerable prior research in the field of grammar-based testing, both stochastic [43, 60] and combinatorial [19,

33, 42, 56, 72], with important recent advances in formulating coverage criteria and achieving automation [16, 35]. These results can be easily reproduced to provide an extensive test data suite containing different wiki text fragments to explore every detail specified by the grammar in a fully automated fashion. Such test data suites can be used to determine existing parsers' conformance, can help in developing new parsers, find problematic combinations that are treated differently by different parsers, etc.

Improve grammar readability. It is known that the grammar is meant to both define the language for the computer to parse, and describe it for the language engineers to understand. However, these two goals are usually conflicting, and more often than not, one opts for an executable grammar that is harder to read, than for a perfectly readable one that cannot be used in constructing grammarware. Unfortunately, the effort and expertise needed to fully achieve either of them, and most language documents contain non-operational grammars [28, 72, 79]. The practice of using two grammars: the "more readable" one and the "more implementable", adopted in the Java specification [18], has also proven to be very ineffective and error-prone [38, 39].

Perform automated adaptation. Grammars commonly need to be adapted in order to be useful and efficient in wide range of circumstances [32]. Grammar transformation frameworks such as GDK [30], GRK [34] or XBGF [74] can be used to apply adapting transformations in a safe disciplined way with validation of applicability preconditions and full control over the language delta. In fact, some of the transformations can even be generated automatically and applied afterwards.

Establish inter-grammar relationships. As of today, several MediaWiki notation grammars exist and are available in one form or another: in EBNF [76], in ANTLR [5], etc (none of them are fully operational). Furthermore, there exist various other wiki notations: Creole [22], Wikidot [68], etc. Relationships among all these notations are unknown: they are implicit even when formal grammars actually exist, and are totally obscured when the notation is only documented in a manual. A special technique called language convergence can help to reengineer such relationships in order to make stronger claims about compatibility and expressivity [37, 75, 77].

1.2 Related work: grammar recovery initiatives

Most of operational grammars for mainstream software languages are handcrafted, many are not publicly disclosed, few are documented. The first case reported in detail in 1998 was PLEX (Programming Language for EXchanges), a proprietary DSL for real time embedded software systems by Ericsson [59], a

successful application of the same technology to COBOL followed [62]. Grammar recovery technique is not only needed for legacy languages, examples of more modern and presumably more accurately engineered grammars being non-trivially extracted include C# in [73] and [75, §3] and Java in [38] and [39]. The whole process of MediaWiki grammar extraction is documented by this report, all corrections and refactorings are available online, as is the end result (under CC-BY-SA license).

1.3 Related work: Wiki Creole

Wiki Creole 1.0[1] is an attempt for engineering an ideal wiki syntax and a formal grammar for it. While the goal of specifying the wiki syntax with a grammar is not foreign to us, but the benefits listed in [22, p.3] are highly questionable:

1. **Trivial parser construction.** In the paper cited above it is claimed that applying a parser generator is trivial. However, the main prerequisite for it is successful grammar adaptation for the particular parsing technology [32]. A Wiki Creole grammar was specifically geared toward ANTLR, and it is a highly sophisticated task to migrate it anywhere if at some point ANTLR use is deemed to be undesirable. Hence, the result is not reproducible without considerable effort and expertise.

2. **Foundation for subsequent semantics specification.** The grammar can certainly serve as a basis for specifying semantics. However, the choice of a suitable calculus for such semantics specification is of even more importance. Furthermore, syntax definition does not guarantee the absence of ambiguities in semantics, or even changes of semantics as a part of language evolution (cf., evolutionary changes of HTML elements).

3. **Improved communication between wikiware developers.** The paper claimed that if wiki syntax is specified with a grammar, there can be no different interpretations of it. However, it is quite common to have different interpretations (dialects) of even mainstream programming languages, plus wiki technology in its current state heavily relies on fault tolerance (somewhat less so in the future when no bare text editing should be taking place).

4. **Same rendering behaviour that users rely on.** Depending on the browser or the particular gadget that the end user deploys to access the wiki, rendering behaviour can be vastly different, and this has nothing to do with the syntax specification.

5. **Simplified syntax extension.** It is a very known fact in formal grammar theory [1] that grammar classes are not compositional: that is, the result of combining two LL(*) grammars (which ANTLR uses) does not necessarily belong to the LL(*) class; we can only prove that it will still be

[1] http://wikicreole.org

context-free [9]. In other words, it is indeed easy to specify a syntax extension, but such the extended grammar sometimes will not be operational. Modular grammars can be deployed in frameworks which use different parsing technologies, such as in Meta-Environment [24] or in Rascal [26] or in MPS [66], but not in ANTLR.

6. **Performance predictions.** The paper claims that it is easier to predict performance of a parser made with "well-understood language theory" than with a parser based on regular expressions. However, there are implementation algorithms of regular expressions that demonstrate quadratic behaviour [12], and ANTLR uses the same technology for matching lookahead anyway, which immediately means that their performance is the same.

7. **Discovering ambiguities.** It is true that ambiguity analysis is easier on a formal grammar than on the prose, but it is not achieved by "more rigorous specification mechanism" and even the most advanced techniques of today do not always succeed [4].

8. **Well-defined interchange format.** A well designed interchange format between different types of wikiware is a separate effort that should be based on appropriate generalisations of many previously existing wiki notations, not on one artificially created one, even if that one is better designed.

In general, Wiki Creole initiative is relevant for us because it can serve as a common grammar denominator later to converge several wiki grammars [37, 77], but is neither contributing nor conflicting directly with our grammar recovery project.

2 Grammar notation

One of the first steps in grammar extraction is understanding the grammar definition formalism (i.e., the notation) used in the original artefact to describe the language. In the case of MediaWiki, Backus-Naur form is claimed to be used [45]. Manual cursory examination of the grammar text [47, 46, 51, 52, 49, 50, 48, 44] allows us to identify the following metasymbols in the spirit of [20] and [75]:

Name	Value
Start grammar symbol	`<source lang=bnf>`
End grammar symbol	`</source>`
Start comment symbol	`/*`
End comment symbol	`*/`
Defining symbol	`::=`
Definition separator symbol	`|`
Start nonterminal symbol	`<`
End nonterminal symbol	`>`
Start terminal symbol	`"`
End terminal symbol	`"`
Start option symbol	`[`
End option symbol	`]`
Start group symbol	`(`
End group symbol	`)`
Start repetition star symbol	{
End repetition star symbol	}
Start repetition plus symbol	{
End repetition plus symbol	}+

As we know from [3] and its research in [75, §6.3], BNF was originally defined as follows:

Name	Value
Defining symbol	:≡
Definition separator symbol	$\overline{\texttt{or}}$
Terminator symbol	↩
Start nonterminal symbol	`<`
End nonterminal symbol	`>`

While the difference in the appearances of defining symbols is minor and is commonly overlooked, there are several properties of the notation used for MediaWiki grammar definition that place it well outside BNF, namely:

- Using delimiters to explicitly denote terminal symbols (instead of using underlined decoration for keywords and relying on implicit assumptions for non-alphanumeric characters).
- Presence of comments in the grammar (not in the text around it).
- Allowing inconsistent terminator symbol (i.e., a newline or a double newline, sometimes a semicolon).
- Having metalanguage symbols for marking optional parts of productions.
- Having metalanguage symbols for marking repeated parts of productions.
- Having metalanguage symbols for grouping parts of productions.

Hence, it is not BNF. For the sake of completeness, let us compare it to the classic EBNF, originally proposed in [71] (sometimes that dialect is referred to as Wirth Syntax Notation) and standardised much later by ISO as [20]:

Name	Value in WSN	Value in ISO EBNF
Concatenate symbol		,
Start comment symbol		(*
End comment symbol		*)
Defining symbol	=	=
Definition separator symbol	|	|
Terminator symbol	.	;
Start terminal symbol	"	"
End terminal symbol	"	"
Start option symbol	[	[
End option symbol	]	]
Start group symbol	(	(
End group symbol	)	)
Start repetition star symbol	{	{
End repetition star symbol	}	}
Exception symbol		-
Postfix repetition symbol		*

We notice again a list of differences of MediaWiki grammar notation versus WSN and ISO EBNF:

- Allowing inconsistent terminator symbol (i.e., a newline or a double newline).
- Presence of comments (consistent only with ISO EBNF).
- Lack of concatenate metasymbol (consistent only with WSN).
- Having metalanguage symbol for exceptions (consistent only with WSN).
- Not having a specially designated postfix symbol for denoting repetition (consistent only with WSN).

Hence, the notation adopted by MediaWiki grammar, is neither BNF nor EBNF, but an extension of a subset of EBNF. Since we cannot reuse any previously existing automated grammar extractor, we define this particular notation with EDD (EBNF Dialect Definition), a part of SLPS (Software Language Processing Suite) [80] — and use **Grammar Hunter**, a universal configurable grammar extraction tool, for extracting the first version. The definition itself is a straightforward XML-ification of the first table of this section, so we leave it out of this document. The only addition is switching on the options of disregarding extra spaces and extra newlines that are left after tokenising the grammar. The EDD is freely available for re-use in the subversion repository of SLPS[2].

[2] Available as `config.edd`.

3 Guided grammar extraction

Since the grammar extraction process is performed for this particular notation for the first time, we use *guided* extraction, when the results of the extraction are visually compared to the original text by an expert in grammar engineering. This document is a detailed explanation of observations collected in that process and actions undertaken to resolve the spotted issues.

Given previous experience, it is safe to assume that once the grammar is extracted, we would like to change some parts of it (for grammar adaptation [32], deyaccification [58] and other activities common for grammar recovery [34]). In order for those changes to stay fully traceable and transparent, we will take the approach of *programmable grammar transformation*. In this methodology, we take a baseline grammar and an operator suite and by choosing the right operators and parametrising them, we *program* the desired changes in the same way mainstream programmers use programming languages to create software. These transformation scripts are executable with the grammar transformation engine: any meta-programming facility would suffice, for this particular work we use XBGF [74] which was shown in [39] to be the best and the most versatile grammar transformation infrastructure at this moment. The tools of SLPS that surround XBGF also allow for easy publishing by providing immediate possibilities to transform XBGF scripts to LaTeX or XHTML.

3.1 Source for extraction

The grammar of MediaWiki is available on subpages of [45]. Striving for more automation, we can use the "raw" action to download the content from the same makefile that performs the extraction[3]. For example, the wiki source of Article Title [47] is `http://www.mediawiki.org/w/index.php?title=Markup_spec/BNF/Article_title&action=raw`. In order to make our setup stable for the future when the contents of the wiki page may change (in fact, changing them is one of the main objectives of this work), we can add the revision number to that command, making it `http://www.mediawiki.org/w/index.php?title=Markup_spec/BNF/Article_title&action=raw&oldid=295042`.

