The Grammar Hammer of 2012*

Vadim Zaytsev, vadim@grammarware.net

Software Analysis and Transformation (SWAT) Team
Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands

December 19, 2012

Contents

*The title relates both to the folklore story of a steel driving man named John Henry dying with a hammer in his hand instead of losing to a steam drill [Nel08] and to a psychologist Abraham Maslow stating that if the only tool you have is a hammer, it is tempting to treat everything as if it were a nail [Mas62].

1 Introduction

The purpose of this report is documenting personal research results of the year 2012 in a form primarily intended for assessment of their scientific merit as a foundation for future work, not for quantitative assessment of the resulting publication record. This can be considered as an aggressive form of self-archiving initiative [Har01] where scientific and engineering contributions are not only logged, but also put in perspective by a separate first class atomic scientific knowledge object. This report is mostly meant for my SWAT colleagues. However, it is open for broad audience and meant to be readable by any researcher with reasonable degree of familiarity with computer science. It can be consumed as a self-contained document, but many details are not pulled in from available referenced sources.

We start right away with a the overview of the field (§2.1) followed by brief descriptions of major (§2.2) and minor (§2.3) contributions, followed by a more elaborate motivation for creation of this document (§2.4). Next, all research topics are laid out in detail one by one (§3). For the sake of complexity, a separate overview of all involved venues (§4) is included. §5 concludes the report.

2 Preliminaries

2.1 Background notions

Software language is a concept that generalises over programming languages, markup languages, database schemata, data structures, abstract data types, data types, modelling languages, ontologies, etc. Whenever we observe some degree of *commitment to structure*, we can identify it with a language, which elements (symbols) can be separately defined and the allowed combinations of them can be somehow specified. Studying software language engineering is important because of possibly gained insights into relations between the way such languages are defined and used in different technological spaces (e.g., we can study data binding as a way to map a relational database to an object model, or language convergence as a way to compare an XML schema with a syntax definition).

Formal grammars is a long-existing approach of dealing with languages — originally context free grammars [Cho56] were mainly aimed at textual programming languages [ASU85], but later other variants of grammars were proposed, including keyword grammars [GM77], indexed grammars [Aho68], lexicalised grammars [SAJ88], object grammars [SCL12], pattern grammars [Gre96], array grammars [SSK72], puzzle grammars [Niv+91], picture grammars [MS67], picture processing grammars [Cha70], tile grammars [RP05], grid grammars [Dre+01], motion picture grammars [BAF97], pair grammars [Pra71], triple graph grammars [Sch95], deterministic graph grammars [Cau07], string adjunct grammars [JKY69], head grammars [Pol84], tree adjunct grammars [JLT75], tree description grammars [Kal01], description tree grammars [RVSW95], description tree substitution grammars [RWVS01], functional grammars [Łuk77], Łukaszewicz universal grammars [Łuk82], two level grammars [Wij74], van Wijngaarden grammars [Wij65], metamorphosis grammars [Col78], affix grammars [Kos91], extended affix grammars [Mei90], attribute grammars [Knu90], extended attribute grammars [WM83], definite clause grammars [PW86], minimalist grammars [LR01], categorial grammars [Ajd35], type grammars [Lam58], pregroup grammars [Lam08], Montague universal grammars [Mon70], logic grammars [AD89], assumption grammars [DTL97], constraint handling grammars [Chr05], abductive logic grammars [CD09], simple transduction grammars [LS68], inversion transduction grammars [Wu97], range concatenation grammars [Bou98], island grammars [DK99], bridge grammars [NNEH09], skeleton grammars [KL03], permissive grammars [Kat+09], conjunctive grammars [Okh01], Boolean grammars [Okh04], Peirce grammars [Böt01], transformational grammars [DeR74], probabilistic grammars [Kor11], notional grammars [And91], analytic grammars [For04], parsing schemata [Sik97], cooperating string grammar systems [CV+95], cooperating array grammar systems [DFP95], cooperating puzzle grammar systems [SSC06], etc[1]. A grammar of a software language, which specifies commitment to grammatical structure, is called a *grammar in a broad sense* [KLV05a], even if in practice it defines a metamodel or an API, thus not officially being a grammar at all. The grammarware technological space is commonly perceived as mature and drained of any scientific challenge, but provides many unsolved problems for researchers who are active in that field.

For the last years, and specifically in 2012, I have focused my efforts on using grammar-based techniques in the broad field of software language engineering.

[1]The earliest possible reference is given for each variant, preferably from the programming language research field.

2.2 Major contributions in a nutshell

This section contains brief descriptions of the contributions of 2012 and some statements about their usability and/or importance. Sections that contain extended descriptions of the contributions with some level of technical detail, are referenced in parenthesis.

Guided grammar convergence (§3.1).

Grammar convergence is a lightweight verification method for establishing and maintaining the correspondence between grammar knowledge ingrained in various software artifacts [LZ09a]. The method entails programming grammar transformation steps with a general purpose grammar transformation operator suite. It was acknowledged in [Cam10, p.34] as "a product-line approach to provide [...] an organised software structure". Yet, the method had some weak sides that inspired further investigation.

One of the biggest issues is maintenance of the grammar relationships. Once they have been established by programming grammar transformation steps, it becomes very hard to coevolve these steps with eventual changes in the source grammars. An ideal solution would be a way to automatically reestablish grammar relationships based on declarative constraints. This way is *guided grammar convergence*: instead of programming the transformations, we construct an idealised "master grammar" that shows the most essential properties of all grammars that are to be converged, and the transformation steps are then derived automatically, guided by the structure of the master grammar.

The transformation inference algorithm relies on the source grammars and their metasyntax. This method was prototyped twice: in Python and in Rascal, and tested successfully on 12 grammars in a broad sense obtained from different technological spaces. It has not been properly published after being rejected three times [Zay12h; Zay12i; Zay12j], but received encouraging feedback from some of those venues and from one presentation [Zay12g].

Grammar transformation (§3.2)

Grammar convergence, evolution, maintenance and any other activity that deals with changes, can profit from expressing such changes in the functional way: every step is represented as a function application, where a function is a transformation operator such as *rename* or *add*. The latest of such operator suites has been developed in 2010 [Zay+08, XBGF Manual] and shown to be superior to its alternatives [LZ11, §4].

During 2012, XBGF has been:

- reimplemented in Rascal, which led to ex-

tensive testing and more systematic specification of operator semantics (§3.2.1);
- extended for bidirectionality by pairing operators, introducing lacking ones and abandoning unsuitable ones (§3.2.2);
- experimentally extended for adaptability (§3.2.3);
- extended by mining patterns of it usage (§3.2.4);
- investigated for migration from the functional paradigm to the declarative one (§3.2.5).

Each of these initiatives is a nontrivial project complete with conceptual motivation, programmed prototypes and obtained results (positive for the first three, controversial for the fourth and decisively negative for the last one).

Metasyntactic experiments (§3.3)
Metasyntax as a language in which grammars are specified, was a topic briefly touched in my PhD thesis [Zay10], but never officially published. In 2012, I finally dedicated enough time and attention to engineer a proper prototype for metasyntax specifications (§3.3.1) and their transformations (§3.3.2), as well as to perform a series of experiments on metasyntax-driven grammar recovery (§3.3.3) and convergence (§3.3.4). This area has now been exhaustively covered, and the only possible future extensions must rely on going way beyond textually specified context-free grammars.

To be completely frank, it should be noted here that most of the experiments with metasyntax were done in the course of 2011 and were only polished, presented and published in 2012 (which still required considerable effort).

Tolerant parsing overview (§3.4)
Just like the grammar recovery paper came with an extensive related work section which listed all grammar recovery initiatives in the last decade or two [Zay12x, §2], a new parsing algorithm that I tried to propose (§3.8.2) came with an extensive overview of all methods of tolerant parsing known to grammarware engineers up to date (§3.4). While the iterative parsing method was novel but ultimately dull and uninteresting, the overview itself was received very warmly during the presentation on it [Zay12ag]. One of the reviewers of [Zay12n] has also advised to throw away the thing I thought was the main contribution of the paper, and extend the thing I thought of as a byproduct, into a longer journal article. While surprising at first, this seems indeed like a reasonable course of action.

2.3 Selected minor contributions

In the following sections, I will present a detailed overview of major (§§3.1–3.7) and minor (§3.8) contributions, but the border between them is naturally flexible. Thus, in the previous section introduced only four of the best major ones, and this section will introduce several middleweight contributions ("less major" mixed with "not so minor" ones).

Grammar mutation (§3.8.1)
It has been noted in [Zay12p; Zay12r] that there is a separate group of grammar changes that reside between traditional grammar transformations ("rename X to Y") and the grammar transformation operators ("rename"), which was labelled as a grammar *mutation* and formalised differently from them. While the only truly important property of grammar mutation in the context of [Zay12p; Zay12r] was that they are considerably harder to bidirectionalise, a lot of useful grammar manipulations like "rename all uppercase nonterminals to lowercase" or "eliminate all nonterminals unreachable from the root" belong to the class of mutations, so it deserves to be studied closer. In [Zay12ah], I have composed a list of 16 mutations identified in already published academic papers or in publicly available grammarware source code, but the paper was not accepted, so the topic remains only marginally explored.

Iterative parsing (§3.8.2)
Starting from a fresh yet weird topic of what "the cloud" can mean for grammarware engineering, I ended up proposing an algorithm for *parsing in the cloud*, which was **not** based on parallel parsing [Alb+94], but rather on island grammars [Moo01; KL03]. The whole topic is questionable and only suitable for a "wild ideas workshop", as was nicely put by one of the reviewers, but is still potentially of some interest. The paper containing the algorithm was rejected twice [Zay12n; Zay12o] so far, and requires investing more time in empirical validation at least, in order to increase the chances of acceptance.

Unparsing in a broad sense (§3.8.3)
I could not help noticing that parsing (i.e., mapping strings to graphs) receives much more research attention than the reverse process of unparsing (i.e., mapping graphs to strings). However, the only thing I did accomplish this year was to collect a couple of references on existing research and make a "new ideas" extended abstract [Zay12ai], which was classified as a "request for discussion" and rejected. I am already prepared to give a discussion-provoking presentation on this topic, but it requires much more effort to be invested until more tangible results are obtained.

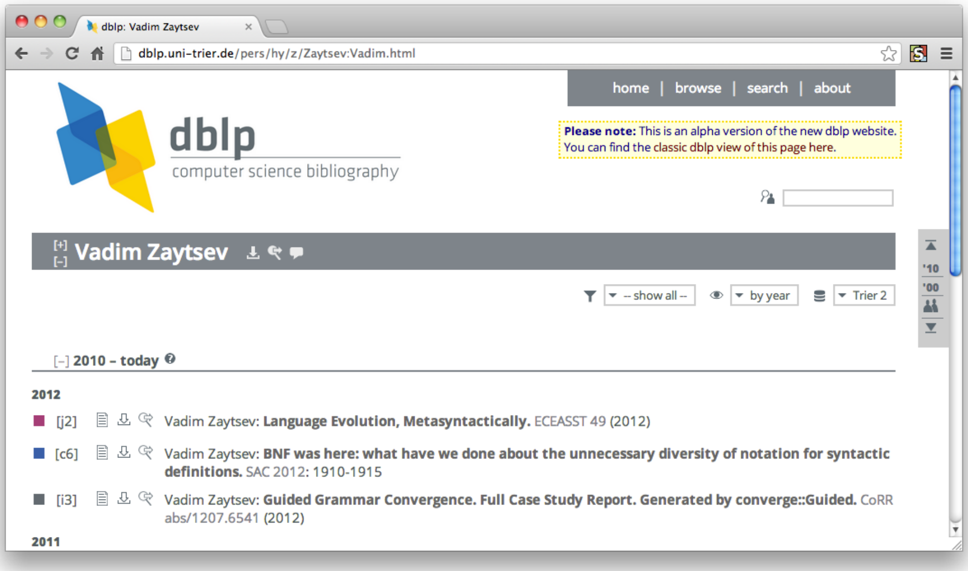

Figure 1: The results of 2012, according to DBLP.

Megamodelling (§3.5)
Megamodelling is higher abstraction level form of modelling that is concerned with software languages and technologies and relations between them. This year I have published some papers with megamodels in them [Zay12u; Zay12v; Zay12ab; Zay12ac; Zay12k] and touched upon the topic in a range of presentations [Zay12s; Zay12ad; Zay12aa; Zay12w]. Much more work on this topic is planned for 2013.