3.2 Article title

Parsing Article Title [47] with Grammar Hunter is not hard and does not report many problems. One particular peculiarity that we notice when comparing the resulting grammar with the original, is the "`...?`" symbol:

```
<canonical-page-first-char> ::= <ucase-letter> | <digit> | <underscore> | ...?
<canonical-page-char>       ::= <letter> | <digit> | <underscore> | ...?
```

The "`...?`" symbol is not explained anywhere, but the intuitive meaning is that it is a metasymbol for a possible future extension point. For example, if in the future one decides to allow a hash symbol (`#`) in an article title (currently not allowed for technical reasons), it will be added as an alternative to the

[3] Available as `Makefile`.

production defining `canonical-page-char`. The very notion of such extension points contradicts the contemporary view on language evolution. It is commonly assumed that a grammar engineer cannot predict in advance all the places in the grammar that will need change in the future: hence, it is better to not mark any of such places explicitly and assume that any place can be extended, replaced, adapted, transformed, etc. Modern grammar transformation engines such as XBGF [74], Rascal [26] or TXL [13] all have means of extending a grammar in almost any desired place. Since it seems reasonable to remove these extension points at all, we can do it with XBGF after the extraction[4]:

```
vertical( in canonical-page-first-char );
removeV(
 canonical-page-first-char:
       "." "." "." "?"
);
horizontal( in canonical-page-first-char );
vertical( in canonical-page-char );
removeV(
 canonical-page-char:
       "." "." "." "?"
);
horizontal( in canonical-page-char );
vertical( in page-first-char );
removeV(
 page-first-char:
       "." "." "." "?"
);
horizontal( in page-first-char );
vertical( in page-char );
removeV(
 page-char:
       "." "." "." "?"
);
horizontal( in page-char );
```

By looking at the grammar where this transformation chain does not apply, one can notice productions in this style:

```
<canonical-article-title>       ::= <canonical-page> [<canonical-sub-pages>]
<canonical-sub-pages>          ::= <canonical-sub-page> [<canonical-sub-pages>]
<canonical-sub-page>           ::= <sub-page-separator> <canonical-page-chars>
```

In simple words, what we see here is an optional occurrence of a nonterminal called `canonical-sub-pages`, which is defined as a list of one or more nonterminals called `canonical-sub-page`. So, in fact, that optional occurrence consists of zero or more `canonical-sub-page` nonterminals. However, these observations are not immediate when looking at the definition, because the production is written with explicit right recursion. This style of writing productions belong to very early versions of compiler compilers like YACC [21], which required manual optimisation of each grammar before parser generation was possible. It has been reported later on multiple occasions [25, 58, etc] that it is highly undesirable to perform premature optimisation of a general purpose grammar for a specific parsing technology that may or may not be used with it at some

[4]Part of `remove-extension-points.xbgf`.

point in the future. The classic construct of a list of zero or more nonterminal occurrences is called a Kleene closure [1] or Kleene star (since it is commonly denoted as a postfix star) and is omnipresent in modern grammarware practice.

Using the Kleene star makes the grammars much more concise and readable. Most parser generators that require right-recursive (or left-recursive) expansions of a Kleene star, can do them automatically on the fly. Another possible reason for not using a star repetition could have been to stay within limits of pure BNF, but since we have already noted earlier that this goal was not reached anyway, we see no reason to pretend to seek it. A well-known grammar beautification technique known as "deyaccification" [58] is performed by the following grammar refactoring chain[5]:

```
massage(
 canonical-sub-pages?,
 (canonical-sub-pages | ε));
distribute( in canonical-sub-pages );
vertical( in canonical-sub-pages );
deyaccify(canonical-sub-pages);
inline(canonical-sub-pages);
massage(
 (canonical-sub-page+ | ε),
 canonical-sub-page*);
massage(
 canonical-page-chars?,
 (canonical-page-chars | ε));
distribute( in canonical-page-chars );
vertical( in canonical-page-chars );
deyaccify(canonical-page-chars);
inline(canonical-page-chars);
massage(
 (canonical-page-char+ | ε),
 canonical-page-char*);
massage(
 sub-pages?,
 (sub-pages | ε));
distribute( in sub-pages );
vertical( in sub-pages );
deyaccify(sub-pages);
inline(sub-pages);
massage(
 (sub-page+ | ε),
 sub-page*);
massage(
 page-chars?,
 (page-chars | ε));
distribute( in page-chars );
vertical( in page-chars );
deyaccify(page-chars);
inline(page-chars);
massage(
 (page-char+ | ε),
 page-char*);
```

[5] Part of deyaccify.xbgf.

Even the simplest metrics can show us that these refactorings have simplified the grammar, reducing it from 15 VAR and 25 PROD to 11 VAR and 17 PROD [55], without any fallback in functionality. They have also removed technological idiosyncrasies and improved properties that are somewhat harder to measure, like readability and understandability.

3.3 Article

Article [46] contains seven grammar fragments, out of which only the first three conform to the chosen grammar notation. The last four were copy-pasted from elsewhere and use a different EBNF dialect, which we luckily can also analyse and identify:

Name	Value
Defining symbol	=
Definition separator symbol	|
Start special symbol	?
End special symbol	?
Start terminal symbol	"
End terminal symbol	"
Start option symbol	[
End option symbol	]
Start group symbol	(
End group symbol	)
Start repetition star symbol	{
End repetition star symbol	}
Exception symbol	-

We will not lay out its step by step comparison with the notation used in the rest of the MediaWiki grammar, but it suffices to say that the presence of the exception symbol in the metalanguage is enough to make some grammars inexpressible in a metalanguage without it. BGF does not have a metasymbol for exception, but we still could express the dialect in EDD[6] and extract these parts of the grammar with it. Judging by the presence of the Kleene star in the metalanguage, the grammar engineers who developed those parts did not intend to stay within BNF limits. Thus, we can also advise to add the use of a plus repetition for denoting a sequence of one or more nonterminal occurrences, in order to improve readability of productions like these:

```
Line = PlainText { PlainText } { " " { " " } PlainText { PlainText } } ;
Text = Line { Line } { NewLine { NewLine } Line { Line } } ;
```

Or, in postfix-oriented BNF that we use within SLPS:

```
Line:
       PlainText PlainText* (" " " "* PlainText PlainText*)*
Text:
       Line Line* (NewLine NewLine* Line Line*)*
```

[6] Available at metawiki.edd.

Compare with the version that we claim to be more readable:

```
Line:
       PlainText+ (" "+ PlainText+)*
Text:
       Line+ (NewLine+ Line+)*
```

In fact, many modern grammar definition formalisms have a metaconstruct called "separator list", because `Text` above is nothing more than a (multiple) `Newline`-separated list of `Line`s. We do not enforce this kind of metaconstructs here, but we do emphasize the fact that the very understanding of `Text` being a separated list of `Line`s was not clear before our proposed refactoring. In the case if MediaWiki still wants the grammar representation to have only one type of repetition or even no repetition at all, such a view can be automatically derived from the baseline grammar preserved in a more expressive metalanguage. The refactorings that utilise the plus notation are rather straightforward[7]:

```
massage(
 PlainText PlainText*,
 PlainText+);
massage(
 Line Line*,
 Line+);
massage(
 NewLine NewLine*,
 NewLine+);
massage(
 " " " "*,
 " "+);
```

Further investigation draws our attention to these productions:

```
PageName = TitleCharacter , { [ " " ] TitleCharacter } ;
PageNameLink = TitleCharacter , { [ " " | "_" ] TitleCharacter } ;
```

The comma used in both productions is not a terminal symbol "`,`": in fact, it is a concatenate symbol from ISO EBNF [20]. Since ISO EBNF is not the notation used, the commas must have been left out unintentionally—this is what usually happens when grammars are transformed manually and not in a disciplined way. Grammar Hunter assumed that the quotes were forgotten in this place (since a comma is not a good name for a nonterminal), so we need to project it away (the corresponding operator is called **abstractize** because it shifts a grammar from concrete syntax to abstract syntax). These are the transformations that we write down[8]:

```
abstractize(
 PageName:
       TitleCharacter ⟨","⟩ (" "? TitleCharacter)*
);
abstractize(
 PageNameLink:
       TitleCharacter ⟨","⟩ ((" " | "_")? TitleCharacter)*
```

[7] Part of `utilise-repetition.xbgf`.
[8] Complete listing of `remove-concatenation.xbgf`.

```
);
```

The following fragment uses excessive bracketing: parenthesis are used to group symbols together, which is usually necessary for inner choices and similar cases when one needs to override natural priorities. However, in this case it is unnecessary:

```
SectionTitle = ( SectionLinkCharacter - "=" )
                                    { [ " " ] ( SectionLinkCharacter - "=" ) } ;
LinkTitle = { UnicodeCharacter { " " } } ( UnicodeCharacter - "]" ) ;
```

Excessive bracketing is not a problem for SLPS toolset since all BGF grammars are normalised before serialisation, and it includes a step of refactoring trivial subsequences, but we still report it for the sake of reproducibility within a different environment.

The following grammar production uses a strange-looking construction that is explained in the text to be the "non-greedy" variant of the optional newline:

```
<special-block-and-more>  ::=
                 <special-block> ( EOF | [<newline>] <special-block-and-more>
                                       | (<newline> | "") <paragraph-and-more> )
```

The purpose of a syntax definition such as a BNF is to define syntax of a language. Thus, any references to the semantics of the parsing process should be avoided. The definition of "greediness" as ordered alternatives, given at the first page of [45], contradicts the classic definition based on token consumption, and contradicts the basics of EBNF. Approaches alternative to context-free grammars such as PEG [17] should be considered if ordered alternatives are really required. For EBNF (or BGF), we refactor the singularity as follows[9]:

```
massage(
 (newline | ε),
 newline?);
```

Since at this point the subgrammar of this part must be rather consistent, we can execute some simple grammar analyses to help assess the grammar quality. One of them is based on a well-known notion of bottom and top nonterminals [58, 59]: a top is one that is defined but never used; a bottom is one that is used but never defined. We were surprised to see `WhiteSpaces` in the list of top nonterminals, while `Whitespaces` was in the list of bottom nonterminals. Apparently, a renaming is needed[10]:

```
unite(WhiteSpaces, Whitespaces);
```

The definition of nonterminal `BlockHTML` contains textual annotation claiming that it is not yet referred to. We decided to parse it anyway and validate that assertion afterwards. Indeed, it showed up as an unconnected grammar fragment, which we can then safely remove[11]:

```
eliminate(BlockHTML);
```

[9] Part of `utilise-question.xbgf`.
[10] Part of `unify-whitespace.xbgf`.
[11] Part of `connect-grammar.xbgf`.

3.4 Noparse block

Apart from quoting the language name in the source tags, which makes the start grammar symbol change from `<source lang=bnf>` to `<source lang="bnf">`, the Noparse Block [51] uses the same EBNF dialect that we derived as the starting step of our extraction. However, there are two major exceptions:

- Round brackets and square brackets have swapped their meaning.
- A lookahead assertion metasymbol is used, borrowed from Perl Compatible Regular Expressions library.

The first impression given by cursory examination of the extracted grammar is that it uses excessive bracketing (mentioned in the previous section):

```
<pre-block>               ::= <pre-opening-tag> (<whitespace>) <pre-body>
                                  (<whitespace>) [<pre-closing-tag> | (?=EOF) ]
<pre-opening-tag>         ::= "&lt;pre" (<whitespace> (<characters>)) "&gt;"
<pre-closing-tag>         ::= "&lt;/pre" (<whitespace>) "&gt;"
<pre-body>                ::= <characters>
```

However, if we assume this to be true, the meaning of the grammar will become inadequate: for example, it will have mandatory whitespace in many places. On the other hand, making the last part of the grammar (`<nowiki-closing-tag> | (?=EOF)`) optional is also inadequate, because optional assertion will never make sense. This particular lookahead assertion is displayed as `(?=EOF)` and means basically an ε that must be followed by EOF (even that definition is not that apparent from the low-level description saying *"It asserts that an EOF follows, but does not consume the EOF."*). The presence or absence of lookahead based facilities is heavily dependent on the parsing technology, and therefore should be avoided as much as possible, as noted by multiple sources [25, 36, 58]. More straightforward and high level assertions like "should be followed by" and "should not be followed by" are available in modern metaprogramming languages like Rascal [26] instead.

Since the general problem of leaving opened tags at the end of the article text is much bigger than the tags described in this part of the grammar, we opt for removing these assertions altogether and solving the problem later with suitable technology. EBNF has never been intended for and has never been good at defining tolerant parsers [27]. Since we have to construct another EBNF dialect in order to parse the Noparse Block fragment correctly anyway, we specify "`(?=EOF)`" as a notation for ε (otherwise we would have to fix the problem later with a horizontal remove operator from XBGF). Those explicit empty sequence metasymbols need to be refactored into proper optional symbols[12]:

```
massage(
 (nowiki-closing-tag | ε),
 nowiki-closing-tag?);
massage(
 (pre-closing-tag | ε),
 pre-closing-tag?);
```

[12] Part of `remove-lookahead.xbgf`.

```
massage(
 (html-closing-tag | ε),
 html-closing-tag?);
```

In every notation that comprises similar looking symbols and metasymbols that can be encountered within the same context, there is need for *escaping* some special characters. In this part of the MediaWiki grammar escaping is done in HTML entities, which is not explainable with grammar-based arguments. However, we recall that our extraction source is a handcrafted grammar that was meant to reproduce the behaviour of the MediaWiki PHP parse—so, in a sense, it was (manually) extracted, and what we have just encountered is in fact a legacy artefact randomly inherited from its source. Such legacy should be removed by following transformation steps[13]:

```
renameT("&lt;nowiki", "<nowiki");
renameT("&lt;/nowiki", "</nowiki");
renameT("&lt;pre", "<pre");
renameT("&lt;/pre", "</pre");
renameT("&lt;html", "<html");
renameT("&lt;/html", "</html");
renameT("&lt;!--", "<!--");
replace("&gt;", ">");
```

There are two more problems in the Noparse Block part that concern the nonterminal `characters`. First, it is undefined (bottom). As we will see in §3.8, there is a nonterminal called `character`—issues like these with "forgetting" to define some nonterminals with readable names are quite common in handcrafted grammars, as noted by [28] and other sources. A trivially guessed definition for `characters` is either "one-or-more" or "zero-or-more" repetition of `character`. Since `characters` is mostly used as an optional nonterminal, we assume that it is one or more[14]:

```
define(
 characters:
       character+
);
```

The second problem is its usage in `html-comment` (remember that round brackets mean optionality here):

```
<html-comment>              ::= "&lt;!--" ({ characters }) "-->"
```

Since we do not need to make a Kleene repetition optional, we can refactor it as follows[15]:

```
unfold(characters in html-comment);
massage(
 character+*,
 character*);
massage(
 character*?,
 character*);
```

[13] Complete listing of `dehtmlify.xbgf`.
[14] Part of `connect-grammar.xbgf`.
[15] Part of `refactor-repetition.xbgf`.

```
/* not properly fleshed out, haven not tried all the combinations */
<article-link>          ::= [<interwiki-prefix> | ":"] [<namespace-prefix
                          | "/" <article-title>
                          | { "../" } [<article-title>]

<article-title>         ::= { [<title-legal-chars> | "%" ] } +

/* Specified using regex format, obviously... */
<title-legal-chars>     ::= " %!\"$&'()*,\\-.\\/0-9:;=?@A-Z\\\\^_`a-z~\\x


<interwiki-prefix>        ::= <interwiki> ":"
<interwiki>               ::= STRING_FROM_DB

<namespace-prefix>        ::= [<namespace>] ":"
<namespace>               ::= STRING_FROM_CONFIG
/* is it? parser.php gives it as  "[_0-9A-Za-z\x80-\xff]" */

<link-description>        ::= LEGAL_ARTICLE_ENTITY
<extra-description>       ::= <letter> [<extra-description>]

<internal-link-start>     ::= "[["
<internal-link-end>       ::= "]]"

<pipe>                    ::= "|"

/* Almost anything seems to be allowed, but it won't necessarily be treat
<section-id>            ::= { [<title-legal-chars> | "%" | "#" ] } +
```

Figure 1: A syntax that even MediaWiki cannot colour-code properly [52].

```
massage(
 character*,
 character+?);
fold(characters in html-comment);
```

More detailed information about leaving combinations of various kinds of repetition and optionality in the deployed grammar will be given in the next section.

3.5 Links

Links definitions [52] exhibit bits of yet another notation, namely the one where a set of possible values is given, assuming that only one should be picked. In the MediaWiki grammar it is erroneously called a "regex format"—regular expressions do use this notation in some places, but not everywhere and it is not exclusive to them. This notation is very much akin to "one-of" metaconstructs also encountered in definitions of other software languages such as C# [75, §3.2.4]. In the MediaWiki grammar, it looks like this:

```
/* Specified using regex format, obviously... */
<title-legal-chars>  ::= " %!\"$&'()*,\\-.\\/0-9:;=?@A-Z\\\\^_`a-z~\\x80-\\xFF+"
```

The unobviousness of the notation is perfectly simplified by the fact that even the MediaWiki engine itself fails to parse and colour-code it correctly, as seen on Figure 1. In fact, when we look at the expression more closely, we can notice that it is even incorrect in itself, since it uses double-escaping for most backslashes (ruining them) and does not escape the dot (which denotes any character when unescaped). Some other characters like `*` or `+` should arguably

also be escaped, but it is impossible to decide firmly on escaping rules when we have no engine to process this string. However, the correct expression should have looked similar to this:

```
<title-legal-chars>  ::= " %!\"$&'()*,\-\.\/0-9:;=?@A-Z\\^_‘a-z~\x80-\xFF+"
```

Which we rewrite as (some invisible characters are omitted for readability):

```
<title-legal-chars>    ::= " " | "%" | "!" | """ | "$" | "&" | "'" | "(" | ")"
 | "*" | "," | "-" | "." | "/"
 | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
 | ":" | ";" | "=" | "?" | "@"
 | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M"
 | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
 | "\" | "^" | "_" | "`"
 | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
 | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
 | "~" | "¡" | "¢" | "£" | "¤" | "¥" | "¦"
 | "§" | "¨" | "©" | "ª" | "«" | "¬" | " " | "®" | "¯" | "°" | "±" | "²" | "³"
 | "´" | " " | "¶" | "·" | "¸" | "¹" | "º" | "»" | "¼" | "½" | "¾" | "¿" | "À"
 | "Á" | "Â" | "Ã" | "Ä" | "Å" | "Æ" | "Ç" | "È" | "É" | "Ê" | "Ë" | "Ì" | "Í"
 | "Î" | "Ï" | "Ð" | "Ñ" | "Ò" | "Ó" | "Ô" | "Õ" | "Ö" | "×" | "Ø" | "Ù" | "Ú"
 | "Û" | "Ü" | "Ý" | "Þ" | "ß" | "à" | "á" | "â" | "ã" | "ä" | "å" | "æ" | "ç"
 | "è" | "é" | "ê" | "ë" | "ì" | "í" | "î" | "ï" | "ð" | "ñ" | "ò" | "ó" | "ô"
 | "õ" | "ö" | "÷" | "ø" | "ù" | "ú" | "û" | "ü" | "ý" | "þ" | "ÿ" | "+"
```

This refactored version with all alternatives given explicitly was created automatically by a trivial Python one-liner and can be parsed without any trouble by Grammar Hunter. We should also note that the name for this nonterminal is misleading, since it represents only one character. This is not a technical mistake, but we can improve *learnability* of the grammar by fixing it[16]:

```
renameN(title-legal-chars, title-legal-char);
```

Grammar Hunter displays an error message but is capable of dealing with this fragment:

```
<article-link> ::=
                [<interwiki-prefix> | ":" ] [<namespace-prefix] <article-title>
```

The problem in this grammar production is in "`[<namespace-prefix]`" (note the unbalanced angle brackets). The start nonterminal symbol here is followed by the name of the nonterminal and then by the end option symbol without the end nonterminal symbol. This kind of problems are rather common in grammars that have been created manually and have never been tested in any environment that would make them executable or validate consistency otherwise. Grammar Hunter can resolve this problem by using the heuristic of next best guess, which is to assume that the nonterminal name ended at the first alphanumeric/non-alphanumeric border that happened after the unbalanced start nonterminal symbol.

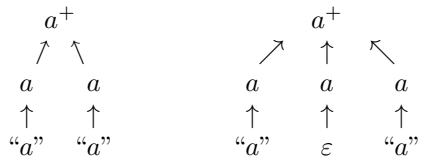
Next, consider the following two grammar productions that lead to several problems simultaneously:

[16] Part of `fix-names.xbgf`.

```
<article-title>          ::= { [<title-legal-chars> | "%" ] } +
<section-id>             ::= { [<title-legal-chars> | "%" | "#" ] } +
```

As we have explained above, the grammar notation used for the MediaWiki grammar was never defined explicitly in any formal or informal way, so we had to infer it in §2. When inferring its semantics, we had two options: to treat the plus as a postfix metasymbol or to treat "{" and "}+" as bracket metasymbols. Both variants are possible and feasible, since Grammar Hunter is capable of dealing with ambiguous starting metasymbols (i.e., "{" as both a start repetition star symbol and a start repetition plus symbol). We obviously opt for the latter variant because from the formal language theory we all know that for any x it is always true that $(x^*)^+ \equiv x^*$, so a postfix plus operation on a star repetition is useless and we tend to assume good faith of grammar engineers who made use of it. But even if we assume it to be a transitive closure (a plus repetition), which is a common notation for a sequence of one or more occurrences of a subexpression, the productions become parseable, but they are bound to deliver problems with ambiguities [4] on later stages of grammar deployment, since in these particular grammar fragments optional symbols are iterated.

To give a simple example, suppose we have a nonterminal x defined as a^+, and a itself is defined as "a"? (either "a" or ε). Then the following are two distinct possibilities to parse "aa" with such a grammar:



The number of such ways to parse even the simplest of expressions is infinite, and special algorithms need to be utilised to detect such problems at the parser generator level. Thus, to prevent this trouble from happening, we massage the productions above to use a simple star repetition instead, which is an equivalent unambiguous construct[17]:

```
massage(
 (title-legal-chars | "%")?+,
 (title-legal-chars | "%")*);
massage(
 (title-legal-chars | "%" | "#")?+,
 (title-legal-chars | "%" | "#")*);
```

Reading further, we notice one of the nonterminals being defined with explicit right recursion:

```
<extra-description>      ::= <letter> [<extra-description>]
```

The problem is known and has been discussed above, all we need here is proper deyaccification[18]:

```
massage(
```

[17] Part of `utilise-repetition.xbgf`.
[18] Part of `deyaccify.xbgf`.

```
 extra-description?,
 (extra-description | ε)
 in extra-description);
distribute( in extra-description );
vertical( in extra-description );
deyaccify(extra-description);
```

The last problem with the Links part of the grammar is the use of natural language inside a BNF production:

```
<protocol>          ::= ALLOWED_PROTOCOL_FROM_CONFIG (e.g. "http://", "mailto:")
```

Examples are never a part of a syntax definition: the alternatives are either listed exhaustively (like we will do later when we make the grammar complete) or belong in the comments (like it was undoubtedly intended here). A projection is needed to remove them from the raw extracted grammar[19]:

```
project(
 protocol:
       ALLOWED_PROTOCOL_FROM_CONFIG ⟨(e "." g "." "http://" "," "mailto:")⟩
);
```

3.6 Magic links

Just like Noparse Block discussed above in §3.4, Magic Links [49] also uses `<source lang="bnf">` as the start grammar symbol, but this is the least problem encountered in this fragment. Consider the following productions:

```
<isbn>          ::= "ISBN" (" "+) <isbn-number> ?(non-word-character /\b/)
<isbn-number>  ::= ("97" ("8" | "9") (" " | "-")?) (DIGIT (" " | "-")?)
                                                    {9} (DIGIT | "X" | "x")
```

We see a notation where:

- A postfix plus repetition metasymbol is used, which is not encountered anywhere else in the MediaWiki.
- The character used as the postfix repetition metasymbol clashes with end repetition plus metasymbol from Inline Text [48] and Links [52][20].
- A postfix optionality metasymbol is used, which is not encountered anywhere else in the MediaWiki.
- The character used as the postfix optionality metasymbol clashes with start special metasymbol and end special metasymbol from Article [46], Inline Text [48] and Special Block [50].
- The same character used as the postfix optionality metasymbol is used as in a prefix notation that relies on lookahead.
- A regular expression is used inside the lookahead assertion.