Open notebook computer science (§3.7)
Open notebook science is an open science paradigm of doing research in a transparent way. It is already a fairly widely accepted methodology in areas like chemistry [San08] and drug discovery [Sin08] and is generally perceived as the next big step after open access [Llo08]. However, in computer science and software engineering it has never been a tradition to keep a lab notebook, and it takes quite some time to maintain it, with few apparently visible benefits. I have been experimenting quite a lot with this idea, but finally decided to come out to a bigger public with two presentations in 2012 [Zay12z; Zay12af]. In general, I believe this is a reasonable idea, and I will keep practicing open notebook science myself, but it will take quite some effort to put it carefully into words in order to publish, so I am not even sure it is feasible to expect a publication in 2013.

2.4 Motivation for this report

The progress of a scientist is traditionally measured by an outsider by the papers that the scientist produces. According to DBLP, the main supplier of bibliography lists currently, the year 2012 for me yielded the following results (see the screencapture on Figure 1): one journal paper [Zay12r], one conference proceedings paper [Zay12d], one preprint [Zay12k]. However, the first one is an only slightly extended version of a workshop paper [Zay12p] written mostly in 2011; the second one was written and accepted in 2011; and the third one was intended to be a supplementary material for another paper that is not yet accepted anywhere. Additionally, there are three more post-proceedings papers in print [Zay12x; Zay12ac; Zay12v], which are already finished and submitted and will eventually appear in the ACM Digital Library — when they do, they will also be listed at DBLP under 2012, but at that time it will be too late to write a year report.

What about the self-archiving initiative [Har01]? Luckily, I disclose relatively large amounts of dark data [Goe07] about my research activities, having an extensively linked daily updated website with an open notebook (see §3.7) and many generated lists, including the current publishing progress, as seen on the the screencapture on Figure 2. Even judging by the bare numbers, one can already tell that this list contains much more information than the DBLP list. However, it also has its problems: the "published" column contains the works of previous years that happened to be delayed enough for the post-proceedings to appear in January 2012 [FLZ12]; as well as mentions of drafts planned for future publication (easily localised in the last column). It also contains editorial work for non-mainstream venues [JZ12b; JZ12a] which is of much lesser relevance because there is no scientific value to it. What it does not contain, is relations between all these papers: obviously some papers are enhanced version of previously rejected drafts, but in order to

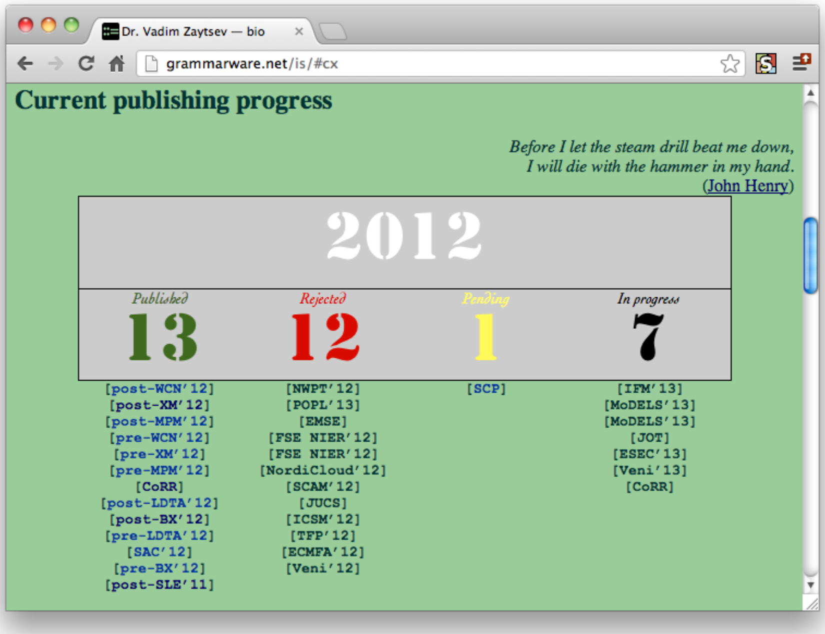

Figure 2: The results of 2012, according to the self-archiver.

figure them out, one needs to read the open notebook at `http://grammarware.net/opens` or analyse it automatically (no readily available tools are provided).

Personally, I can state that guided grammar convergence (see §3.1) is my top result of the year. However, it has not (yet) been properly published. After being rejected at ECMFA [Zay12h] and ICSM [Zay12i], it received very positive reactions from POPL [Zay12j], yet was also deemed not mature enough for publication. Still, having to figure out what are the limits of the proposed methodology and how to describe it well, does not change the fact that this is my best contribution of the year 2012.

Grammar transformation operator suites like XBGF (see §3.2.1), ΞBGF (§3.2.2), EXBGF (§3.2.4), ΔBGF (§3.2.5) and NBGF (§3.2.3) represent massive amounts of work, but they are not publishable by themselves, if at all. Still, each of them represents a milestone enabling further advances. Engineering work that supports scientific research, has rarely been explicitly noted and appreciated.

Quoting [San08]: *"The notebook is about publishing data as quickly as possible. The paper is about synthesizing knowledge from all those results."* Hence, this report is aimed at synthesizing knowledge about the experiments and achievements undertaken during the course of 2012 by me (possibly in collaboration with someone else) within the NWO project 612.001.007, "Foundations for a Grammar Laboratory". It holds the most value for myself and my project colleagues, but is also available for anyone interested in the topics discussed: unlike open notebook entries, this report is a proper atomic scientific knowledge object [Giu+10; Sim+11]. Only two topics directly relevant to the project, are not included: one must remain hidden according to the rules of the target venue, and for the other one the context and consequences are not yet understood enough even for such a lightweight presentation.

3 Topics overview

3.1 Guided grammar convergence

Let us consider two grammars in a broad sense [KLV05a]. We say that they represent one *intended* software language, if there exists a complete bidirectional mapping between language instances that commits to grammatical structure of different grammars. For example, if a parser produces parse trees that can always be converted to abstract syntax trees expected by a static analysis tool and back, it means that they represent the same intended language. As another example, consider an object model used in a tool that stores its objects in an external database (XML or relational): the existence of a bidirectional mapping between entries (trees or tables) in the database and the objects in memory, means that they represent the same intended language, even though they use very different ways to describe it. An equivalence class spawned by this definition (i.e., a set of different grammars of the same intended language) effectively forms a *grammarware product line* of products that perform different tasks on instances of the same intended language: in that sense, for example, all Java-based tools form a product line, if they agree on a language version and do not employ any highly permissive methods that would shift them into a broader class. For the sake of simplicity, let us focus on *grammar product lines*: collections of grammars of the same intended language. The relation between a grammar product line and a grammarware product line is justified by research on automated derivation of grammar-based tools like parsers, environments, documentation, formatters and renovators from grammars [Kli93; SV99; Jon02; KLV05a; Cam+10; ZL11].

Suppose that we have two grammars: one that we call a *master grammar* (a specially pre-constructed abstract grammar of the intended language) and one that we call a *servant grammar* (a grammar derived from a particular language implementation). In general, there are four phases of guided grammar convergence, and they are presented in this section in the *reverse* order. First, we consider the simplest scenario when all mismatches are of *structural* nature. Then, we move on to a more complicated situation when a *nominal* matching between sets of nonterminals is unknown. Since this is rather uncommon (most methods used in practice for imploding parse trees to abstract syntax trees, from Popart [Wil97] to Rascal [KSV11], heavily rely on equality of names), a new method for matching nonterminals has been developed. In short, it comprises construction of production signatures for each production rule in both grammars, and a search for equivalent and weakly equivalent production rules with respect to those signatures. Once a name resolution relation has been successfully built, a previously discussed structural matching can be applied. We will also discuss *normalisations* that can transform any arbitrary grammar to a form easily consumable by our nominal and structural matching algorithms. Finally, I will list additional problems that indicate *grammar design* decisions and therefore not affected by normalisations. However, I describe how to automatically detect such issues and to address them with grammar mutations.

Structural matching

Let us assume the simplest scenarios: the two input grammars have the same set of nonterminals; neither of them has terminals; the starting nonterminal is the same and that the sets of production rules are different but have the same cardinality. These would be typical circumstances if, for example, the grammars define two alternative abstract syntaxes for the same intended language.

We can start from the roots of both grammars and traverse them synchronously top-down, encountering only the following four circumstances:

Perfect match. Convergence is trivially achieved.

Nonterminal vs. value. By "values" I mean nonterminals that are built-in in the underlying framework (e.g., "string").

Production rule	Production signature
p_1=(*program* → *function*$^+$)	$\{\langle function, +\rangle\}$
p_2=(*function* → *str* *str*$^+$ *expr*)	$\{\langle expr, 1\rangle, \langle str, 1{+}\rangle\}$
p_3=(*expr* → *str*)	$\{\langle str, 1\rangle\}$
p_4=(*expr* → *int*)	$\{\langle int, 1\rangle\}$
p_5=(*expr* → *apply*)	$\{\langle apply, 1\rangle\}$
p_6=(*expr* → *binary*)	$\{\langle binary, 1\rangle\}$
p_7=(*expr* → *cond*)	$\{\langle cond, 1\rangle\}$
p_8=(*apply* → *str* *expr*$^+$)	$\{\langle expr, +\rangle, \langle str, 1\rangle\}$
p_9=(*binary* → *expr* *operator* *expr*)	$\{\langle expr, 11\rangle, \langle operator, 1\rangle\}$
p_{10}=(*cond* → *expr* *expr* *expr*)	$\{\langle expr, 111\rangle\}$

Table 1: Production rules of the master grammar for FL, with their production signatures.

Sequence element permutations can be automatically detected and converged.

Lists of symbols. Many frameworks that have components with grammatical knowledge, have a notion of a list or a repetition of symbols in their metalanguage.

It can be shown that these four are the only possibilities, and that their resolution can be resolved.

Nominal resolution

In a more complicated scenario, let us consider the case of different nonterminal sets in two input grammars, and for simplicity we assume that all production rules are vertical (non-flat) and chained (if there is more than one production rule for the same nonterminal, all of them are chain productions — i.e., have one nonterminal as their right hand side). Next, we define a *footprint* of a nonterminal in an expression as follows:

$$\pi_n(x) = \begin{cases} \{1\} & \text{if } x = n \\ \{?\} & \text{if } x = n? \\ \{+\} & \text{if } x = n^+ \\ \{*\} & \text{if } x = n^* \\ \bigcup\limits_{e\in L} \pi_n(e) & \text{if } x \text{ is a sequence } L \\ \varnothing & \text{otherwise} \end{cases}$$

By extension, we define a footprint of a nonterminal in a production rule as a footprint of it in its right hand side:

$$\pi_n(m \to e) = \pi_n(e)$$

Based on that, we define a *production signature*, or a prodsig, of a production rule, by collecting all footprints of all nonterminals encountered in its right hand side:

$$\sigma(p) = \{\langle n, \pi_n(e)\rangle \mid n \in \mathbb{N},\ \pi_n(e) \neq \varnothing\}$$

A good example of how production signatures look like, is to be found on Table 1.

We say that two production rules are *prodsig-equivalent*, if and only if there is a unique match between tuple ranges of their signatures:

$$p \doteq q \iff \forall\langle n,\pi\rangle \in \sigma(p),\ \exists!\langle m,\xi\rangle \in \sigma(q),\ \pi = \xi$$

Similarly, a weak prodsig-equivalence $p \doteqdot q$ is defined by dropping the uniqueness constraint and weakening the equality constraint in the last definition to footprint equivalence which disregards repetition kinds ($+$ is equivalent to $*$). Then it can be proven that for any two strongly prodsig-equivalent production rules p and q, $p \doteq q$, a *nominal resolution* relationship has the form of:

$$p \diamond q = \sigma(p) \circ \overline{\sigma(q)}$$

where $\rho_1 \circ \rho_2$ is a composition of two relations in the classic sense and $\overline{\rho}$ is the classic inverse of a relation. Moreover, for any two weakly prodsig-equivalent production rules p and q, $p \doteqdot q$, there is (at least one) nominal resolution relationship $p \diamond q$ that satisfies the following:

$$\forall\langle a,b\rangle \in p \diamond q : a = \omega \vee b = \omega \vee$$
$$\exists\pi, \exists\xi, \pi \approx \xi, \langle a,\pi\rangle \in \sigma(p), \langle b,\xi\rangle \in \sigma(q)$$

and

$$\forall\langle a,b\rangle \in p \diamond q, \forall\langle c,d\rangle \in p \diamond q : a = c \Rightarrow b = d$$

Where ω is used to explicitly denote unmatched nonterminals.

Abstract Normal Form

In order to fit any grammar into the conditions required by the previously described matching techniques, we demand the following normalisation:

1. lack of labels for production rules
2. lack of named subexpressions
3. lack of terminal symbols
4. maximal outward factoring of inner choices
5. lack of horizontal production rules
6. lack of separator lists
7. lack of trivially defined nonterminals (with α, ε or φ)
8. no mixing of chain and non-chain production rules
9. the nonterminal call graph is connected, and its top nonterminals are the starting symbols of the grammar

It can be shown that transforming any grammar into its Abstract Normal Form is in fact a grammar mutation (see §3.8.1). In the prototype, I have implemented it to effectively generate bidirectional grammar transformation steps, so the normalisation preserves any information that it needs to abstract from.

Grammar design mutation

Some grammar design smells (terminology per [Sto12a]) like yaccification (per [SV99; BSV98]) or layered expressions (per [LZ09a]) have shown to be persistent enough to survive all normalisations and cause problems for establishing nominal and structural mappings. They can be identified and dealt with by automated analyses and mutations, but so far I have to proof that they are the only possible obstacles, and no guarantees about any other smells problematic for guided grammar convergence.

3.1.1 Generalisation of production signatures

The method of establishing nonterminal mappings of different grammars of the same intended language, can be generalised as follows. Suppose that we have a metalanguage. Without loss of generality, let us assume that each grammar definition construct that is present in it, can be referred to by a single symbol: "**,**", "**?**", "*****", etc and uses prefix notation. This metasyntactic alphabet Λ will form the foundation of our footprints and signatures. Let us also assume that all metasymbols are unary or are encoded as unary, except for two composition constructs: a sequential "**,**" and an alternative "|", which take a list of symbols.

Then, a footprint of any nonterminal n in an expression x is a multiset of metasymbols that are used for occurrences of n within x:

$$\pi_n(x) = \begin{cases} \{1\} & \text{if } x = n \\ \{\mu\} & \text{if } x = \mu(n), \mu \in \Lambda \\ \bigcup\limits_{e \in L} \pi_n(e) & \text{if } x = \mathbf{,}(L) \\ \varnothing & \text{otherwise, also if } x = |(L) \end{cases}$$

Our previously given definition of a production signature can still be used with this generally redefined footprints.

It is well known that language equivalence is undecidable. Any formulation of the grammar equivalence problem, that is based on language equivalence, is thus also undecidable. Grammar convergence [LZ09a; LZ11] is a practically reformulated grammar equivalence problem that uses automated grammar transformation steps programmed by a human expert. By using these generalised metasyntactic signatures, we can *infer converging transformation steps* automatically, thus eliminating the weakest link of the present methodology. However, this is not the only application of the generalisation.

The most trivial use of metasyntactic footprints and signatures would lie in *grammarware metrics*. Research on software metrics applied to context-free grammars has never been an extremely popular topic, but it did receive some attention in the 1970s [Gru71], 1980s [Kel81] and even recently [PM04; Čre+10]. Using quantitative aspects of metasyntactic footprints and signatures (numbers of different footprints within the grammar, statistics on them, etc) is possible and conceptually akin to using micropatterns [GM05] and nanopatterns [Bat10], but nothing of this kind has ever been done for grammars (in a broad sense or otherwise).

A different more advanced application of metasyntactic footprints and signatures is the analysis of their usage by mining existing grammar repositories like Grammar Zoo [Zay+08]. This can lead to not only improving the quality of the grammars by increasing their utilisation of the metalanguage functionality, but also to *validation of metalanguage design*. The whole programming language community uses dialects and variations of BNF [Bac60] and EBNF [Wir77], but their design has never been formally verified. However, one may expect that introducing EBNF elements like symbol repetition to BNF can be justified by analysing plain BNF grammars and finding many occurrences of encoding them ("yaccification", etc). It will also be interesting to see what new features the EBNF lacks practically — none of the existing proposals so far (ABNF [Ove05], TBNF [Man06], etc) were ever formally validated.

3.1.2 History of attempted publication

Initially, the idea of guided grammar convergence has emerged as a contribution for ECMFA [Zay12h]. The level of contribution was praised by the reviewers, but the paper itself was deemed inappropriate for a heavily model-related venue. A bit later it was resubmitted after minor revision to ICSM [Zay12i], where it was received even colder, presumably because the reviewers were seeking a more practical side which was not demonstrated well enough. After much more effort put into experiments, prototypes, auxiliary material [Zay12k] and a complete rewrite of the paper itself, the method was submitted to POPL [Zay12j]. It was unanimously rejected, but with very constructive and encouraging reviews. In 2013, they will be taken into account when the paper will be submitted again (the last time as a conference paper — otherwise I will admit it to be impossible for me to explain this method within the common limitations and go for a much longer self-contained journal submission).

In [Zay12m], I have attempted to sell the very act of validating the new method of guided grammar convergence by letting it cover the older case study done with contemporary grammar convergence, as a some sort of experimental replication in a broad sense. The reviewers praised the nonconformism and originality of the approach, and rejected the paper.

The generalisation of the method was proposed as an extended abstract to NWPT [Zay12t], where the reviewers did not see any merit in it (which I personally found strange since both ICSM and POPL reviewers insisted that various components of the method like ANF and prodsigs must be treasured as standalone contributions which applicability is much wider than the automated convergence of grammars). Either my way of explaining was bad enough to obfuscate this point, or I have terribly misunderstood their call for papers.

3.2 Grammar transformation languages

3.2.1 XBGF

XBGF, standing for Transformation of BNF-like Grammar Format, is a domain-specific language for automated programmable operator-based transformations of grammars in a broad sense. It has been previously implemented in Prolog (which was mostly done by Ralf Lämmel) and published as a part of a journal article [LZ11, §4], as well as a separate online manual [Zay+08, XBGF Manual] — in fact, just a byproduct of the research on language documentation [ZL11].

XBGF is essentially finished work: it is working, it is useful for experiments, it has documentation, it has a test suite, etc. The only thing that was added in the course of 2012 is the reimplementation of XBGF in Rascal [KSV11]. Beside some metaprogramming, this reimplementation led to streamlining some of the applicability preconditions and postcondition, which could be viewed as a very minor scientific contribution.

3.2.2 ΞBGF

If XBGF was read as "iks bee gee eff", then ΞBGF is "ksee bee gee eff", its bidirectional counterpart. Inspired by the call for papers of BX'12 (The First Workshop on Bidirectional Transformations, see §4.1), I was experimenting with bidirectionality in the grammarware technological space, and this language is what came out of it. 80% of the work for creating it involved trivial coupling of grammar transformation operators like *chain* and *unchain*, but the remaining 20% have provided a lot of fuel for thinking about what seemed to be a polished and finished product. ΞBGF was published as a part of online pre-proceedings [Zay12p], and then, after the second round of reviews, as a journal article [Zay12r]. The only problem was that the BX paper took off on its own, so the bidirectional grammar transformation operator suite seems like one of many byproducts there. There was a failed attempt to craft a paper that would be more focused on ΞBGF (and other aspects of grammar transformation not covered sufficiently by the BX submission), but a wrong venue was targeted, which resulted in desk rejection [Zay12ah].

3.2.3 NXBGF?

Another property of programmable grammar transformations that always bothered me, was their rigidity: once written, they are hard to maintain and adapt, and one little change in the original grammar (for example, when the extractor is changed) can unexpectedly and unpredictably break (make defunct) some of the transformation steps much later in the chain, and there is no method available to detect the change impact. Analysing this problem led to an idea that was originally in preparation for the FM+AM workshop (see §4.2), but was not ready before the deadline, so it went to the Extreme Modelling Workshop instead, where it received surprisingly warm reaction.

The idea is: negotiations. Whenever an error arises (usually an applicability condition is not met), instead of failing the whole chain, try to recover by negotiating the outcome with the data about near-failure and some external entity (usually an oracle or a human operator). For example, when we want to rename a nonterminal that does not exist, the transformation engine may seek nonterminals with names similar to the required one, and try renaming them.

The idea of negotiated grammar transformations was published in the online proceedings [Zay12u] and then in the ACM Digital Library [Zay12v], after which I was invited to submit an extended version to a journal. This will soon lead to a prototype implementation of such a system and perhaps to some interesting experiments with it. If this advancement yields a yet another grammar transformation operator suite, it may or may not be named "NXBGF".

3.2.4 EXBGF

Considerations about the state of XBGF led me to start cursory reexamination of the available transformation scripts. The Java case study undertaken in 2009–2010 and published as a conference paper [LZ09b], a journal paper [LZ11] and open source repository [Zay+08], provided me with plenty of them. Manual ad hoc pattern recognition has resulted in development of a new operator suite, with higher order operators such as *exbgf:pull-out*, which would be equivalent to a superposition of *xbgf:horizontal*, *xbgf:factor*, *xbgf:extract* and *xbgf:vertical*. As shown on Table 2, size metrics show a drop of 23–26% in Extended XBGF with respect to XBGF, but also the complexity was obviously decreased. However, the results were not extremely convincing and lacked real strength since only a few uses per high level operator were found, and the new EXBGF language was not designed systematically. Besides all that, the case study I have done, is, strictly speaking, about *refactoring* XBGF scripts to Extended XBGF, so claims about usefulness of EXBGF for *creating* new transformation scripts, should be stated with caution.

EXBGF was first described as an idea as a part of [Zay12ah]. After its rejection, it was developed further and laid out in much more detail in a journal submission, which was also eventually rejected [Zay12m]. The fact that I presented Extended XBGF first as a "trend" and then as an "experiment", perfectly reflects my point of view that it is not a solid contribution on its own.

3.2.5 ΔBGF?

If there was one good outcome of getting a grammar transformation paper [Zay12ah] rejected at a functional programming conference, then this is it: I started contemplating how to specify them in a non-so-functional way. Having recently been to a bidirectional transformations workshop helped, and I started researching *tridirectional* transformations (in fact, they quickly turned multidirectional). The idea was clean and simple: do not specify grammar changes as functions; instead, specify them as predicates. Such a predicate would, for example, introduce a nominal binding between nonterminals in different grammars — after which, the actual renaming steps can be easily inferred from such a binding predicate.

Unfortunately, this idea was so beautiful in theory, but proven nearly impossible in practice (or in detailed theory, for that matter). The main problem lies with the order of execution: a functional grammar transformation script specifies that order naturally, while a list of predicates does not. As I found out the hard way, my prototypes were still clean and beautiful when they dealt with one transformation step; reasonable tricks and extensions could let me go up to three steps; beyond that some serious redesign was needed; and so far I have not figured out how to overcome this.

3.3 Metasyntax

Whenever we have a software language, we can speak of its *syntax* as a way it allows and disallows structural combinations of elements: programming languages rely on keywords and possibly layout conventions; spreadsheets have ways of distinguishing between cells and referring to one from another; markup languages have symbol sequences of special meaning; musical notes are arranged on a grid; graphs must have uniquely identifiable nodes and edges connecting exactly two each; etc. Then, a *metasyntax* is a way of specifying this syntax. In the classic programming language theory, languages are textual and can be processed as sequences of lexems, and the metasyntax is Backus Normal Form [Bac60], also called Backus Naur Form [Knu64], or its enhanced variant Extended Backus Naur Form [Wir77]. Despite the fact that EBNF has been standardised by ISO [ISO96], there is no agreement in the software language engineering community on the exact variant of EBNF: some people just prefer using ":≡" or "≜" instead of "=" for esthetic reasons or prefer separating production rules with double newlines for readability reasons and for the sake of easy processing.

The idea was hinted in my PhD thesis in 2010 [Zay10], completely worked out in 2011 and was put to several good uses in 2012. These are listed in the following subsections.

	jls1	jls2	jls3	jls12	jls123	r12	r123	Total
XBGF, LOC	682	6774	10721	5114	2847	1639	3082	30859
EXBGF, LOC	399	5509	7524	3835	2532	1195	2750	23744
	−42%	−19%	−30%	−25%	−11%	−27%	−11%	−23%
genXBGF, LOC	516	5851	9317	4548	2596	1331	2667	26826
	−24%	−14%	−13%	−11%	−9%	−19%	−13%	−13%
XBGF, nodes	309	3,433	5,478	2,699	1,540	786	1,606	15851
EXBGF, nodes	177	2,726	3,648	1,962	1,377	558	1,446	11894
	−43%	−21%	−33%	−27%	−11%	−29%	−10%	−25%
genXBGF, nodes	326	3,502	5,576	2,726	1,542	798	1,610	16080
	+6%	+2%	+2%	+1%	+0.1%	+2%	+0.3%	+1%
XBGF, steps	67	387	544	290	111	77	135	1611
EXBGF, steps	42	275	398	214	98	50	120	1197
...pure EXBGF	27	104	162	80	30	34	44	
...just XBGF	15	171	236	134	68	16	76	
	−37%	−29%	−27%	−26%	−12%	−35%	−11%	−26%
genXBGF, steps	73	390	555	296	112	83	139	1648
	+9%	+1%	+2%	+2%	+1%	+8%	+2%	+2%

Table 2: Size measurements of the Java grammar convergence case study, done in XBGF and in EXBGF. In the table, XBGF refers to the original transformation scripts, EXBGF to the transformations in Extended XBGF, genXBGF measures XBGF scripts generated from EXBGF. LOC means lines of code, calculated with `wc -l`; nodes represent the number of nodes in the XML tree, calculated by XPath; steps are nodes that correspond to transformation operators and not to their arguments. Percentages are calculated against the XBGF scripts of the original study.