[19] Complete listing of `remove-comments.xbgf`.

[20] Indirect clash of "}+" being an end repetition plus symbol as well as a sequence of an end repetition star symbol and a postfix repetition metasymbol.

- A terminal symbol (“9”) is not explicitly marked as such.
- A nonterminal symbol (“DIGIT”) is not explicitly marked as such.

Along with the discussion from §3.4, we first remove the lookahead assertions. They (arguably) do not belong in EBNF at all, and definitely do not belong in such a form[21]:

```
project(
 isbn:
       "ISBN" " " " "+" isbn-number ⟨("?" non-word-character "/" "\" b "/")⟩
);
```

We do not even try to add the postfix plus repetition metasymbol to the notation definition, since it is used only once, since it clashes with something else, and since there is a special nonterminal `spaces` that should be used instead anyway[22]:

```
replace(
 " " " "+",
 spaces);
```

Then we adjust the grammar for the untreated postfix question metasymbol[23]:

```
abstractize(
 isbn-number:
       "97" ("8" | "9") (" " | "-") ⟨"?"⟩ DIGIT (" " | "-") ⟨"?"⟩ "9"*
              (DIGIT | "X" | "x")
);
widen(
 (" " | "-"),
 (" " | "-")?
 in isbn-number);
```

3.7 Special block

Just as in [47], the Special Block uses a special metasymbol for omitted grammar fragments [50]. This case is subtly different from the one discussed in §3.2 in a sense that it explicitly says in the accompanying text that “The dots need to be filled in”. This information is undoubtedly useful, but considering the fact that its very presence renders the grammar non-executable, we decide to remove it from the grammar and let the documentation tell the story about how much of the intended language does the grammar cover[24]:

```
vertical( in special-block );
removeV(
 special-block:
       "." "." "."
);
horizontal( in special-block );
```

[21] Part of `remove-lookahead.xbgf`.
[22] Part of `unify-whitespace.xbgf`.
[23] Part of `utilise-question.xbgf`.
[24] Part of `remove-extension-points.xbgf`.

In the same first production there is an alternative that reads `<nowiki><table></nowiki>`, which seems like either a leftover after manually cleaning up the markup, or a legacy escaping trick. Either way, `nowiki` wrapping is not necessary for displaying this fragment and is generally misleading: the chevrons around "`table`" mean to denote it explicitly as a nonterminal, not as an HTML tag. We project away the unnecessary parts[25]:

```
vertical( in special-block );
project(
 special-block:
       ⟨nowiki⟩ table ⟨/ nowiki⟩
);
horizontal( in special-block );
```

There are also more cases of excessive bracketing which are fixed automatically by Grammar Hunter:

```
<defined-term>            ::= ";" <text> [ (<definition>)]
```

A nonterminal symbol called `dashes` is arguably superfluous and can be replaced by a Kleene star of a dash terminal:

```
<horizontal-rule>         ::= "----" [<dashes>] [<inline-text>] <newline>
<dashes>                  ::= "-" [<dashes>]
```

Still, we can keep it in the grammar for the sake of possible future BNF-ification, but refactor the idiosyncrasy (the right recursion)[26]:

```
massage(
 dashes?,
 (dashes | ε)
 in dashes);
distribute( in dashes );
vertical( in dashes );
deyaccify(dashes);
```

The worst part of the Special Block part is the section titled "Tables": it contains eight productions in a different notation, with a comment "From meta...minor reformatting". This reformatting has obviously been performed manually, since it does not utilise the standard notation of the rest of the grammar, nor is it compatible with the MetaWiki notation that we have encountered in §3.3: the defining symbol is from the MediaWiki notation, the terminator symbol is from the MetaWiki notation, etc:

[25] Part of `fix-markup.xbgf`.
[26] Part of `deyaccify.xbgf`.

Name	Value
Defining symbol	`::=`
Terminator symbol	`;`
Definition separator symbol	`|`
Start special symbol	`?`
End special symbol	`?`
Start terminal symbol	`"`
End terminal symbol	`"`
Start nonterminal symbol	`<`
End nonterminal symbol	`>`
Start option symbol	`[`
End option symbol	`]`

To save the trouble of post-extraction fixing, we used this configuration as a yet another EDD file to extract this grammar fragment and merge it with the rest of the grammar. The naming convention of the fragment is still not synchronised with the rest (i.e., camel case vs. dash-separated lowercase), but we will deal with it later in §5.

We also see a problem similar to the one discussed above in §3.4, namely an optional zero-or-more repetition:

```
<space-block>             ::= " " <inline-text> <newline> [ {<space-block-2} ]
```

The solution is also already known to us[27]:

```
massage(
 space-block-2*?,
 space-block-2*);
```

When comparing the list of top nonterminals with the list of bottom ones, we notice `TableCellParameters` being used while `TableCellParameter` being defined. Judging by its clone named `TableParameters`, the intention was to name it plural, so we perform unification[28]:

```
unite(TableCellParameter, TableCellParameters);
```

3.8 Inline text

Suddenly, [48] uses bulleted-list notation for listing alternatives in a grammar:

```
<text-with-formatting>    ::=
                            | <formatting>
                            | <inline-html>
                            | <noparseblock>
                            | <behaviour-switch>
                            | <open-guillemet> | <close-guillemet>
                            | <html-entity>
                            | <html-unsafe-symbol>
                            | <text>
                            | <random-character>
                            | (more missing?)...
```

[27] Part of `refactor-repetition.xbgf`.
[28] Part of `fix-names.xbgf`.

This is almost never encountered in grammar engineering, but not completely unknown to computer science—for example, TLA$^+$ uses this notation [40]. In our case it is confusing for Grammar Hunter since newlines are also used in the notation to separate production rules, and since it only happens in two productions, we decide to manually remove the first bar there. The last line of the sample above also shows an extension point discussed earlier in §3.2 and §3.7, which we remove[29]:

```
vertical( in text-with-formatting );
removeV(
 text-with-formatting:
       (more missing "?") "." "." "."
);
horizontal( in text-with-formatting );
```

Nonterminal `noparseblock` is referenced in the same grammar fragment, but never encountered elsewhere in the grammar, later we will unite it with `noparse-block` when specifically considering enforcing consistent naming convention in §5.5.

The next problematic fragment is the following:

```
<html-entity-name>      ::= Sanitizer::$wgHtmlEntities (case sensitive)
 (* "Aacute" | "aacute" | ... *)
```

It has three problems:

- Referencing PHP variables from the grammar is unheard of.
- Static semantics within postfix parenthesis in plain English is not helpful.
- A comment that uses "`(*`" and "`*)`" as delimiters instead of "`/*`" and "`*/`" used in the rest of the grammar.

These identified problems can be solved with projecting excessive symbols, leaving only one nonterminal reference, which will remain undefined for now[30]:

```
project(
 html-entity-name:
       ⟨(Sanitizer ":" ":" "$")⟩ wgHtmlEntities ⟨(case sensitive (("*" "Aacute")
                     | "aacute" | ("." "." "." "*")))⟩
);
```

Later in §5.6 we will reuse the source code of `Sanitizer` class to formally complete the grammar by defining `wgHtmlEntities` nonterminal.

The following fragment combines two double problems that have already been encountered before. The first problem is akin to the one we have noticed in §3.5, namely having a nonterminal with "`-characters`" in its name, which is supposed to denote only one character taken from a character class; the second part of that problem is the usage of the regular expression notation. The second problem is an omission/extension point (cf. §3.2 and §3.7), which is expressed in Latin:

[29] Part of `remove-extension-points.xbgf`.
[30] Complete listing of `remove-php-legacy.xbgf`.

```
<harmless-characters>    ::= /[A-Za-z0-9] etc
```

We rewrite it as follows:

```
<harmless-characters>    ::=
   "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M"
 | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
 | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
 | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
 | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The name of the nonterminal symbol `harmless-characters` is misleading, since it represents only one character. In fact, simple investigation into top and bottom nonterminals [36] shows that it is not referenced anywhere in the grammar, but a nonterminal `harmless-character` is used in the definition of `text`. Hence, we want to unite those two nonterminals[31]:

```
unite(harmless-characters, harmless-character);
```

The immediately following production contains a special symbol written in the style of ISO EBNF and MetaWiki:

```
<random-character>       ::= ? any character ... ?
```

Instead of adjusting the assumed notation definition, we choose to let Grammar Hunter parse it as it is, and to subsequently transform the result to a special BGF metasymbol with the same semantics (i.e., "any character")[32]:

```
redefine(
 random-character:
       ANY
);
```

The next problematic fragment once again contains omission/extension points:

```
<ucase-letter>           ::= "A" | "B" | ... | "Y" | "Z"
<lcase-letter>           ::= "a" | "b" | ... | "y" | "z"
<decimal-digit>          ::= "0" | "1" | ... | "8" | "9"
```

Since in fact they represent all possible alternatives from the given range, we rewrite them as follows:

```
<ucase-letter>           ::=
  "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M"
| "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
<lcase-letter>           ::=
   "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m"
 | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
<decimal-digit>          ::=
   "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The same looking metasymbol is used later as a pure extension point:

```
<symbol>              ::= <html-unsafe-symbol> | <underscore> | "." | "," | ...
```

The crucial difference in the semantics of these two metasymbols both denoted as "`...`" lies in the fact that in the former one (i.e., `"A" | ... | "Z"`) it is basically a macro definition that can be expanded by any human reader, but in the latter one (i.e., `"." | ...`) the only thing the reader learns from

[31] Part of `fix-names.xbgf`.

[32] Part of `define-lexicals.xbgf`.

looking at it is that something can or should be added. Hence, following the conclusions we drew above, we expand the former omission metasymbol right in the grammar source, but we remove the latter omission metasymbol with grammar transformation[33]:

```
vertical( in symbol );
removeV(
 symbol:
       "." "." "."
);
horizontal( in symbol );
```

Finally, we notice some of the productions using explicit right recursion:

```
<newlines>              ::= <newline> [<newlines>]
<space-tabs>            ::= <space-tab> [<space-tabs>]
<spaces>                ::= <space> [<spaces>]
<decimal-number>        ::= <decimal-digit> [<decimal-number>]
<hex-number>            ::= <hex-digit> [<hex-number>]
```

The deyaccifying transformation steps are straightforward[34]:

```
massage(
 newlines?,
 (newlines | ε)
 in newlines);
distribute( in newlines );
vertical( in newlines );
deyaccify(newlines);
massage(
 space-tabs?,
 (space-tabs | ε)
 in space-tabs);
distribute( in space-tabs );
vertical( in space-tabs );
deyaccify(space-tabs);
massage(
 spaces?,
 (spaces | ε)
 in spaces);
distribute( in spaces );
vertical( in spaces );
deyaccify(spaces);
massage(
 decimal-number?,
 (decimal-number | ε)
 in decimal-number);
distribute( in decimal-number );
vertical( in decimal-number );
deyaccify(decimal-number);
massage(
 hex-number?,
 (hex-number | ε)
 in hex-number);
distribute( in hex-number );
vertical( in hex-number );
```

[33] Part of remove-extension-points.xbgf.
[34] Part of deyaccify.xbgf.

```
deyaccify(hex-number);
```

We should specially note here that the form used to define `spaces` prevented us earlier in §3.6 from using less invasive grammar transformation operators. What we ideally want is a transformation that is as semantics preserving as possible[35]:

```
fold(space);
fold(spaces);
```

It is intentional that these two steps affect the whole grammar. We will return to this issue later in §5.2.

There are two views given on `formatting`: an optimistic one and a realistic one. Since the grammar needs to define the allowed syntax in a structured way, we scrap the the latter[36]:

```
removeV(
 formatting:
       apostrophe-jungle
);
eliminate(apostrophe-jungle);
```

The whole section describing Inline HTML was removed from [48] prior to extraction because it combines two aspects that are not intended to be defined with (E)BNF: it defines a different language embedded inside the current one (this can be done in a clean way by using modules in advanced practical frameworks like Rascal [26]) and it tries to define rules for automated error fixing (cf. fault-tolerant parsing, tolerant parsing, etc). It suffices to note here that the metalanguage used in the parts of that section that were formulated not in plain English, is fascinatingly different from the parts of the MediaWiki grammar that we have already processed: it uses attributed (parametrised) nonterminals and postfix modifiers for case (in)sensitivity. The same metasyntax is used in the next section about images, so we do need to find a way to process chunks like this:

[35] Part of `unify-whitespace.xbgf`.

[36] Complete listing of `remove-duplicates.xbgf`.

```
 ImageModeManualThumb  ::= mw("img_manualthumb");
 ImageModeAutoThumb    ::= mw("img_thumbnail");
 ImageModeFrame        ::= mw("img_frame");
 ImageModeFrameless    ::= mw("img_frameless");

/* Default settings: */
 mw("img_manualthumb") ::= "thumbnail=", ImageName | "thumb=", ImageName
 mw("img_thumbnail")   ::= "thumbnail" | "thumb";
 mw("img_frame")       ::= "framed" | "enframed" | "frame";
 mw("img_frameless")   ::= "frameless";

 ImageOtherParameter   ::= ImageParamPage | ImageParamUpright | ImageParamBorder
 ImageParamPage        ::= mw("img_page")
 ImageParamUpgright    ::= mw("img_upright")
 ImageParamBorder      ::= mw("img_border")

/* Default settings: */
 mw("img_page")        ::= "page=$1" | "page $1"  ??? (where is this used?)
 mw("img_upright")     ::= "upright" [, ["=",] PositiveInteger]
 mw("img_border")      ::= "border"
```

We try to list the problems within that grammar fragment:

- Parametrised nonterminals are used in a style of function calls. This is not completely uncommon to grammarware since the invention of van Wijngaarden grammars [63] and attribute grammars [29], but unnecessary here.
- Some productions end with a terminator symbol "`;`", others don't.
- Concatenate metasymbol "`,`" is used rather inconsistently (occurs between some metasymbols, doesn't occur between some nonterminal symbols).
- Inline comments are given in English without consistent explicit separation from the BNF formulae.

The shortest way to overcome these difficulties is to reformat them lexically, unchaining parametrised nonterminals and appending terminator symbols to productions that did not have them. The result looks like this:

```
 ImageModeManualThumb ::= "thumbnail=", ImageName | "thumb=", ImageName ;
 ImageModeAutoThumb   ::= "thumbnail" | "thumb";
 ImageModeFrame       ::= "framed" | "enframed" | "frame";
 ImageModeFrameless   ::= "frameless";

 ImageOtherParameter  ::= ImageParamPage | ImageParamUpright | ImageParamBorder
 ImageParamPage       ::= "page=$1" | "page $1"; /* ??? (where is this used?) */
 ImageParamUpgright   ::= "upright" [, ["=",] PositiveInteger]
 ImageParamBorder     ::= "border"
```

One of the fragments fixed in this way contains postfix metasymbols for case insensitivity:

```
 <behaviourswitch-toc>            ::= "__TOC__"i
 <behaviourswitch-forcetoc>       ::= "__FORCETOC__"i
 <behaviourswitch-notoc>          ::= "__NOTOC__"i
 <behaviourswitch-noeditsection>  ::= "__NOEDITSECTION__"i
 <behaviourswitch-nogallery>      ::= "__NOGALLERY__"i
```

These untypical metasymbols are parsed by Grammar Hunter as separate nonterminals, which we remove by projection[37]:

```
project(
 behaviourswitch-toc:
       "__TOC__" ⟨i⟩
);
project(
 behaviourswitch-forcetoc:
       "__FORCETOC__" ⟨i⟩
);
project(
 behaviourswitch-notoc:
       "__NOTOC__" ⟨i⟩
);
project(
 behaviourswitch-noeditsection:
       "__NOEDITSECTION__" ⟨i⟩
);
project(
 behaviourswitch-nogallery:
       "__NOGALLERY__" ⟨i⟩
);
```

There is also a mistake that is easily overlooked unless you analyse top and bottom nonterminals (look at the second option):

```
 ImageAlignParameter        ::= ImageAlignLeft | ImageAlign|Center |
                                ImageAlignRight | ImageAlignNone
```

This extra unnecessary bar is parsed as a regular choice separator, so we need to fix it this way[38]:

```
replace(
 (ImageAlign | Center),
 (ImageAlignCenter));
```

The same analysis shows us a fragment in the resulting grammar, which is unconnected because `ImageOption` does not list it with the others[39]:

```
vertical( in image-option );
addV(
 image-option:
       image-other-parameter
);
horizontal( in image-option );
```

The first and the last productions of the Images subsection contain an explicitly marked nonterminal symbol:

```
ImageInline                ::= "[[" , "Image:" , PageName, ".",
             ImageExtension, ( { <Pipe>, ImageOption, } ) "]]" ;
Caption                    ::= <inline-text>
```

A production in the middle of the Images subsection and the first production of the Media subsection make inconsistent use of a concatenate symbol:

[37] Complete listing of `remove-postfix-case.xbgf`.
[38] Part of `fix-names.xbgf`.
[39] Part of `connect-grammar.xbgf`.

```
ImageSizeParameter          ::= PositiveNumber "px" ;
MediaInline             ::=  "[[" , "Media:" , PageName "." MediaExtension "]]" ;
```

And, finally, the last production of the Media subsection contains a wrong defining symbol:

```
MediaExtension = "ogg" | "wav" ;
```

These three problems were reported and overcome by Grammar Hunter but not solved automatically, because usually there is more that one way to resolve such issues, and a human intervention is needed to make a choice. After the unified notation is enforced everywhere, we can extract the grammar and continue recovering it with grammar transformation steps. It should be noted that Grammar Hunter could not resolve the lack of concatenate symbols, since it starts assuming that the following symbol is a part of the current one (originally the concatenate symbol was proposed in [20] in order to allow nonterminal names contain spaces), but it easily dealt with excessive concatenate symbols because they just virtually insert ε here and there, which gets easily normalised.

Back to the rest of the section, we have a fragment with essentially an extension point specified in plain English as the right hand side of a production:

```
GalleryImage              ::=   (to be defined: essentially   foo.jpg[|caption] )
```

We can easily decide to disregard this definition in favour of a really working one[40]:

```
redefine(
 GalleryImage:
       ImageName ("|" Caption)?
);
```

After analysing top and bottom nonterminals, we easily spot `unespaced-less-than` being bottom and `unescaped-less-than` being top—apparently, they were meant to be one, and the other one is a misspelled variation typically found in big handcrafted grammars. The same issue arises with some other nonterminals, apparently this grammar fragment was typed by someone rather careless at spelling[41]:

```
unite(unespaced-less-than, unescaped-less-than);
unite(ImageParamUpgright, ImageParamUpright);
unite(ImageValignParameter, ImageVAlignParameter);
```

3.9 Fundamental elements

Surprisingly for those who did not look at the text of the Inline Text part, the Fundamental Elements [44] does not contain any new grammar productions for us, because all of them were encountered within the Inline Text, slightly reordered.

[40] Part of `remove-extension-points.xbgf`.
[41] Part of `fix-names.xbgf`.

4 Conclusion

This section contains the list of imperfections found in the MediaWiki grammar definition. In the parenthesis we refer to the section in the text that unveils the problem or explains it.

- Non-extended Backus-Naur form was claimed to be used (§2)
- Three different metalanguages used for parts of the grammar (§3.3, §3.4, §3.7)
- Bulleted-list notation for alternatives is used, both untraditional and inconsistent with other grammar fragments (§3.8)
- Atypical metasymbols used:
  - "`...?`" (§3.2) — not defined, assumed to be an extension point
  - "`(?=EOF)`" (§3.4) — defined in terms of lookahead symbols
  - "`(`" and "`)`" (§3.4) — unexpectedly used to denote optionality
  - "`[`" and "`]`" (§3.4) — unexpectedly used for grouping
  - "`+`" (§3.6) — not defined, assumed to be a plus repetition
  - "`?`" (§3.6) — not defined, assumed to be a postfix optionality
  - "`?()`" (§3.6) — not defined, assumed to be a lookahead assertion
  - "`...`" (§3.7, §3.8) — omissions due to the lack of knowledge
  - "`...`" (§3.8) — omissions to denote values from the range of alternatives
  - "`(*`" and "`*)`" (§3.8) — start and end comment symbols
- An undesirable omission/extension point metasymbol was used (§3.2, §3.7, §3.8)
- An undesirable exception metasymbol was used (§3.3)
- An attempt to use metasyntax to distinguish between two choice semantics (§3.3)
- "Yaccified" productions with explicit right recursion (§3.2, §3.8)
- Underused metalanguage functionality: obfuscated "plus" repetitions and separator lists (§3.3)
- Misspelled nonterminal names w.r.t. case: `WhiteSpaces` vs. `Whitespaces` (§3.3), `InlineText` vs. `inline-text` (§3.7, §3.8), etc
- Mistyped nonterminal names: `unespaced-less-than` vs. `unescaped-less-than` and `ImageParamUpgright` vs. `ImageParamUpright` (§3.8)

- Varying grammar fragment delimiters (§3.4, §3.6)
- Not marking terminals explicitly with the chosen notation (§3.6)
- Not marking nonterminals explicitly with the chosen notation (§3.6)
- Escaping special characters with HTML entities (§3.4)
- Usage of "regexp format" to specify title legal characters (§3.5)
- Insufficient and excessive escaping within "regexp format" (§3.5)
- Misleading nonterminal symbol name: plural name for a single character (§3.5, §3.8)
- Improper omission of the end nonterminal metasymbol (§3.5)
- Natural language (examples given in parenthesis) as a part of a BNF production (§3.5)
- Inherently ambiguous constructs like $a?^{+}$ and $a*?$ (§3.4, §3.5, §3.7)
- Excessive bracketing (§3.3, §3.7)
- Unintentionally undefined nonterminals (§3.4)
- Referencing PHP variables like `Sanitizer::$wgHtmlEntities` and configuration functions like `mw("img_thumbnail")` (§3.8, §5.6)

5 Finishing touches

Table 1 shows the progress of several grammar metrics during recovery: TERM is the number of unique terminal symbols used in the grammar, VAR is the number of nonterminals defined or referenced there, PROD is the number of grammar production rules (counting each top alternative in them) [31]. We have already discussed bottom and top nonterminals from [36, 58, 59] earlier in §3.3. It is known and intuitively understood that high numbers of top and bottom nonterminals indicate unconnected grammar. In the ideal grammar, only few top nonterminals exist (preferably just one, which is the start symbol) and only few bottoms (only those that need to be defined elsewhere—lexically or in another language) [36]. Thus, our finishing touches mostly involved inspection of the tops and bottoms and their elimination. The very last step called "subgrammar" in Table 1 extracted only the desired start symbol (`wiki-page`) and all nonterminals reachable from its definition.

Using the terminology of [36], in this section we move from a level 1 grammar (i.e., raw extracted one) to a level 2 grammar (i.e., maximally connected one).