3.3.1 Notation specification

The first step in treating metalanguages as first class entities is, of course, encapsulating a particular metalanguage with a specification that defines it. By extending the list of possible metasymbols from the ISO EBNF standard [ISO96] and by reusing the empirically constructed Table 6.1 from my thesis [Zay10, p.135], I was able to construct such a specification, which was subsequently named EDD, for EBNF Dialect Definition. It was then turned into a small nicely packaged paper for the PL track of SAC [Zay12d] — the very fact that it was published separately, gave me a lot of freedom later, when I did not feel like I need to introduce all the metasymbols all over again in each work that followed.

3.3.2 Transforming metasyntaxes

Once you have a notation specification as a first class entity, you can define transformations on them. This was probably the first transformation language that I have designed, where the main complexity was not in defining the transformation operators as such, but rather in coupling them with the grammar transformation steps that they imply. The transformation suite consisted of just three operators:

rename-metasymbol(s, v_1, v_2) where s is the metasymbol and values v_1 and v_2 are strings
For example, we can decide to update the notation specification from using "`:`" as a defining metasymbol to using "`::=`". This is the most trivial transformation, but also bidirectional by nature.

introduce-metasymbol(s, v) where s is the metasymbol and v is its desired string value
For example, a syntactic notation can exist without terminator metasymbol, and we may want to introduce one.

eliminate-metasymbol(s, v) where s is the metasymbol and v is its current string value
Naturally, eliminate and introduce together form a bidirectional pair. Specifying the current value of a metasymbol is not necessary, but enables extra validation, as well as trivial bidirectionalisation.

Yet, the final megamodel of the infrastructure that did not even consider language instances (only grammars and metasyntaxes) looked as complex as Figure 3. The paper about evolution of metalanguages had a bidirectionality flavour and was conditionally accepted at the BX workshop [Zay12p], and then also for the journal special issue [Zay12r].

3.3.3 Notation-parametric grammar recovery

In all previously published grammar recovery initiatives [BSV97; LV99; SV00; LV01a; LV01b; Läm05; Zay05; LZ09a; Zay10; LZ11; Zay11b; Zay12ae] the step of transforming the raw grammar-containing text obtained from the language manual was either not automated (the grammar was re-typed from scratch in the notation required by the target grammarware framework), or semi-automated (comprised

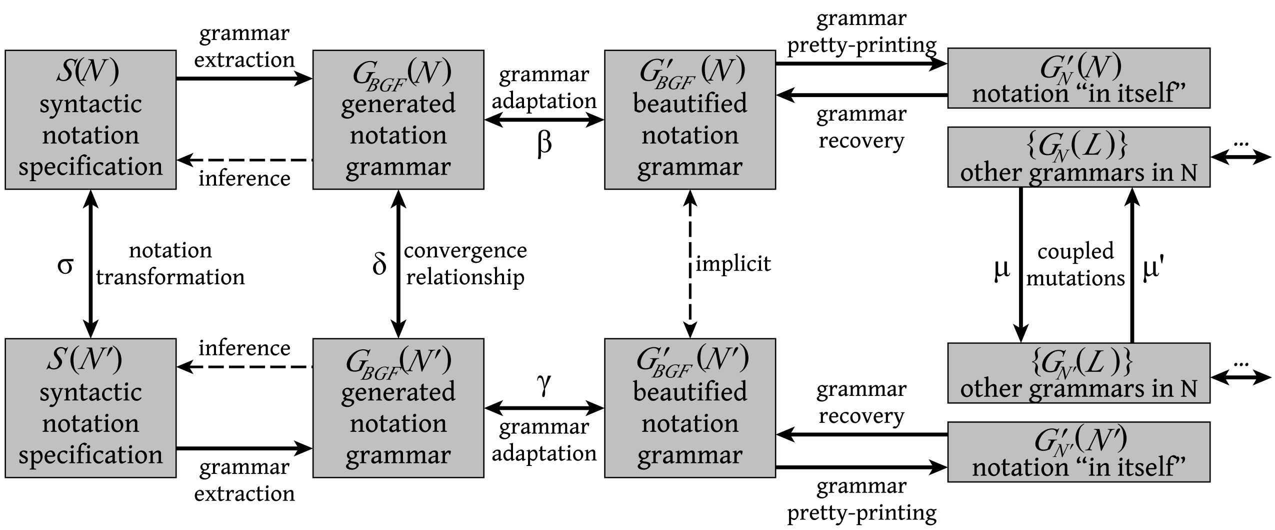

Figure 3: Components of a notation evolution: σ, a bidirectional *notation specification transformation* that changes the notation itself; δ, a *convergence relationship* that can transform the notation grammars; γ, a bidirectional *grammar adaptation* that prepares a beautified readable version of N'. μ, an unidirectional *coupled grammar mutation* that migrates the grammarbase according to notation changes; possibly μ', an unidirectional *coupled grammar mutation* that migrates the grammarbase according to the inverse of the intended notation changes.

many rounds of test-driven improvement), or automated with a throwaway tool (one that can not be reused unless the replication deploys exactly the same EBNF dialect). Having a notation specification as a first class entity, we can step up from throwaway tools to throwaway notation specifications: at least they take minutes to create, not days.

Notation-parametric grammar recovery [Zay12y; Zay12x] was my best result of 2011, and this year it was officially published and put to several good uses. These uses are not exactly publishable simply because grammar recovery from (nearly) well-formed has become a trivial process itself, but there was one story that was enabled by this triviality. The grammar of MediaWiki syntax, for recovery of which I have a previously exposed preprint [Zay11b], is a unique case of using multiple notations within one community-created grammar. With any other recovery method, it would have been easier to just retype the grammar again in a uniform fashion, but notation-parametric grammar recovery allowed to treat all six different incoherent metalanguages with relative ease and derive the final grammar from the inconsistent input. A continuation of this topic was intended to be a published closure on the case of MediaWiki grammar recovery, but was unfortunately rejected in the end [Zay12ae].

3.3.4 Notation-driven grammar convergence

Grammar convergence was originally a lightweight verification method not intended for full automation [LZ09a]. However, seeing how many transformations that were in fact converging grammars, it was possible to infer automatically for the metalanguage evolution case study [Zay12t] (see also §3.3.2), I could not help starting to wonder whether and to what extent it was possible to drive the automated convergence process by the notation properties. The result of that was the methodology of guided grammar convergence, which was already covered by §3.1.

3.4 Tolerance in parsing

Originally, the "parsing in the cloud" paper [Zay12n; Zay12o] was intended to present a useful crossing of the in-the-cloud and as-a-service paradigm with the engineering discipline for grammarware. However, the related work digging quickly got out of hand and turned into a contribution of its own. The overview of many grammar-based techniques with some level of tolerance towards their input data and its weak commitment to grammatical structure, was presented at the PEM Colloquium [Zay12ag] (see also §3.8.8), where it was received very warm acceptance and led to many useful insights. It has been advised to me both by reviewers and colleagues to put more effort into demonstrative prototype and publish the overview with them separately from the parsing algorithm (see §3.8.2) itself. This is among one of the planned activities for 2013.

So far, at least the following tolerant parsing methods have been identified: ad hoc lexical analysis [BSV00; KLV05b], hierarchical lexical analysis [MN95], iterative lexical analysis [Cox03], fuzzy parsing [Kop97], parsing incomplete sentences [Lan88], island grammars [DK99], lake grammars [Moo01], robust multilingual parsing [SCD03], gap parsing [BN05], bridge grammars [NNEH09], skeleton grammars [KL03], breadth-first parsing [Lee67; Oph97], grammar relaxation [ASU85], agile parsing [Dea+03], permissive grammars [Kat+09], hierarchical error repair [BH82], panic mode [ASU85], noncorrecting error recovery [Ric85], precise parsing [AU72]. It remains to be seen whether they form a straight spectrum from lexical analysis to strict syntactic analysis.

3.5 Megamodelling

In computer science, *modelling* happens when a real artefact is represented by its abstraction, which is then called a model; *metamodelling* happens when the structure of such models is analysed and expressed as a model for models, or a metamodel; and *megamodelling* happens when the infrastructure itself, involving multiple models and metamodels, is modelled. The need for megamodels is being advocated at least since 2004 [BJV04; FN04].

The current state of the art is: in the simplest cases, people do not need a special formalism to state that, for example, "models A and B conform to the metamodel C"; in somewhat more complicated scenarios scientists and engineers tend to develop their own domain-specific *ad hoc megamodelling* methodologies and employ them in narrow domains; and in truly complex situations, any existing approach only adds to complexity, overwhelming stakeholders with a yet another view on the system architecture. However, at least one solid business case was found for megamodelling: the problem of comparing different technological spaces [KBA02]: for example, comparing the relations between XML documents, schemata, data models and validators, with relations between object models, source code and compilers.

At the University of Koblenz-Landau, the Software Languages Team is dedicated to develop a general purpose megamodelling language called MegaL [FLV12]. After attending presentations about MegaL on several occasions, I have paid a working visit to them in July. The consequences of that visit: I tried to use MegaL for my own megamodelling needs on several occasions [Zay12u; Zay12v; Zay12k], I have presented an extensive overview of currently existing ad hoc megamodelling techniques (see §3.5.1), and I have proposed my own method of dealing with overly complex megamodels (see §3.5.2).

3.5.1 MegaL dissection

So far at least these previously existing ad hoc megamodelling approaches have been spotted: ATL [Jou+08], UNCOL [Bra61; Con58], tombstone [MHW70], grammarware megamodelling [KLV05a], software evolution megamodelling [FN04], evolution of software architectures [Gra07a; Gra07b], MEGAF [Hil+10], global model management [Vig+11], grammar convergence [LZ09a; Zay10; Zay11a], software language engineering [Zay10; Zay+08], modelling language evolution framework [MV11], metasyntactic evolution [Zay12p; Zay12r].

My superficial overview of them, comparing them with MegaL, was presented to the MegaL designers in July [Zay12s], and my current research activities include active collaboration with them with a paper presenting a unified model for megamodelling in mind.

3.5.2 Renarrating megamodels

Having seen enough presentations on megamodelling made me realise that they are very easy to follow even for untrained people, unlike the resulting megamodels that contain far too much detail and are very intimidating. So, my take on this problem was introducing two operations: slicing (to make megamodels smaller) and narrating (to traverse the elements in the megamodel). If we have them, we can take the baseline megamodel that only experts can try to understand, and cut it to consumable chunks bundled with the story that introduces the remaining elements one by one and explaining each step. The resulting paper was sent to a workshop on Multiparadigm Modelling, where it was presented as a poster [Zay12aa], published in online pre-proceedings [Zay12ab] and is currently on its way to the post-proceedings in the ACM DL [Zay12ac].

3.6 Grammar repository

My first project proposal ever, titled "Automated Reuse-driven Grammar Restructuring", was sent to the NWO Veni program in January, passed a rebuttal phase in May and was finally rejected in July after informing me that it ended up in the category "very good" [Zay12a]. The idea described there was small and elegant: mining grammarware repositories. While repository mining techniques receive quite some attention nowadays, very few people actually have entire repositories filled with grammars: let's face it, they are omnipresent yet at the same time scarce. However, I already have this initiative called Grammar Zoo [Zay+08], which contains many grammars of languages big and small, and armed with the arsenal of extraction tools developed in my PhD time, it can grow even more. The goal of such mining is, of course, to reverse engineer reusable grammar fragments and forward engineer the discipline of their composition.

A paper advocating the need and the usefulness of the repository itself, was written and submitted to a journal in November [Zay13]. The outcome will only become known in 2013.