	TERM	VAR	PROD	Bottom	Top
After extraction	304	188	691	78	29
After `utilise-repetition.xbgf`	304	188	691	78	29
After `remove-concatenation.xbgf`	304	188	691	78	29
After `remove-extension-points.xbgf`	304	188	684	73	29
After `remove-php-legacy.xbgf`	302	188	684	70	29
After `deyaccify.xbgf`	302	187	680	70	29
After `remove-comments.xbgf`	300	187	680	68	29
After `remove-lookahead.xbgf`	300	184	680	66	29
After `remove-duplicates.xbgf`	300	183	678	66	29
After `dehtmlify.xbgf`	299	183	678	66	29
After `utilise-question.xbgf`	299	183	678	66	29
After `fix-markup.xbgf`	299	183	678	64	29
After `define-special-symbols.xbgf`	299	183	678	62	29
After `fake-exclusion.xbgf`	299	183	678	58	26
After `remove-postfix-case.xbgf`	299	183	678	57	26
After `fix-names.xbgf`	307	182	681	37	14
After `unify-whitespace.xbgf`	307	181	681	31	13
After `connect-grammar.xbgf`	307	181	671	16	7
After `refactor-repetition.xbgf`	307	181	671	16	7
After `define-lexicals.xbgf`	310	187	671	9	7
After subgrammar	310	177	664	8	1

Table 1: Simple metrics computed on grammars during transformation.

5.1 Defining special nonterminals

There is a range of nonterminals used in the MediaWiki grammar that have noticeably specific names (starting and ending with a question sign or being uppercased): they are not defined by the grammar, but usually the text around their definition is enough for a human reader to derive the intended semantics and then to specify lacking grammar productions. We also unify the naming convention while doing so (the final steps of that unification will be present in §5.5) and leave some nonterminals undefined (bottom) to serve connection points to other languages (more of that in §5.6)[42]:

```
vertical( in TableCellParameter );
removeV(
 TableCellParameter:
       ?HTML cell attributes ?
);
addV(
 TableCellParameter:
       html-cell-attributes
);
horizontal( in TableCellParameter );
vertical( in TableParameters );
removeV(
 TableParameters:
       ?HTML table attributes ?
);
addV(
```

[42] Complete listing of `define-special-symbols.xbgf`.

```
 TableParameters:
       html-table-attributes
);
horizontal( in TableParameters );
define(
 FROM_LANGUAGE_FILE:
       "#redirect"
);
inline(FROM_LANGUAGE_FILE);
define(
 STRING_FROM_DB:
       "Wikipedia"
);
inline(STRING_FROM_DB);
define(
 STRING_FROM_CONFIG:
       STR
);
inline(STRING_FROM_CONFIG);
define(
 NS_CATEGORY:
       "Category"
);
inline(NS_CATEGORY);
define(
 ALLOWED_PROTOCOL_FROM_CONFIG:
       "http://"
       "https://"
       "ftp://"
       "ftps://"
       "mailto:"
);
inline(ALLOWED_PROTOCOL_FROM_CONFIG);
unite(LEGAL_ARTICLE_ENTITY, article-title);
```

5.2 Unification of whitespace and lexicals

Another big metacategory of nonterminal symbols represent the lexical part, which is not always properly specified by a syntactic grammar. In the MediaWiki grammar case, there were several attempts to cover all lexical peculiarities including problems arising from using Unicode (i.e., different types of spaces and newlines), so the least we can do is to unify those attempts. Future work on deriving a level 3 grammar from the result of this project, will use test-driven correction to complete the lexical part correctly [36]. Our current goal is to provide a high quality level 2 grammar without destroying too much information that can be reused later[43]:

```
unite(?_variants_of_spaces_?, space);
unite(?_carriage_return_and_line_feed_?, newline);
unite(?_carriage_return_?, CR);
unite(?_line_feed_?, LF);
inline(NewLine);
```

[43] Part of `unify-whitespace.xbgf`.

```
unfold(newline in Whitespaces);
fold(newline in Whitespaces);
unite(?_tab_?, TAB);
```

Another specificity is only referenced but not defined directly by the grammar. According to the text of Inline Text section [48], this is a patch for dealing with French punctuation. It is highly debatable whether such specificity should be found in the baseline grammar, but since it is not defined properly anyway, we decide to root it out[44]:

```
vertical( in text-with-formatting );
removeV(
 text-with-formatting:
       open-guillemet
);
removeV(
 text-with-formatting:
       close-guillemet
);
horizontal( in text-with-formatting );
```

Some bottom lexical nonterminals are trivially defined in BGF[45]:

```
define(
 TAB:
       "\t"
);
define(
 CR:
       "\r"
);
define(
 LF:
       "\n"
);
define(
 any-text:
       unicode-character*
);
define(
 sort-key:
       any-text
);
define(
 any-supported-unicode-character:
       ANY
);
```

5.3 Connecting the grammar

The Magic Links part (see 3.6) apparently referenced some nonterminals that were never used. We can easily pinpoint them with a simple grammar analysis

[44] Part of `unify-whitespace.xbgf`.
[45] Part of `define-lexicals.xbgf`.

showing bottom nonterminals, and after that program the appropriate transformations[46]:

```
define(
 digits:
       digit+
);
unite(digit, decimal-digit);
unite(DIGIT, decimal-digit);
```

Undefined nonterminals `PositiveInteger` and `PositiveNumber` both can be merged with this new nonterminal[47]:

```
unite(PositiveInteger, digits);
unite(PositiveNumber, digits);
```

Nonterminal `newlines` defined at [48] and [44], is also never used and can be eliminated[48]:

```
eliminate(newlines);
```

Last connecting steps are easy since there are not that many top and bottom nonterminals left, and a simple human inspection can show that some of them are actually misspelled pairs like this one[49]:

```
unite(ImageModeThumb, image-mode-auto-thumb);
unite(category, category-link);
```

In Links section [52] there is a discussion on whether there should be a syntactic category for all links (i.e., internal and external). The discussion seems to be unfinished, with the nonterminal `link` specified, but unused (i.e., top). Since the definition is already available, we decided to use it by folding wherever possible[50]:

```
fold(link);
```

5.4 Mark exclusion

BGF does not have a metaconstruct for exclusion ("`a` should be parseable as `b` but not as `c`", mostly specified as "`<a> ::= <b> - <c>`" within the MediaWiki grammar), but we still want to preserve the information for further refactoring. One of the ways to do so is to used a marking construct usually found in parameters to transformation operators such as **project** or **addH**[51]:

```
replace(
 ?_all_supported_Unicode_characters_?_-_Whitespaces,
 ⟨(any-supported-unicode-character Whitespaces)⟩);
replace(
```

[46] Part of `connect-grammar.xbgf`.
[47] Part of `connect-grammar.xbgf`.
[48] Part of `connect-grammar.xbgf`.
[49] Part of `connect-grammar.xbgf`.
[50] Part of `connect-grammar.xbgf`.
[51] Part of `fake-exclusion.xbgf`.

```
 UnicodeCharacter_-_WikiMarkupCharacters,
 ⟨(UnicodeCharacter WikiMarkupCharacters)⟩);
replace(
 SectionLinkCharacter_- "=",
 ⟨(SectionLinkCharacter "=")⟩);
replace(
 UnicodeCharacter_- "]",
 ⟨(UnicodeCharacter "]")⟩);
replace(
 UnicodeCharacter_-_BadTitleCharacters,
 ⟨(UnicodeCharacter BadTitleCharacters)⟩);
replace(
 UnicodeCharacter_-_BadSectionLinkCharacters,
 ⟨(UnicodeCharacter BadSectionLinkCharacters)⟩);
```

5.5 Naming convention

There are three basic problems with the naming convention if we look at the whole extracted grammar, namely:

Unintelligible nonterminal names. When looking at a particular grammar production rule situated close to a piece of text explaining all kinds of details that did not fit in the BNF, it is easy to overlook non-informative names. In the case of MediaWiki, in the final grammar we have bottom nonterminals with the names like `FROM_LANGUAGE_FILE`, `STRING_FROM_CONFIG`, `STRING_FROM_DB`. Such names do not belong in the grammar, because they obfuscate it, and the main reason for having a grammar printed out in an EBNF-like form in the first place is to make it readable for a human.

Letters capitalisation. Nonterminal names can be always written in lowercase, or in uppercase, or in any mixture of them. The choice of parsing technology can influence that choice: for instance, Rascal [26] can only process capitalised nonterminal names and ANTLR [54] treats uppercase nonterminals and non-uppercase ones differently. These implicit semantic details need to be acknowledged and accounted for, in a consistent manner, which was not the case in the MediaWiki grammar.

Word separation. Most of the nonterminals have names that consist of several natural words (e.g., "wiki" and "page"). There are several ways to separate them: by straightforward concatenating ("wikipage"), by camelcasing ("WikiPage" or "wikiPage"), by hyphenating ("wiki-page"), by allowing spaces in nonterminal names ("wiki page"), etc. It does not matter too much which convention is used, as long as it is the same throughout the whole grammar. In the case of MediaWiki there is no consistency, which leads to not only decreased readability, but also to problems like `noparse-block` being defined in [51] and `noparseblock` being used in [48] (they were obviously meant to be one nonterminal).

The complete transformation script enforcing a consistent naming convention and fixing related problems on the way, looks like this[52]:

```
unite(noparseblock, noparse-block);
unite(GalleryBlock, gallery-block);
unite(ImageInline, image-inline);
unite(MediaInline, media-inline);
unite(Table, table);
unite(Text, text);
unite(InlineText, inline-text);
unite(Pipe, pipe);
renameN(AnyText, any-text);
renameN(BadSectionLinkCharacters, bad-section-link-characters);
renameN(BadTitleCharacters, bad-title-characters);
renameN(Caption, caption);
renameN(GalleryImage, gallery-image);
renameN(ImageAlignCenter, image-align-center);
renameN(ImageAlignLeft, image-align-left);
renameN(ImageAlignNone, image-align-none);
renameN(ImageAlignParameter, image-align-parameter);
renameN(ImageAlignRight, image-align-right);
renameN(ImageExtension, image-extension);
renameN(ImageModeAutoThumb, image-mode-auto-thumb);
renameN(ImageModeFrame, image-mode-frame);
renameN(ImageModeFrameless, image-mode-frameless);
renameN(ImageModeManualThumb, image-mode-manual-thumb);
renameN(ImageModeParameter, image-mode-parameter);
renameN(ImageName, image-name);
renameN(ImageOption, image-option);
renameN(ImageOtherParameter, image-other-parameter);
renameN(ImageParamBorder, image-param-border);
renameN(ImageParamPage, image-param-page);
renameN(ImageParamUpright, image-param-upright);
renameN(ImageSizeParameter, image-size-parameter);
renameN(ImageValignBaseline, image-valign-baseline);
renameN(ImageValignBottom, image-valign-bottom);
renameN(ImageValignMiddle, image-valign-middle);
renameN(ImageVAlignParameter, image-valign-parameter);
renameN(ImageValignSub, image-valign-sub);
renameN(ImageValignSuper, image-valign-super);
renameN(ImageValignTextBottom, image-valign-text-bottom);
renameN(ImageValignTextTop, image-valign-text-top);
renameN(ImageValignTop, image-valign-top);
renameN(Line, line);
renameN(LinkTitle, link-title);
renameN(MediaExtension, media-extension);
renameN(PageName, page-name);
renameN(PageNameLink, page-name-link);
renameN(PlainText, plain-text);
renameN(SectionLink, section-link);
renameN(SectionLinkCharacter, section-link-character);
renameN(SectionTitle, section-title);
renameN(TableCellParameters, table-cell-parameters);
renameN(TableColumn, table-column);
renameN(TableColumnLine, table-column-line);
renameN(TableColumnMultiLine, table-column-multiline);
```

[52] Part of `fix-names.xbgf`.

```
renameN(TableFirstRow, table-first-row);
renameN(TableParameters, table-parameters);
renameN(TableRow, table-row);
renameN(TitleCharacter, title-character);
renameN(UnicodeCharacter, unicode-character);
renameN(UnicodeWiki, unicode-wiki);
renameN(WikiMarkupCharacters, wiki-markup-characters);
```

As one can see, we reinforce hyphenation in almost all places, except for nonterminals inherited from other languages (e.g., `blockquote` from HTML). The list of plain renamings was derived automatically by a Python one-liner that transformed CamelCase to dash-separated names. The XBGF engine always checks preconditions for renaming a nonterminal (i.e., the target name must be fresh), so then it was trivial to turn the non-working **renameN** calls into **unite** calls.