3.7 (Open) Notebook Science

Open Notebook Science is an open science paradigm of doing research in a transparent way [Llo08]. It involves keeping a lab notebook that collects all data and metadata on experiments, hypotheses, results, details and other observations that occur during the research phase, so that after the final objective is reached (or deemed unreachable), the complete path towards it can be exposed and made publicly available for inspection, replication and reuse. The open notebook approach is fairly well-known and somewhat popular in fields like biology and chemistry [San08; Sin08], that strive on experimental frameworks and traditionally involve lab notebooks, so in practice exercising this approach had the only consequence of sharing the already existing notebook and systematically referring to it from the papers. In computer science, however, there are none to few adopters of this approach, mainly due to the seeming complexity of the method and the amount of extra effort that is needed to set up and to maintain such a lab notebook and the lack of positive feedback from it in the form of community encouragement and peer acknowledgement.

During the Software Freedom Day, I have given a presentation, explaining one possible feasible way to start practicing open notebook science for computer science and software engineering researchers, with the case study of myself [Zay12z]. A couple of days later SL(E)BOK organisers have heard about it and asked me to record a keynote presentation [Zay12af] about that, linking open access ideas with the existing research on "scientific knowledge objects" (SKO) and on a "body of knowledge" [Giu+10; Sim+11].

In short, open notebook science strives to enable open access to atomic SKOs; to expose all the dark data [Goe07] from failed experiments and unpublished results; to self-archive [Har01] subatomic SKOs, which are relevant for the final result, but smaller than a "publon". Examples of subatomic SKOs include:

- Commits to an open source repository;
- Tweets on work-related subjects;
- Quora answers on work-related topics;
- Papers: preprints, reports, drafts, etc;
- Presentations: slides, screencasts, etc;
- Blog posts;
- Wiki edits;
- Exposed tools;
- Documentation;
- Shared raw data;
- Auxiliary material.

As it has been pointed out to me by some of the attendees of both talks, the topic of subatomic SKOs is bigger than just open notebook computer science, because if I can show the usefulness of keeping a notebook of actions for a researcher, it does not necessary mean that the notebook must be public to profit from its traceability. The first comprehensive paper on this topic is still in the process of being designed, but hopefully will be submitted somewhere during the next year or two.

3.8 Minor topics

Additionally to the topics and achievements I consider major for 2012, there are several lesser contributions: their are either topics that did not receive enough attention to yield a solid major contribution (yet not insignificant enough to be omitted from the report completely); or just not traditionally considered worthy of mentioning (programming, engineering, organising effort).

One topic is intentionally hidden from this section, in order to prevent jeopardising an upcoming submission to a strictly double blind peer reviewed venue.

3.8.1 Grammar mutation

In the paradigm of programmable grammar transformations, the semantics of each of the transformation operators is bound to the operator itself, and may require arguments to be provided before the actual input grammar. Such partially evaluated operators (with all arguments provided, but no input grammar yet) are treated as transformation steps, and their applicability constraints only depend on the grammar: if they hold, the change takes place; if they do not, an error occurs instead. In other words, the exact consequence of the transformation step depends on operands, not on the grammar. However, those applicability constraints can also be processed as filters: whatever part of the grammar satisfies them, will be transformed — that way, the exact change in the grammar depends on the grammar, not on the operands.

As an example, consider renaming grammatical symbols: "rename nonterminal" itself is an operator. Its semantics can be expressed easily on the classic definition of a grammar. If the input grammar is $G = \langle \mathbb{N}, \mathbb{T}, \mathbb{P}, S \rangle$, then the output must be

$$G' = \langle \mathbb{N} \cap \{x\} \cup \{y\}, \mathbb{T}, \mathbb{P}|_{x\to y}, S' \rangle$$

where x and y are operands; S' is S unless $S = x$ and y otherwise; and $A|_{x\to y}$ means substitution (for example, by term rewriting). When x and y are provided, then G' above becomes fully defined and yields meaningful results when applicability conditions (e.g., $x \in \mathbb{N}$ and $y \notin \mathbb{N}$) are satisfied. Renaming a terminal symbol is specified similarly.

However, "renaming all lowercase nonterminals to uppercase" is not an operator (or at least even it is made one, it will be of much higher level than the simple "rename"), and it is not an atomic transformation step either: in fact, it can lead to any number of changes in the grammar from 0 to $|\mathbb{N}|$, depending on G. This number absolutely cannot be known before G is provided.

This kind of grammar manipulation was identified first as a part of research on bidirectional transformations [Zay12p; Zay12r; Zay12q; Zay12c] (because they are not bidirectionalisable), where it received the name of "grammar mutation". Later there was an endeavour to compose a comprehensive list of useful grammar mutations as a part of [Zay12ah], but it was rejected.

For the sake of providing a better overview of the current state of research on grammar mutations, I collect all of them in the exhaustive list below. Note that conceptually the same mutations may have been appearing under different names in various sources: for example, the first mutation in the list, "remove all terminal symbols", has previously been known as a transformation "*stripTs*" [LZ09a, §5.3] and as a generator "*striptxbgf*" [Zay10, §4.9,§4.10.6.1].

Remove all terminal symbols [LZ09a; Zay12h; Zay12i; Zay12j; Zay10; Zay11a]
: A simple grammar mutation that is helpful when converging a concrete syntax and an abstract syntax of the same intended language. While the abstract syntax definition may have differently ordered parameters of some of its constructs, and full convergence will require dealing them them and rearranging the structure with (algebraic) semantic-preserving transformations, we will certainly not encounter any terminal symbols and can safely employ this mutation.

Remove all expression selectors [LZ09a; Zay12h; Zay12i; Zay12j; Zay10]
: Named (selectable) subexpressions are encountered in many contexts, but the choice of names for them is usually even more subjective than the naming convention for the nonterminal symbols.

Remove all production labels [Zay12h; Zay12i; Zay12j]
: Technically, having production label is the same as making a selectable subexpression out of the right hand side of a nonterminal definition. Still, in some frameworks the semantics and/or the intended use for labels and for selectors differ.

Disciplined rename [Zay12p; Zay12r; Zay10; Zay11a]
: There are several different well-defined naming conventions for nonterminal symbols in current practice of grammarware engineering, in particular concerning multiword names. Enforcing a particular naming convention such as making all nonterminal names uppercase or turning camelcased names into dash-separated lowercase names, can be specified as a unidirectional grammar mutation (one for each convention).

Reroot to top [Zay11a; Zay12h; Zay12i; Zay12j]
: A top nonterminal is a nonterminal that is defined in the grammar but never used [LV01b]. In many cases it is realistic to assume that the top nonterminals are intended starting symbols (roots) of the grammar. A variation of this mutation was used in §3.1 with an additional requirement that a top nonterminal must not be a leaf in the relation graph. This is a rational constraint since a leaf top nonterminal defines a separated component.

Eliminate top [Zay10; Zay11a]
: In the situations when the root is known with certainty, we can assume all other top (unused) nonterminals to be useless, since they are unreachable from the starting symbol and are therefore not a part of the grammar.

Extract subgrammar [Zay12h; Zay12i; Zay12j]
: Alternatively, we can generalise the last mutation to a parametrised one: given a grammar and a nonterminal (or a list of nonterminals), we can always automatically construct another grammar with the given nonterminal(s) as root(s) and the contents formed by all production rules of all nonterminals reachable from the assumed root nonterminal(s). Constructing a subgrammar starting with the already known roots will eliminate top nonterminals.

Make all production rules vertical [Zay10; Zay11a; Zay12h; Zay12i; Zay12j]
: Vertical definitions contain several alternative production rules, while horizontal ones have one with a top level choice. There are different approaches known to handle this distinction, including complete transparency (one form being a syntactic sugar of the other). For normalisation purposes or for quick convergence of a consistently vertical grammar and a consistently horizontal one, we can use this automated mutation.

Make all production rules horizontal [Zay10; Zay11a]
: A similar grammar mutation is possible, yet much less useful in practice.

Distribute all factored definitions [Zay12h; Zay12i; Zay12j]
: Aggressive factoring a-la `xbgf:distribute` can also be discussed.Surfacing all inner choices in a given grammar is a powerful normalisation technique.

Make all potentially horizontal rules vertical [Zay10; Zay11a]
: Technically, this mutation is a superposition of distribution of all factored definition and converting all resulting horizontal production rules to an equivalent vertical form.

Deyaccify all yaccified nonterminals [Zay10; Zay11a]
: A "yaccified" definition [Läm01; JM01] is named after YACC [Joh75], a compiler compiler, the old versions of which required explicitly defined recursive nonterminals — i.e., one would write

A : B and A : A B, because in LALR parsers like YACC left recursion was preferred to right recursion (contrary to recursive descent parsers, which are unable to process left recursion directly at all). The common good practice is modern grammarware engineering is to use iteration metalanguage constructs such as B* for zero or more repetitions and B+ for one or more — this way, the compiler compiler can make its own decisions about the particular way of implementation, and will neither crash nor perform any transformations behind the scenes. However, many grammars [Zay+08] contain yaccified definitions, and usually the first step in any transformation that attempts to reuse such grammars for practical purposes, start with deyaccification, which can be easily automated.

Remove lazy nonterminals [Zay10; Zay11a]
Many grammars, in particular those that strive for better readability or for generality, contain excessive number of nonterminals that are used only once or chain production rules that are unnecessary for parsing and for many other activities one can engage in with grammars. We have used an optimising mutation that removes such elements with *xbgf:inline* and *xbgf:unchain* on several occasions, including improving readability of automatically generated grammars.

Normalise to ANF [Zay12h; Zay12i; Zay12j]
The Abstract Normal Form (ANF) was introduced in §3.1 as means of limiting the search space for guided grammar convergence. Technically, such normalisation is equivalent to a superposition of removing all labels, removing all selectors, removing all terminals, surfacing all inner choices, converting all horizontal production rules to a vertical form, rerooting to top non-leaf nonterminals and eliminating others unreachable from them. For conceptual foundations of ANF the reader is redirected to the article where it was proposed.

Fold all grouped subexpressions [Zay12p; Zay12r]
In the context of metalinguistic evolution, we need to construct a coupled mutation for the grammarbase, if the notation change contains retiring of a metasyntactic construct that is in use. One of such constructs is the possibility to group symbols together in an atomic subsequence — a feature that is often taken for granted and therefore misused, improperly documented or implemented. Naturally, eliminating grouped subexpressions entails folding them to newly introduced nonterminals by means of *xbgf:extract*.

Explicitly encode all separator lists [Zay12p; Zay12r]
Our internal representation of grammars for software languages, following many other syntactic notations, contains a construct for defining separator lists. For example: {A ","}+ is a syntactic sugar for A ("," A)* or (A ",")* A — all three variants specify a comma-separated list of one or more As. When such a construct needs to be retired from the notation, the coupled grammar mutation must refactor its occurrences to explicitly encode separator lists with one of the equivalent alternatives.

A full fledged paper shining enough light on grammar mutations, is still being written and will hit the submission desks in 2013.

3.8.2 Iterative parsing

As the main (intended) contribution of [Zay12n; Zay12o], I have proposed the algorithm for iterative parsing. The basic idea is very simple: we take the baseline grammar and skeletonise it as far as it can be automated, in such a way that the relation between the "lakes" and the nonterminals in the baseline grammar are preserved. Then, our parse tree will give the basic structure and a number of watery fragments parsed with useless lake grammars (usually in a form of "anything but newline" or "something in balanced out curly brackets"). If needed, any of those lakes can be parsed further with a subgrammar of the baseline grammar, with the new root being the nonterminal that corresponds to the lake.

This parsing approach was being sold as "parsing in the cloud" in [Zay12n; Zay12o], which was certainly not the best (even though the coolest) way to look at it. Other applications for this form of lazy parsing can be found in debugging (disambiguation, fault localisation) and other areas that traditionally profit from laziness. This remains future work.

3.8.3 Unparsing techniques

One of the most confusing paper that I have submitted anywhere in 2012, was the one about unparsing techniques [Zay12ai]. Only after finishing writing it, I have realised how big and overwhelming this topic is. The paper was rightfully rejected after being classified as a "request for discussion": a much deeper survey of (some of) the presented topics must be composed sooner or later, but it requires much careful consideration. I have not done much in this topic after that, but there was at least one paper published recently that explicitly considered unparsing [SCL12].

The starting idea is simple as a sunrise: there was a lot of effort put in researching parsing techniques, so why not the opposite? The unparsing techniques can be understood in a very broad sense: pretty-printing, syntax highlighting, structural import yielding an editable textual representation, bidirectional construction of equivalent views, etc.