5.6 Embedded languages

We may recall seeing `wgHtmlEntities` undefined nonterminal being referenced in §3.8. There are more like it—in fact, at the end of our recovery project there are 8 bottom nonterminals in the grammar:

- `LEGAL_URL_ENTITY`: designates a character that is allowed in a URL; defined by the corresponding RFC [6].
- `inline-html`: was removed deliberately due to incompleteness and questionable representation; defined partially by the accompanying English text, partially by the HTML standard [57].
- `math-block`: the syntax used by the math extension to MediaWiki [67].
- `CSS`: cascading style sheets used to specify layout of tables and table cells [8].
- `html-table-attributes` and `html-cell-attributes`: also layout of tables and table cells, but in pure HTML.
- `wgHtmlEntities`: one of the HTML entities ("quot", "dagger", "auml", etc).

They are all essentially different languages that are reused here, but are not exactly a part of wiki syntax. Some wiki engines may allow for different subsets of HTML and CSS features to be used within their pages, but conceptually these limitations are import parameters, not complete definitions. For instance, we could derive a lacking grammar fragment for `wgHtmlEntities` by looking at the file `mw_sanitizer.inc` from MediaWiki distribution[53]:

[53] Available as `mediawiki.config.wiki`.

```
<wgHtmlEntities> ::= "Aacute" | "aacute" | "Acirc" | "acirc" | "acute" | "AElig"
| "aelig" | "Agrave" | "agrave" | "alefsym" | "Alpha" | "alpha" | "amp" | "and"
| "ang" | "Aring" | "aring" | "asymp" | "Atilde" | "atilde" | "Auml" | "auml"
| "bdquo" | "Beta" | "beta" | "brvbar" | "bull" | "cap" | "Ccedil" | "ccedil"
| "cedil" | "cent" | "Chi" | "chi" | "circ" | "clubs" | "cong" | "copy"
| "crarr" | "cup" | "curren" | "dagger" | "Dagger" | "darr" | "dArr" | "deg"
| "Delta" | "delta" | "diams" | "divide" | "Eacute" | "eacute" | "Ecirc"
| "ecirc" | "Egrave" | "egrave" | "empty" | "emsp" | "ensp" | "Epsilon"
| "epsilon" | "equiv" | "Eta" | "eta" | "ETH" | "eth" | "Euml" | "euml" | "euro"
| "exist" | "fnof" | "forall" | "frac12" | "frac14" | "frac34" | "frasl"
| "Gamma" | "gamma" | "ge" | "gt" | "harr" | "hArr" | "hearts" | "hellip"
| "Iacute" | "iacute" | "Icirc" | "icirc" | "iexcl" | "Igrave" | "igrave"
| "image" | "infin" | "int" | "Iota" | "iota" | "iquest" | "isin" | "Iuml"
| "iuml" | "Kappa" | "kappa" | "Lambda" | "lambda" | "lang" | "laquo" | "larr"
| "lArr" | "lceil" | "ldquo" | "le" | "lfloor" | "lowast" | "loz" | "lrm"
| "lsaquo" | "lsquo" | "lt" | "macr" | "mdash" | "micro" | "middot" | "minus"
| "Mu" | "mu" | "nabla" | "nbsp" | "ndash" | "ne" | "ni" | "not" | "notin"
| "nsub" | "Ntilde" | "ntilde" | "Nu" | "nu" | "Oacute" | "oacute" | "Ocirc"
| "ocirc" | "OElig" | "oelig" | "Ograve" | "ograve" | "oline" | "Omega"
| "omega" | "Omicron" | "omicron" | "oplus" | "or" | "ordf" | "ordm" | "Oslash"
| "oslash" | "Otilde" | "otilde" | "otimes" | "Ouml" | "ouml" | "para" | "part"
| "permil" | "perp" | "Phi" | "phi" | "Pi" | "pi" | "piv" | "plusmn" | "pound"
| "prime" | "Prime" | "prod" | "prop" | "Psi" | "psi" | "quot" | "radic"
| "rang" | "raquo" | "rarr" | "rArr" | "rceil" | "rdquo" | "real" | "reg"
| "rfloor" | "Rho" | "rho" | "rlm" | "rsaquo" | "rsquo" | "sbquo" | "Scaron"
| "scaron" | "sdot" | "sect" | "shy" | "Sigma" | "sigma" | "sigmaf" | "sim"
| "spades" | "sub" | "sube" | "sum" | "sup" | "sup1" | "sup2" | "sup3" | "supe"
| "szlig" | "Tau" | "tau" | "there4" | "Theta" | "theta" | "thetasym" | "thinsp"
| "THORN" | "thorn" | "tilde" | "times" | "trade" | "Uacute" | "uacute" | "uarr"
| "uArr" | "Ucirc" | "ucirc" | "Ugrave" | "ugrave" | "uml" | "upsih" | "Upsilon"
| "upsilon" | "Uuml" | "uuml" | "weierp" | "Xi" | "xi" | "Yacute" | "yacute"
| "yen" | "Yuml" | "yuml" | "Zeta" | "zeta" | "zwj" | "zwnj"
```

These are 252 entities taken from the DTD of HTML 4.0 [57]. XHTML 1.0 defines an additional entity called "apos" [2], which, technically speaking, can be handled by MediaWiki since in its current state it rewrites wikitext to XHTML 1.0 Transitional. Whether it is the grammar's role to report an error when it is used, remains an open question. Furthermore, suppose we are developing wikiware which is not a WYSIWYG editor, but a migration tool or an analysis tool: this would mean that the details about all particular entities are of little importance, and one could define an entity name to be just any alphanumeric word. Questions like these arise when languages are combined, and for this particular project we leave the bottom nonterminals that represent import points, undefined.

6 Results and future work

This document has reported on a successful grammar recovery effort. The input for this project was a community-created MediaWiki grammar manually extracted from the PHP tool that is used to transform wiki text to HTML. This grammar contained unconnected fragments in at least five different notations, bearing various kinds of errors from conceptual underuse of base notation to simple misspellings, rendering the grammar fairly useless. As an output we provide a level 2 grammar, ready to be connected to adjacent modules (grammars of HTML, CSS, etc) and made into a higher level grammar (e.g., test it on a real wiki code). Naturally, this effort is one step in a long way, and we take the rest of the report to sketch the next milestones and planned deliverables:

Fix grammar fragments. The first thing we can do is regenerate the original grammar fragments in the same notation. One one hand, this would help to not alienate the grammar from its creators; on the other hand, the fragments will use a consistent notation throughout the grammar and be validated as not having any misspellings, metasymbol omissions, etc.

Derive several versions. Just in case the same MediaWiki grammar is needed in several different notations (e.g., BNF and EBNF), we can derive them from the baseline grammar with either inferred or programmable grammar transformation.

Propose a better notation. Whether or not the pure BNF grammar is delivered to Wikimedia Foundation, it will be of limited use to most people. ANTLR notation that Wiki Creole used, is more useful, but even less easy to comprehend. Both more readable and more expressive variants of grammar definition formalisms exist and can be advised for use based on the required functionality.

Find ambiguities and other problems. Various grammar analysis techniques referenced in the text above can be used to perform deeper analyses on the grammar in order to make it fully operational in Rascal, resolve existing ambiguities, and perhaps even spot problems that are unavoidable with the current notation.

Complete the lexical part. Some lexical definitions were already found in the source grammar, and were mostly preserved through the recovery process. A level 3 grammar can be derived from our current result by reinspecting these definitions together with textual annotations found on MediaWiki.org.

References

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

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

[3] 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.

[4] H. J. S. Basten. Tracking Down the Origins of Ambiguity in Context-free Grammars. In *Proceedings of the 7th International colloquium conference on Theoretical Aspects of Computing*, ICTAC'10, pages 76–90, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] S. Bennett. Markup spec: ANTLR, draft. http://www.mediawiki.org/wiki/Markup_spec/ANTLR/draft, 2008.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. *RFC 3986. Uniform Resource Identifier (URI): Generic Syntax*, January 2005. Available at http://tools.ietf.org/html/rfc3986. Accessed in July 2011.

[7] D. Binkley. Source Code Analysis: A Road Map. In *International Conference on Software Engineering*, pages 104–119, 2007.

[8] B. Bos, T. Çelik, I. Hickson, and H. W. Lie. Cascading Style Sheets, Level 2 Revision 1. CSS 2.1 Specification. *W3C Working Draft*, 6 November 2006. www.w3.org/TR/2006/WD-CSS21-20061106.

[9] N. Chomsky. Three Models for the Description of Language. *IRE Transactions on Information Theory*, 2(2):113–123, 1956.

[10] Content Creation Wiki. Wiki Engines, Accessed in July 2011. http://c2.com/cgi/wiki?WikiEngines.

[11] J. R. Cordy. Comprehending Reality — Practical Barriers to Industrial Adoption of Software Maintenance Automation. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, IWPC'03, pages 196–205, Washington, DC, USA, 2003. IEEE Computer Society.

[12] R. Cox. Regular Expression Matching Can Be Simple And Fast (but is slow in Java, Perl, PHP, Python, Ruby, ...). Technical report, swtch, Jan. 2007.

[13] T. R. Dean, J. R. Cordy, A. J. Malton, and K. A. Schneider. Grammar Programming in TXL. In *Proceedings of the 2nd IEEE International Workshop on Source Code Analysis and Manipulation*, SCAM'02. IEEE, 2002.

[14] J.-M. Favre. Software Language Engineering, Software Linguistics, 22 February 2006. http://megaplanet.org/jean-marie-favre.

[15] J.-M. Favre and T. NGuyen. Towards a Megamodel to Model Software Evolution through Transformations. *Electronic Notes in Theoretical Computer Science, Proceedings of the SETra Workshop*, 127(3), 2004.

[16] B. Fischer, R. Lämmel, and V. Zaytsev. Comparison of Context-free Grammars Based on Parsing Generated Test Data. In U. Assmann, J. Saraiva, and A. Sloane, editors, *Pre-proceedings of the Fourth International Conference on Software Language Engineering*, SLE'11, pages 323–342. Centro de Ciências e Tecnologias de Computação, July 2011. Available via http://slps.sf.net/testmatch.

[17] B. Ford. Parsing Expression Grammars: a Recognition-Based Syntactic Foundation. In *Proceedings of the Symposium on Principles of Programming Languages*, POPL'04, January 2004.

[18] J. Gosling, B. Joy, G. L. Steele, and G. Bracha. *The Java Language Specification*. Addison-Wesley, third edition, 2005. Available at java.sun.com/docs/books/jls.

[19] M. Hennessy and J. F. Power. Analysing the Effectiveness of Rule-coverage as a Reduction Criterion for Test Suites of Grammar-based Software. *Empirical Software Engineering*, 13:343–368, August 2008.

[20] ISO/IEC 14977:1996(E). *Information Technology—Syntactic Metalanguage—Extended BNF*. Available at http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

[21] S. C. Johnson. *YACC—Yet Another Compiler Compiler*. Computer Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[22] M. Junghans, D. Riehle, R. Gurram, M. Kaiser, M. Lopes, and U. Yalcinalp. An EBNF grammar for Wiki Creole 1.0. *SIGWEB Newsletter*, 2007, December 2007.

[23] A. Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 1 edition, 2008.

[24] P. Klint. A Meta-Environment for Generating Programming Environments. *ACM Transactions on Software Engineering and Methodology* (TOSEM), 2(2):176–201, 1993.