Some papers consider *conservative pretty-printing* as a way to preserve peculiar layout pieces (like multiple spaces) during unparsing [Jon02; Ruc96]. This is a narrow application of a general idea of propagating layout through transformations, which is a long-standing and a well-researched problem. However, even the most conservative unparsers have the risk of introducing an inconsistently formatted code fragment, if that code was originally introduced by a source code manipulation technique and not produced by a parser. In other words, replacing a `GO TO` statement with a `WHILE` loop should look differently, depending on how the code around the introduced fragment was formatted. Possibly, results from the grammar inference research field [SC12] can be reused for recovering formatting rules in some reasonable proximity of the code fragment in order to unparse it correctly and avoid code alienation.

Suppose not just one desired textual formatted representation of the language instance exists, but several of them, which form a family, or a product line, like the line of metalanguages considered in §3.3 in the context of metasyntactic evolution. Following that example, suppose we are given a grammar in some internal representation and a syntactic notation specification [Zay12d], then it is somewhat trivial to construct an unparser that would produce the same grammar in a textual form. In other words, such an unparser should generate a text that, given a notation specification, can yield the same grammar after automated notation-parametric grammar recovery [Zay12x, §3]. However, other questions remain. How to find a minimal notation needed to unparse a given grammar? How in general to validate compatibility of a given grammar and a given notation? How to produce grammar transformations (see §3.2) to make the grammar fit the notation, how to produce notation transformations (see §3.3.2) to make the notation fit the grammar, and how to negotiate to find a properly balanced outcome? These questions are not trivial and require investigation. Unparser-completeness has recently been studied in the context of template engines [ABS11].

Unparsing can also be viewed as commitment to grammatical structure [KLV05a]. Can we recover grammars from them, compare and converge them with other grammars of the same language that we would like synchronised (e.g., concrete syntax definition intended for parsing, multiple abstract syntaxes for performing various grammar-based analysis tasks, data models for serialisation)? Are there some specific properties that such grammars always possess? What is the minimal upper formalism for the baseline grammar from which grammars for parsing and unparsing can be derived automatically with a language-independent or language-parametric technology? These questions are not trivial and require investigation.

Connecting to the topic of robust/tolerant parsing (see §3.4), we can consider at least two kinds of techniques that as the opposite: incremental unparsing and unparsing incomplete trees. By *incremental unparsing* I mean a modular technique for unparsing modified code fragments and combining them with the previously unparsed versions of the unmodified code fragments. This is usually not considered for simple cases, but is possibly worth investigating for large scale scenarios (consider architectural modifications to an IT portfolio with hundreds of millions lines of code in dozens of languages). By *unparsing incomplete trees* we define the process of unparsing structured representations of incomplete language instances. Besides scenarios when this technique is used together with tolerant/robust parsing (and then the lacking information may be somehow propagated to the unparser anyway), there are also other scenarios when the gaps are deliberately left out to be filled by the unparser. In documentation generation, this is the way code examples can be treated — for a sample implementation we refer to Rascal Tutor [Kli+12].

For construction of compiler compilers and similar grammarware with unparsing facilities, there is a commonly encountered problem of bracket minimality for avoiding constructions ambiguous for parsing: since brackets are there in the text only to guide the parsing process, they are removed from the AST, so how to put back as few of them as possible during unparsing? This is a typical research question for the unparsing techniques field. One could also investigate various ways to infer grammar adaptation steps needed to unparse the given grammatical structure in order to guarantee the lack of ambiguities if it is to be parsed again.

3.8.4 Migration to git

Following the current trend of leaving old-fashioned open source farms in favour of more modern 2.0 social coding websites, I have migrated the Software Language Processing Repository from SourceForge to GitHub [Zay+08]. The project was started in 2008 by Ralf Lämmel [Läm08] and quickly after that become the main target for my efforts and the main repository for my code. As of now (December 2012), it contains 954 revisions committed by me, 314 by Ralf Lämmel, 44 by Tijs van der Storm and 28 by all other contributors combined.

This would have not been worth mentioning, if I did not migrate all my other repositories to `git` as well, which enabled efficient linking to all of them from the open notebook (see §3.7). For closed source repositories (like ones used for writing papers) we use Atlassian BitBucket instead of GitHub.

3.8.5 Turing machine programming

Two of my colleagues from Centrum Wiskunde & Informatica (CWI), Davy Landman and Jeroen van den

Bos, have built a physical Turing machine with a finite tape and separate program space, from LEGO blocks [Bos+12]. We were all passively yet encouragingly watching them do that and then watching with excitement how the resulting machine could sum two and two in less than half an hour. From the software perspective, they have created a kind of "Turing assembly" DSL that consisted of commands for accessing bits on the tape, moving the head and making decisions on the next command, and was translatable into some real code that could run on the LEGO chip brick. Then, there was a slightly more advanced DSL called "Turing level 2" developed on top of it, enhanced with label names and repetition loops, as well as IDE support features like a visualiser/simulator.

My spontaneous contribution to the project involved writing several programs for the machine in this "Turing language level 2", including copying of unary numbers, incrementing them, performing various forms of addition and finally multiplying two unary numbers. All these programs are publicly accessible at the official repository: `http://github.com/cwi-swat/TuringLEGO/tree/master/examples`.

3.8.6 Grammarware visualisation

Various controversial thoughts on grammar recovery visualisation, related to the previous body of works on grammar recovery both (co)authored by me [Zay05; LZ09a; Zay10; Zay11b; LZ11; Zay12d; Zay12x; Zay12ae] and the giants on shoulders of which I was standing [BSV97; LV99; SV00; LV01a; LV01b; KLV05a], yielded some experimental code, but no valuable stable results.

In a draft sent to the "new ideas" track of FSE 2012 [Zay12aj] to be rejected there, I have argued that introducing or improving visualisation of processes in grammarware engineering has at least these benefits:

Process comprehension: it becomes easier to understand the process and to see what exactly is happening when it is applied to certain input.

Process verification: while complete formal verification of a sophisticated process with many branches and underlying algorithms, may be a challenging task, it is relatively easy to pursue lightweight verification methods. One of them comes more or less for free when an experienced observer can see what is happening and detect peculiarities naturally.

Process improvement: observing a process does not only let one find mistakes in it, but also to get familiar with bottlenecks and other problematic issues, which in turn will help to suggest refinements and improvements.

Interactiveness: there are many examples of processes which are impossible or unfeasibly hard to automate completely, but for which reasonable automation schemes exist that exercise "semi-automation" and require occasional feedback from a system operator. The request-response loop for such feedback can be drastically shortened in the case of interactive visualisations.

The point of the paper was well-received by the FSE NIER reviewers: nobody tried to argue that visualisation techniques would be useless. However, I obviously overestimated a contribution that I could make with providing a "mile wide, inch deep" (a quote from one of the reviews) overview, so perhaps a much later overview with the list of solid achieved results, would be in order. For the sake of completeness of this report, I list the nine showcases that were briefly described in the NIER submission below. Each item of this list is a relatively low hanging fruit for an article or a series thereof.

Grammar recovery: the state of the art in automated grammar recovery (see also §3.3.3) is to work based on a set of appropriate heuristics [LZ11; Zay12x]. Proper visualisation of them would help: dealing with some particularly tricky notations; verifying that the heuristics do what they are intended to do; collecting evidence and statistics on the use of certain heuristics; proposing additional heuristics and other process improvements.

API-fication is a term used in [KLV05a] to describe a process of replacing low level API calls for manipulating a data structure with more expressive and more maintainable high level API calls generated from a grammar [JO04]. Thus, API-fication is a form of grammar-aware software renovation where surfacing grammar knowledge is a crucial contribution of the process. Visualising both the API calls themselves and the improvement steps on them, can serve as a motivation and even as a lightweight verification of API-fication.

Grammar transformation & convergence. There are at least two commonly used ways to visualise a grammar: in a textual form as (E)BNF; or as a syntax diagram ("railroad track"). Neither of them has a designated visualisation notation for transformations.

Mapping between grammar notations is of the biggest challenges in research on grammars in a broad sense, since grammarware strives to cover such a big range of various structural definitions. Mapping between EBNF dialects [Zay12r], X/O mapping [Läm07], O/R mapping [O'N08], R/X mapping [Fer+02] and many other inter-notational mappings exist along with intranotational techniques for grammar diffing, graph comparison, nonterminal matching, model weaving, etc. Displaying matching artefacts in a traceable way by metagrammarware tools is usually rather limited and either display local (mis)matches or global statistics.

Grammarware coevolution. Concurrent and coupled evolution of grammars and language instances [Cic+08], of coexisting related grammars [Läm01], of grammars and language transformations [CH06], of language design and implementation [D'H+01] are special mixed cases of mapping and transformations (see last two sections), where we would like to visualise both what kind of matches are made and what kind of actions are inferred from them.

Grammar-based analysis comprises syntactic analysis (parsing), but also similarly geared techniques that never received enough attention. As an example, it would be great to have something to demonstrate hierarchical lexical analysis [MN95] to the same degree as [AMUFVI09] demonstrated for LL and LALR parsing.

Disambiguation is a process of filtering a parse forest or reasoning about the origins of it, in modern generalised parsing algorithms like SGLR [Vis97] or GLL [SJ10]. Visualising SGLR disambiguation [Bra+02] was implemented in the ASF+SDF Meta-Environment as a part of parse tree rendering, so in fact it visualised the ambiguities themselves and not the process of removing them, which was still of considerable help. More recent GLL disambiguation algorithms [Bas10] were expressed mostly in a textual form even within a PhD project entirely dedicated to ambiguity detection [Bas11] — primarily because there is no clear understanding of how exactly they would be useful to visualise.

Grammar-based testing methods based on combinatorial (non-probabilistic) exploration of the software language under test, have emerged from recent research [LS06; FLZ12]. Visualising coverage achieved by them and adjusting the visualisation with each new test case should help both to keep track of the process by expressing its progress, and to localise grammar fragments responsible for the failing test cases.

Grammar inference is a family of methods of inferring the grammar, partially or completely, from the available codebase and even from code indentation [Čre+05; Nie+07; SC12]. Such inference is a complicated process based on heuristics and sometimes even on search-based methods. As a consequence, each attempt at grammar inference remains somehow unconnected to the rest of the research field: adoption of such methods by scientists and engineers outside the original working group happens rarely, if ever. One can think that a proper visualisation of such process would help new users to get acquainted with a grammar reconstruction system and tweak it to their needs.

Production rule	**Prod. signature**
$\mathrm{p}(`', \mathit{Expr}, \mathit{Expr}_1)$	$\{\langle \mathit{Expr}_1, 1\rangle\}$
$\mathrm{p}(`', \mathit{Expr}, \mathit{str})$	$\{\langle \mathit{str}, 1\rangle\}$
$\mathrm{p}(`', \mathit{Expr}, \mathit{Expr}_2)$	$\{\langle \mathit{Expr}_2, 1\rangle\}$
$\mathrm{p}(`', \mathit{Expr}, \mathit{Expr}_3)$	$\{\langle \mathit{Expr}_3, 1\rangle\}$
$\mathrm{p}(`', \mathit{Expr}, \mathit{int})$	$\{\langle \mathit{int}, 1\rangle\}$
$\mathrm{p}(`', \mathit{Function}, \mathrm{seq}([\mathit{str}, *(\mathit{str}), \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 1\rangle, \langle \mathit{str}, 1*\rangle\}$
$\mathrm{p}(`', \mathit{Program}, *(\mathit{Function}))$	$\{\langle \mathit{Function}, *\rangle\}$
$\mathrm{p}(`', \mathit{Expr}_1, \mathrm{seq}([\mathit{str}, *(\mathit{Expr})]))$	$\{\langle \mathit{str}, 1\rangle, \langle \mathit{Expr}, *\rangle\}$
$\mathrm{p}(`', \mathit{Expr}_2, \mathrm{seq}([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1\rangle, \langle \mathit{Expr}, 11\rangle\}$
$\mathrm{p}(`', \mathit{Expr}_3, \mathrm{seq}([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111\rangle\}$

Table 3: The JAXB grammar in a broad sense: in fact, an object model obtained by a data binding framework. Generated automatically by JAXB [FV99] from the XML schema for the Factorial Language [Zay12k].

NB: the last item was written before the publication of the excellent grammar inference field overview [SC12], which can also be seen as considering visualisation in a very broad sense.

Another newer initiative which can be seen as grammarware process visualisation, concerns guided convergence (see also §3.1). We can recall that the whole process of the guided grammar convergence is rather complicated and involves normalising the input grammar and going through several phases of unification to ensure the final nonterminal mapping that looks like this [Zay12k]:

$$\begin{aligned} \mathit{jaxb} \diamond \mathit{master} = \{ & \langle \mathit{Expr}_2, \mathit{binary}\rangle, \\ & \langle \mathit{Expr}_3, \mathit{conditional}\rangle, \\ & \langle \mathit{int}, \mathit{int}\rangle, \\ & \langle \mathit{Function}, \mathit{function}\rangle, \\ & \langle \mathit{str}, \mathit{str}\rangle, \\ & \langle \mathit{Program}, \mathit{program}\rangle, \\ & \langle \mathit{Expr}, \mathit{expression}\rangle, \\ & \langle \mathit{Expr}_1, \mathit{apply}\rangle, \\ & \langle \mathit{Ops}, \mathit{operator}\rangle\} \end{aligned}$$

While preparing the main guided grammar submission, I have noticed that this particular mapping, as well as the normalised grammar (Table 3) and the list of weakly and strongly prodsig-equivalent production rules (Figure 4) can be automatically produced by the convergence tool virtually without any additional effort in a completely transparent, traceable, reliable and reproducible fashion. This led to open publication of [Zay12k], an extended appendix for the main guided grammar convergence paper, which was, except for the two-page introduction, generated automatically, but is still readable and useful.

$$
\begin{aligned}
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_1}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{apply})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{str}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{str})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_2}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{binary})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_3}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{conditional})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{int}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{int})\\
\mathrm{p}\,(`', \mathit{Function}, \mathrm{seq}\,([\mathit{str}, *\,(\mathit{str})\,, \mathit{Expr}])) &\Bumpeq \mathrm{p}\,(`', \mathit{function}, \mathrm{seq}\,([\mathit{str}, +\,(\mathit{str})\,, \mathit{expression}]))\\
\mathrm{p}\,(`', \mathit{Program}, *\,(\mathit{Function})) &\Bumpeq \mathrm{p}\,(`', \mathit{program}, +\,(\mathit{function}))\\
\mathrm{p}\,(`', \mathit{Expr_1}, \mathrm{seq}\,([\mathit{str}, *\,(\mathit{Expr})])) &\Bumpeq \mathrm{p}\,(`', \mathit{apply}, \mathrm{seq}\,([\mathit{str}, +\,(\mathit{expression})]))\\
\mathrm{p}\,(`', \mathit{Expr_2}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])) &\Bumpeq \mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))\\
\mathrm{p}\,(`', \mathit{Expr_3}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) &\bumpeq \mathrm{p}\,(`', \mathit{conditional}, \mathrm{seq}\,([\mathit{expression}, \mathit{expression}, \mathit{expression}]))
\end{aligned}
$$

Figure 4: Matching of production rules with the Abstract Normal Form of the JAXB-produced grammar on the left and the master grammar on the right [Zay12k].

3.8.7 Wiki activity

While contributing to wiki websites is not usually considered an activity worthy of tracking or mentioning in the academic sense, of the 72 wiki-articles I have written in 2012 I can identify at least six that can be viewed as (popular) scientific writing:

- Grammar in a broad sense (11 kB + 1 figure)
- Technological space (16 kB)
- Megamodelling (6 kB + 2 figures)
- Island grammar (12 kB)
- Adriaan van Wijgaarden (21 kB)
- Ninomiya Sontoku (Kinjiro) (10 kB)

3.8.8 Colloquium organisation

Again, participating in organisation of various events is commonly considered normal for a practicing academic researcher, but is never counted as a scientific contribution. Not arguing with that, I am still happy to be able to maintain the existing seminar culture of CWI (Centrum Wiskunde & Informatica, my current employer) as a colloquium organiser of a series of events that have been taken place continuously at least since 1997[2]. Over the course of 2012, **56** presentations were given in total as a part of Programming Environment Meeting (PEM, mostly an inter-institutional outlet), Software Engineering Meeting (SEM, mostly an internal group seminar) and a special one-day event Symposium on Language Composability and Modularity (SLaC'M, most trouble of organising which was taken by Tijs van der Storm). These speakers have appeared at PEM, SEM and SLaC'M in 2012 (in chronological order of their first appearance):

- Dr. Vadim Zaytsev [Zay12l; Zay12c; Zay12ag; Zay12e; Zay12f]
- Atze van der Ploeg [Plo12b; Plo12a]
- Prof. Dr. Serge Demeyer [Dem12]
- Dr. Alexander Serebrenik [Ser12]
- Stella Pachidi [Pac12]
- Dr. Tijs van der Storm [Sto12a; Sto12c; Sto12b]
- Michael Steindorfer [Ste12a; Ste12b; Ste12c]
- Dr. Antony Sloane [Slo12]
- Riemer van Rozen [Roz12a; Roz12b]
- Jeroen van den Bos [Bos12b; LB12; Bos12a]
- Alex Loh [Loh12b; Loh12c; Loh12a]
- Dr. Daniel M. German [Ger12]
- Dr. Michael Godfrey [God12]
- Dr. Mark Hills [Hil12c; Hil12b; Hil12a]
- Davy Landman [LB12; Lan12a]
- Luuk Stevens [Ste12d]
- Dr. Krzysztof Czarnecki [Cza12]
- Prof. Dr. Magne Haveraaen [HB12; Hav12]
- Dr. Anya Helene Bagge [HB12; Bag12]
- Dr. Sunil Simon [Sim12]
- Dr. T. B. Dinesh [Din12]
- Dr. Jurgen Vinju [Vin12a; Vin12b]
- Dr. William R. Cook [Coo12]
- Anastasia Izmaylova [Izm12]
- Dr. Lennart Kats [Kat12]
- Carel Bast, Wim Bast, Tom Brus [BBB12]
- Tesfahun Tesfay [Tes12]
- Dr. William B. Langdon [Lan12b]
- Andrei Varanovich [Var12]
- Dr. Joris Dormans [Dor12]
- Sebastiaan Joosten [Joo12]
- Dr. Magiel Bruntink [Bru12]
- Dr. Patricia Lago [Lag12]
- Prof. Dr. Frank Tip [Tip12]
- Dr. Raphael Poss [Pos12]
- Arjan Scherpenisse [Sch12]

[2] http://event.cwi.nl/pem

4 Venues

Academic venues (mostly conferences, workshops and journals) are essential components of the research process: publishing there means community recognition; submitting eventually leads to receiving peer reviews; and even reading calls for papers can be very inspiring and eye-opening. Below I list two kinds of venues that contributed to my research in 2012: one list is for those where I have submitted, the other one for the rest — I am deeply grateful to all the reviewers and organisers of both kinds. The lists are not meant to cover all possible venues for my field, just those directly relevant to my activities this year.

4.1 Exercised venues

BX 2012 (ETAPS workshop)
I have been a "*bx-curious*" person for quite a while, but BX 2012 was my first venue to come out. A very inspiring call for papers[3], excellent atmosphere during the workshop, friendly and productive reviewers. A typical example of an event that appreciates you preparing a dedicated paper for which this becomes the one and only target venue. I submitted against all the odds (December deadlines are rather stressful), got there against all the odds (had to fly from ETAPS to SAC and then back) and still regretted nothing. I will not attend BX 2013 (my grandmother has her 80th birthday on the day of the workshop, and one has to set priorities), but I would if I could. Definitely recommended for people at least marginally interested in this field [Cza+09].

SAC 2012 (PL track)
A yet another experimental submission in the sense that I did not know almost anyone from the programme committee at that moment. However, I know people from my technological space who published there, and the call for papers[4] was inspiring, so I gave it a try, and did not regret it. The whole conference is huge, so I was afraid that attending would be unproductive, but I was proven wrong: if you know at least a couple of people with similar research interests and stick to them all the time, you will find many other similar researchers to talk to. I did not submit anything to SAC 2013 due to bad planning (holidays right before the deadline are unproductive), but I definitely will consider it very seriously every year from now on.

LDTA 2012 (ETAPS worshop)
Trying to be a good programme committee member, I knew I have to attend, so I have submitted the best result of 2011 there: the Grammar Hunter. I was also pleased to see how the current call for papers[5] positioned LDTA as "SLE, but with more grammarware". The future of LDTA remains to be determined, but it has departed from ETAPS and will most probably join forces with SLE.

ECMFA 2012
The call for papers[6] made it look like I have a chance, so I submitted something that I believed to be of good quality and of possible interest to the modelware researchers. One of the reviewers said that the paper "clearly makes the most contribution of any paper I read", which was rather encouraging, but ended up with rejection. In the end, I must conclude that I should have devoted this time to writing for ICPC or one of the journal special issues with deadlines around early spring.

TFP 2012
The call for papers[7] looked challenging, but I really liked the "trends" aspect of it, since most traditional conferences dislike overview papers unless they are extremely strong and retrospective: there is simply no place for overviews of the current trends, unless you are already in the field and you systematically explore the "future work" sections of all papers you come by. In contrast to BX, this was an example of a venue that did not appreciate preparing a paper specifically for them on a topic relevant to me. In less than two weeks after submission I have received a short notification that it was judged to be out of scope. This was obviously not the only reason since other (stronger, less "trendy") papers from my technological space like [SS12] were accepted, so I can only conclude that I have failed to explain the link between grammar transformation and the functional programming paradigm properly. Given the fact that I am not qualified to report on "trends" in any other field, I doubt that I will try sending anything to this venue in the future, but I surely do not discourage others to do so. Personally for me, it would have been more more productive to pursue MoDELS which had a competing deadline this year.

JUCS (journal)
The call for papers[8] made it clear that this special issue is linked to a workshop where I did not participate, but the call was open, and I answered. I cannot say that that was very appreciated: the reviews for [Zay12ae] came very late

[3] http://www.program-transformation.org/BX12
[4] http://www.cis.uab.edu/bryant/sac2012
[5] http://ldta.info/ldta_2012_cfp.pdf
[6] http://www2.imm.dtu.dk/conferences/ECMFA-2012/contributions/?page=cfp
[7] http://www-fp.cs.st-andrews.ac.uk/tifp/TFP2012/TFP_2012/CFP12.txt
[8] http://www.jucs.org/ujs/jucs/info/special_issues/sbcars_cfp.pdf

(several months after the notification deadline), were extremely short and discouraging.

SCAM 2012
This is the third time I have served as a programme committee member for SCAM, where I have been invited after our paper with Ralf Lämmel got a best paper award in 2009 [LZ09b]. I have never attended since that time, and received a warning that I will not be included next year if I miss the event again. So, putting date-conflicting events like SLE and CSMR aside, I did my best, which for me meant submitting one paper to SCAM and one to the colocated ICSM (see below). The topic chosen for SCAM (island grammars) seemed to be in scope of the call for papers[9], but the paper was seen as weird and immature, and was hopelessly rejected. The reviews it received were pretty helpful, even though one of the reviewers really hated the "in the cloud" aspect (and that is exactly how I tried to sell it). Apparently, putting some effort into submitting something has already been noticed, since I have been, against all the odds, invited to the programme committee again for SCAM 2013.

ICSM 2012
The call for papers[10] came to my attention right after the rejection letter from ECMFA, and I decided that ICSM would be a good venue for the guided grammar convergence methodology (§3.1). Getting a paper there would also increase my chances at going to SCAM (see above for the reasons). Reviews were rather cold, but some of them (except one) useful nonetheless.

NordiCloud (WICSA/ECSA workshop)
Not really being an architecture researcher, I would have never considered going to WICSA/ECSA, but the call for contributions[11] was out precisely a couple of weeks after my SCAM rejection, and I was not feeling enough energy to rewrite the island parsing paper completely, so NordiCloud was a relatively cheap way for me to resubmit the same material after a minor revision. It did not pay off: most of the reviewers were scared off just by seeing a grammar-related submission.

FSE 2012 (NIER track)
The call for papers[12] came out at a very busy time, but four page limit was easily reachable, so I have submitted two papers on different new ideas. Unfortunately, they were indeed more of idealistic proposals for discussing and considering certain aspects, than usual "short papers" that are just normal papers at the early stage. Both were hopelessly rejected, and I still want to find some venue for the future that would be good for sharing and discussing fresh ideas — perhaps OBT? I have to try to find out.

SoTeSoLa 2012 (summer school)
An experiment in "Research 2.0" driven mostly by Jean-Marie Favre and Ralf Lämmel, this summer school was by far not a typical one. There was a lot of innovations: submitting a one-page profile of yourself, making a one-minute video about yourself, listening to lots of remote lectures, having a hackathon distributed in time and space, registering at a social networking website, etc. Not all of them very entirely successful: partly due to being ahead of its time, partly due to other reasons, which are being dissected, analysed and researched now by Jean-Marie Favre. I was involved in all kinds of activities from the relatively early stage, and in the end it was officially classified as serving as a "Social Media Chair" and a "Hackathon Lead Coordinator". This was not a publishing venue, and I did not give any invited lecture, but it was fun to be a part of it.