[25] P. Klint, R. Lämmel, and C. Verhoef. Toward an Engineering Discipline for Grammarware. *ACM Transactions on Software Engineering Methodology* (TOSEM), 14(3):331–380, 2005.

[26] P. Klint, T. van der Storm, and J. Vinju. EASY Meta-programming with Rascal. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *Post-proceedings of the 3rd International Summer School on Generative and Transformational Techniques in Software Engineering* (GTTSE'09), volume 6491 of *Lecture Notes in Computer Science*, pages 222–289. Springer-Verlag, January 2011.

[27] S. Klusener and R. Lämmel. Deriving Tolerant Grammars from a Baseline Grammar. In *Proceedings of the International Conference on Software Maintenance*, ICSM'03, pages 179–188, Los Alamitos, CA, USA, Sept. 2003. IEEE Computer Society.

[28] S. Klusener and V. Zaytsev. Language Standardization Needs Grammarware. JTC1/SC22 Document N3977, ISO/IEC, 2005. Available via http://www.open-std.org/jtc1/sc22/open/n3977.pdf.

[29] D. E. Knuth. The Genesis of Attribute Grammars. In P. Deransart and M. Jourdan, editors, *WAGA*, volume 461 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1990.

[30] J. Kort, R. Lämmel, and C. Verhoef. The Grammar Deployment Kit. In M. G. J. van den Brand and R. Lämmel, editors, *Electronic Notes in Theoretical Computer Science*, volume 65. Elsevier Science Publishers, 2002.

[31] N. A. Kraft, E. B. Duffy, and B. A. Malloy. Grammar Recovery from Parse Trees and Metrics-Guided Grammar Refactoring. *IEEE Transactions on Software Engineering*, 99(RapidPosts):780–794, 2009.

[32] R. Lämmel. Grammar Adaptation. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, volume 2021 of *Lecture Notes in Computer Science*, pages 550–570. Springer-Verlag, 2001. Available at http://homepages.cwi.nl/~ralf/fme01.

[33] R. Lämmel. Grammar Testing. In H. Hussmann, editor, *Proceedings, Fundamental Approaches to Software Engineering* (FASE'01), volume 2029 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 2001.

[34] R. Lämmel. The Amsterdam Toolkit for Language Archaeology. *Electronic Notes in Theoretical Computer Science*, 137(3):43–55, 2005. *Proceedings of the Second International Workshop on Metamodels, Schemas and Grammars for Reverse Engineering* (ATEM'04).

[35] R. Lämmel and W. Schulte. Controllable Combinatorial Coverage in Grammar-Based Testing. In U. Uyar, M. Fecko, and A. Duale, editors, *Proceedings of the 18th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems (TestCom'06)*, volume 3964 of *Lecture Notes in Computer Science*, pages 19–38. Springer Verlag, 2006.

[36] R. Lämmel and C. Verhoef. Semi-automatic Grammar Recovery. *Software—Practice & Experience*, 31(15):1395–1438, Dec. 2001. Available via http://www.cs.vu.nl/grammarware/ge/.

[37] R. Lämmel and V. Zaytsev. An Introduction to Grammar Convergence. In *Proceedings of the 7th International Conference on Integrated Formal Methods* (iFM 2009), volume 5423 of *Lecture Notes in Computer Science*, pages 246–260. Springer, February 2009.

[38] R. Lämmel and V. Zaytsev. Recovering Grammar Relationships for the Java Language Specification. In *Proceedings of the 9th IEEE International Working Conference on Source Code Analysis and Manipulation*, SCAM'09, pages 178–186, Edmonton, Canada, September 2009. IEEE.

[39] R. Lämmel and V. Zaytsev. Recovering Grammar Relationships for the Java Language Specification. *Software Quality Journal*, 19(2):333–378, March 2011. Also available at http://arxiv.org/abs/1008.4188.

[40] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[41] B. Leuf and W. Cunningham. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley Longman, Amsterdam, pap/cdr edition, 2001.

[42] B. A. Malloy and J. F. Power. An Interpretation of Purdom's Algorithm for Automatic Generation of Test Cases. In *In 1st Annual International Conference on Computer and Information Science*, pages 3–5, 2001.

[43] P. Maurer. Generating Test Data with Enhanced Context-free Grammars. *IEEE Software*, 7(4):50–56, 1990.

[44] MediaWiki. Markup spec: BNF: Fundamental elements. http://www.mediawiki.org/wiki/Markup_spec/BNF/Fundamental_elements, 2008. Revision 212918 of 17 September 2008 was used for extraction.

[45] MediaWiki. Markup spec: BNF. http://www.mediawiki.org/wiki/Markup_spec/BNF, 2009.

[46] MediaWiki. Markup spec: BNF: Article. http://www.mediawiki.org/wiki/Markup_spec/BNF/Article, 2009. Revision 281674 of 17 October 2009 was used for extraction.

[47] MediaWiki. Markup spec: BNF: Article title. http://www.mediawiki.org/wiki/Markup_spec/BNF/Article_title, 2009. Revision 295042 of 28 December 2009 was used for extraction.

[48] MediaWiki. Markup spec: BNF: Inline text. http://www.mediawiki.org/wiki/Markup_spec/BNF/Inline_text, 2009. Revision 295055 of 29 December 2009 was used for extraction.

[49] MediaWiki. Markup spec: BNF: Magic links. http://www.mediawiki.org/wiki/Markup_spec/BNF/Magic_links, 2009. Revision 269783 of 8 August 2009 was used for extraction.

[50] MediaWiki. Markup spec: BNF: Special block. http://www.mediawiki.org/wiki/Markup_spec/BNF/Special_block, 2009. Revision 281676 of 17 October 2009 was used for extraction.

[51] MediaWiki. Markup spec: BNF: Noparse block. http://www.mediawiki.org/wiki/Markup_spec/BNF/Noparse-block, 2010. Revision 372814 of 30 December 2010 was used for extraction.

[52] MediaWiki. Markup spec: BNF: Links. http://www.mediawiki.org/wiki/Markup_spec/BNF/Links, 2011. Revision 376721 of 18 January 2011 was used for extraction.

[53] MediaWiki. Parser.php, Accessed in July 2011. http://svn.mediawiki.org/viewvc/mediawiki/trunk/phase3/includes/parser/Parser.php?view=markup.

[54] T. Parr. ANTLR—ANother Tool for Language Recognition. http://antlr.org, 2008.

[55] J. F. Power and B. A. Malloy. A metrics suite for grammar-based software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16:405–426, November 2004.

[56] P. Purdom. A Sentence Generator for Testing Parsers. *BIT*, 12(3):366–375, 1972.

[57] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.01 Specification. *W3C Recommendation*, 24 December 1999. http://www.w3.org/TR/html401/.

[58] M. P. A. Sellink and C. Verhoef. Generation of Software Renovation Factories from Compilers. In *Proceedings of 15th International Conference on Software Maintenance*, ICSM'99, pages 245–255, 1999. Available at http://www.cs.vu.nl/~x/com.

[59] M. P. A. Sellink and C. Verhoef. Development, Assessment, and Reengineering of Language Descriptions. In J. Ebert and C. Verhoef, editors, *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering*, CSMR'00, pages 151–160. IEEE Computer Society Press, March 2000. Available at http://www.cs.vu.nl/~x/cale.

[60] E. G. Sirer and B. N. Bershad. Using Production Grammars in Software Testing. *SIGPLAN Notices*, 35:1–13, December 1999.

[61] M. van den Brand, A. Sellink, and C. Verhoef. Generation of Components for Software Renovation Factories from Context-free Grammars. *Science of Computer Programming*, 36(2–3):209–266, Mar. 2000.

[62] M. G. J. van den Brand, M. P. A. Sellink, and C. Verhoef. Obtaining a COBOL Grammar from Legacy Code for Reengineering Purposes. In M. P. A. Sellink, editor, *Proceedings of the 2nd International Workshop on the Theory and Practice of Algebraic Specifications*, Berlin, 1997. Springer-Verlag.

[63] A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, C. H. A. Koster, M. Sintzoff, C. Lindsey, L. G. L. T. Meertens, and R. G. Fisker. Final Report on the Algorithmic Language Algol 68. Technical report, IFIP Working Group 2.1, Dec. 1968.

[64] M. Črepinšek, M. Mernik, and V. Žumer. Extracting Grammar from Programs: Brute Force Approach. *SIGPLAN Notices*, 40:29–38, Apr. 2005.

[65] N. Veerman. *Automated Mass Maintenance of Software Assets*. PhD thesis, Vrije Universiteit, Jan. 2007. Available at www.cs.vu.nl/~nveerman/research/thesis/thesis.pdf.

[66] M. Völter. Embedded software development with projectional language workbenches. In D. C. Petriu, N. Rouquette, and Ø. Haugen, editors, *MoDELS (2)*, volume 6395 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2010.

[67] T. Wegrzanowski and B. Vibber. Math, Last version in April 2011. http://www.mediawiki.org/wiki/Extension:Math.

[68] Wikidot. Free and Pro Wiki Hosting, 2007–2011. http://www.wikidot.com.

[69] Wikimedia Foundation Project. Parser plan, 2011. http://www.mediawiki.org/wiki/Future/Parser_plan.

[70] M. Wimmer and G. Kramler. Bridging Grammarware and Modelware. In *Proceedings of the 4th Workshop in Software Model Engineering*, WiSME'05, Oct. 2005. Available at www.big.tuwien.ac.at/research/publications/2005/1105.pdf.

[71] N. Wirth. What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions? *Communications of the ACM*, 20(11):822–823, 1977.

[72] V. Zaytsev. Combinatorial Test Set Generation: Concepts, Implementation, Case Study. Master's thesis, Universiteit Twente, Enschede, The Netherlands, June 2004.

[73] V. Zaytsev. Correct C# Grammar too Sharp for ISO. In *Pre-proceedings of the International Summer School on Generative and Transformational Techniques in Software Engineering, Part II, Participants Workshop*, GTTSE'05, pages 154–155, Braga, Portugal, July 2005. Technical Report, TR-CCTC/DI-36, Universidade do Minho. Extended abstract.

[74] V. Zaytsev. *XBGF Reference Manual: BGF Transformation Operator Suite*, 1.0 edition, August 2009. Available via http://slps.sf.net/xbgf.

[75] V. Zaytsev. *Recovery, Convergence and Documentation of Languages*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, October 2010. Available at http://grammarware.net/text/2010/zaytsev-thesis.pdf.

[76] V. Zaytsev. Browsable MediaWiki in EBNF Grammar. http://slps.sf.net/zoo/wiki/mediawiki-ebnf.html, 2011.

[77] V. Zaytsev. Language Convergence Infrastructure. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *Post-proceedings of the 3rd International Summer School on Generative and Transformational Techniques in Software Engineering* (GTTSE'09), volume 6491 of *Lecture Notes in Computer Science*, pages 481–497. Springer-Verlag Berlin Heidelberg, January 2011.

[78] V. Zaytsev. Wiki Migration. http://wikimania2011.wikimedia.org/wiki/Submissions/Wiki_Migration, 2011. Presentation accepted at Wikimania 2011.

[79] V. Zaytsev and R. Lämmel. A Unified Format for Language Documents. In B. Malloy, S. Staab, and M. G. J. van den Brand, editors, *Post-proceedings of the Third International Conference on Software Language Engineering* (SLE'10), volume 6563 of *Lecture Notes in Computer Science*, pages 206–225. Springer, Heidelberg, January 2011.

[80] V. Zaytsev, R. Lämmel, and T. van der Storm. Software Language Processing Suite, 2008–2011. http://slps.sf.net, repository statistics on July 2011: 727 commits by Zaytsev, 304 commits by Lämmel, 44 commits by van der Storm.