SATToSE 2012 (seminar)
A non-publishing seminar series where I have given a presentation on bidirectional grammar transformation [Zay12b]. The material presented there was in a state somewhere between [Zay12r] and the planned future paper on bidirectionalisation.

POPL 2013
The call for papers[13] was concise and crunchy, but POPL is one of the venues that does not require much advertisement. I have poured a lot of effort into [Zay12j], completely redesigned the convergence process (see §3.1), reimplemented the prototype and rewritten the paper with respect to [Zay12h; Zay12i]. In a way, it did pay off: the paper was rejected, but the reviews were among the most useful that I have received this year.

NWPT 2012
The call for papers[14] was brought to my attention by Anya Helene Bagge, a co-organiser of this workshop. In an extended abstract that was submitted there, I apparently went overboard with the required abstraction level and assumed level of grammatical knowledge, and recent POPL rejection has possibly jeopardised the outsourcing of the usefulness statement of the method. Reviews were curt and bleak.

[9] http://scam2012.cs.usask.ca/CFP
[10] http://selab.fbk.eu/icsm2012/download/cfp-icsm2012.pdf
[11] http://46.22.129.68/NordiCloud/?page_id=39
[12] http://www.sigsoft.org/fse20/cfpNewIdeas.html
[13] http://popl.mpi-sws.org/2013/popl2013-cfp.pdf
[14] http://nwpt12.ii.uib.no/call-for-papers

EMSE (journal)
The call for papers[15] called for "experimental replications" and went to great length explaining how important it is to be able to publish not just the experiments themselves, but also replications thereof. I was immediately convinced, but decided to reinterpret the definition of a replication. Instead of doing classical empirical studies, I presented research activities (and in particular prototype engineering) as experiments. That way, the replications were also "experiments" in that sense that were intended to cover an older experiment and could therefore be measured and assessed based on grounds of that coverage. I could even find some related work on the topic in form of papers that described the prototype development process itself. My paper was intended to contain three case studies: (1) replicating the grammar convergence case study of the Factorial Language from [LZ09a] with the guided grammar convergence methodology (see §3.1); (2) replicating a bigger grammar convergence case study of Java from [LZ11] with more abstract and concise Extended XBGF (see §3.2.4); (3) replicating both of these case studies with a bidirectional ΞBGF (see §3.2.2). Due to insane amounts of work that this turned out to be, only the first two replications have made it into a 42-page long paper [Zay12m]. Only one of three reviewers was excited by my approach, and all three agreed that the empirical software engineering journal is not the right venue for such a report.

MPM 2012 (MoDELS worshop)
Basically, this venue was chosen *after* I have written a paper. The text underwent some polishing after the choice was made, but the topic was not adjusted. I have had a nice idea of transforming megamodels in order to make a good story out of them (see §3.5): a substantial contribution was not yet there (and such work is still ongoing), but I wanted to expose it to the public and to discuss it first. The call for papers[16] for MPM looked the most inviting for this kind of cross-paradigm approach among all MoDELS workshops, and indeed the reviewers found the paper weird yet acceptable, so I was able to give a short presentation and hang my poster there [Zay12aa].

XM 2012 (MoDELS worshop)
The topics list[17] provided by the organisers of this workshop was fascinating, and I desperately wanted to submit anything, but eventually gave up to find the time it deserved. Soon after that, the deadline was extended, and I had no other choice than to write down the idea that was floating around in my head for a while (see §3.2.3).

SCP (journal)
The call for systems[18] was very much in sync with what its guest editors have tried to achieve in the last years, and I support them wholeheartedly in that. The Grammar Zoo, one of essential parts of the SLPS [Zay+08], that did not receive a lot of my attention in 2012, but that was always on my mind, was packaged and submitted there both as an available system and as an important repository of experimental systems in grammarware. The outcome will become known in 2013.

4.2 Inspiring venues

There have been many venues that I did not submit anything to, but not because I did not want to. Their calls for papers gave me inspiration to work on something, even though I was not productive enough to be able to fit into their deadlines or produce anything of value at the required level.

MSR 2012
The mining challenge[19] of MSR looked very interesting, so I looked at it, but since I was looking specifically for grammars, it did not work out at all: only two grammars were found, and there was no sensible way to connect them to the rest of the system. If more of them could have been obtained written in a variety of EBNF dialects, it could have become an interesting case study similar to [Zay12ae].

Laws and Principles of Software Evolution
The call for papers[20] for this special issue of JSME looked tempting, so I even emailed the editors, asking for some additional information. Unfortunately, the collaboration that I hoped to achieve with other people, did not work out, and nothing was produced in time.

Success Stories in Model Driven Engineering
The call for papers[21] came out at the time when I was busy with all kinds of other initiatives. Besides that, this special issue of SCP was actually looking for extended reports on already published projects, and I was busy with new experiments. Possibly, a strong "lessons learnt" kind of paper on grammar hunting would make sense, but I was too immersed in new stuff at the time to go back. However, I have to admit that when/if I finally sit down to write a comprehensive grammar recovery paper (i.e., connecting §§3.3.3, 3.8.1, 3.8.6 and 3.6), it must go to either SCP or SP&E.

[15] http://sequoia.cs.byu.edu/lab/?page=reser2013&section=emseSpecialIssue
[16] http://avalon.aut.bme.hu/mpm12/MPM12-CFP.pdf
[17] http://www.di.univaq.it/XM2012/page.php?page_id=13
[18] http://www.win.tue.nl/~mvdbrand/SCP-EST
[19] http://2012.msrconf.org/challenge.php
[20] http://listserv.acm.org/scripts/wa-acmlpx.exe?A2=ind1111&L=seworld&F=&S=&P=25841
[21] http://www.di.univaq.it/ssmde

CloudMDE (ECMFA 2012 workshop)
This was *the* venue that gave me the eerie thought of writing a "parsing in the cloud" paper (see §3.8.2). However, I was disheartened by the rejection of [Zay12h] at ECMFA and decided to not submit anything to ECMFA workshops[22].

ICPC 2012
The call for papers[23] competes date-wise with many other good venues, so this year ICPC just happened to not be among the ones I have chosen as my targets.

RC 2012
The call for papers[24] for the fourth workshop on reversible computation gave me a lot of ideas and keyword pointers for the bidirectionality topic. However, I did not feel confident enough to submit anything. Anyway, thanks a lot and congratulations on becoming a conference in 2013!

CSCW 2013
This is not a typical venue for me, but I have a dream of eventually submitting something wiki-related there. The call for participation[25] was as good as it always is, and even better this year because they have introduced a new rule concerning the paper size: 10 pages is no longer the *limit*, it is rather a *standard*. If your idea fits on smaller number of pages, your reviewers have the right to complain if you try to bloat your submission. On the other hand, if that is not enough, you can always make your paper longer, but the contribution then needs to grow accordingly. I believe that with small incremental and non-disruptive ideas like these, we could achieve modern comfortable publishing models easier than with endeavours to revolutionise the field.

PPDP/LOPSTR 2012
The calls for papers[26,27] were both interesting, but at my level I could not actually decide between the two venues. I was working honestly toward the seemingly achievable goal (see §3.2.5), but it turned out to be unachievable. Being insecure about my ability to write a strong paper about negative results, I gave up.

FM+AM 2012 (SEFM workshop)
This was *the* workshop[28] that set my thoughts in the agile/extreme mode, which ultimately led to the paper at XM (see §3.2.3) simply because I did not manage to complete the work before the FM+AM deadline. Imagine my surprise when I found out that FM+AM was cancelled due to the lack of good submissions!

WoSQ 2012 (FSE workshop)
The call for papers[29] has led me to believe that this would be a good possible venue for the paper on grammar mutations (see §3.8.1). However, the time was too tight, and both of my NIER submissions have been rejected, so an FSE workshop stopped looking that attractive after all.

SQM 2012 (CSMR workshop)
The workshop[30] happened at the same time as I was attending both ETAPS and SAC, so I could not possible be at the third place at the same time as well, but I just want to name it as a relatively small venue where I have enjoyed reviewing a couple of papers as a PC member (will be on PC next year as well).

WCN 2012
The website[31] is in Dutch, as the conference itself. This was my second experience being a Program Chair (the first one was with WCN 2011), and this time I counted: 842 emails needed to be sent or answered by me in order for this conference to happen. Luckily, CWI (my employer) did not mind since they could proudly list "one of theirs" to be the PC at a venue where one of the keynote speakers is Jimmy Wales [Wal12].

5 Concluding remarks

5.1 Immediate results

Writing this extended year report has achieved at least three goals:

Streamlining new ideas. Reexplaning (renarrating?) research ideas and putting them in perspective has helped to crystallise them into publishable achievable objectives.

Knowledge dissemination. This document can serve both as a scientific report for my colleagues and superiors, and as an entrance point for people who want to get acquainted with my results for other reasons.

Case study in self-archiving. As I have said in the introduction, this report can be seen as advanced form of self-archiving. It was a relatively big effort, compared to the traditional "just put the PDF online" thing. Together with the open notebook initiative, it stressed the paradigm and

[22] V. Zaytsev (grammarware). "Yet another bridging attempt failed: my grammar paper got rejected at @ecmfa2012. Now I will also go submit the #CloudMDE draft elsewhere." Tweet. https://twitter.com/grammarware/status/189976445995593728. 11 April 2012, 9:21.
[23] http://icpc12.sosy-lab.org/CfP.pdf
[24] http://www.reversible-computation.org/2012/index91b1.html?call_for_papers
[25] http://cscw.acm.org/participation_paper.html
[26] http://dtai.cs.kuleuven.be/events/PPDP2012/ppdp-cfp.txt
[27] http://costa.ls.fi.upm.es/lopstr12/cfp.pdf
[28] http://ssfm.cs.up.ac.za/workshop/FMAM12.htm
[29] http://sites.google.com/site/wosq2012/cfp
[30] http://sqm2012.sig.eu
[31] http://www.wikimediaconferentie.nl

raised some questions yet to be answered (e.g.: How to properly break one atomic SKO — essentially, a publication — into subatomic ones to distinguish "I've done for the tool that was later described in this paper" from "I've done this for the particular version of this paper, which was later rejected"? What are all possible stages in the SKO lifecycle?).

5.2 Special features

The presence of an open notebook. A lot of claims about dates, continuations and amounts of effort, made on the pages of this report, can be reformulated into queries on the open notebook, and formally validated as such. For now these claims were intentionally done in plain text because no reliable or traditionally acceptable infrastructure exists for them (yet).

Open access window. All the papers mentioned here, were put online immediately after their submission (unless prohibited explicitly by the submission rules), and taken down immediately after their rejection (if any). At this stage, I do not know any better way to expose your research results to the public: official acceptance can take months and years, during which one could have profited from sharing the contents around.

Rejected material. Not all rejected papers are rejected because they are inherently, irreparably bad: some turn out to be out of scope, lacking some essential results or simply not mature enough to be published (yet). With this report, I have exposed most of the dark data concerning my rejected material.

Unfruitful attempts. Also classified as dark data by [Goe07], but of an entirely different nature: these are failed experiments: prototypes that have never made it to the point of being ready to be described in a paper. There can be traces of such unfruitful attempts in presentations and other subatomic SKOs before their futility becomes apparent.

Venues. Knowledge about workshops, conferences and journals seems to float around in the academic community and is usually distributed as folklore, if at all. There are many reasons for doing so, ranging from the lack of incentive to the fear of occasional offence.

5.3 Acknowledgements

I am most grateful to the following people, without whom 2012 would not have been the same:

- Ronald de Wolf, Bas de Lange, Tijs van der Storm, Jean-Marie Favre — for inviting me to present at various events.
- Ralf Lämmel — for inviting me for a working visit to Koblenz and hosting me there.
- Felienne Hermans — for referring me to the matrix grammars, picture calculus and adjacent topics.
- Paul Klint, Jurgen Vinju, Tijs van der Storm, Sander Bohte — for contributing to organising the PEM Colloquium.
- T. B. Dinesh — for introducing me to the notion of renarration.
- Frans Grijzenhout and Sandra Rientjes — for coorganising a conference with me.
- Lodewijk Gelauff — for introducing me to the topic of structured data extraction.
- Mark van den Brand — for appreciating the input in the discussion on the future of LDTA, and for putting trust in me for chairing its programme next year.
- Jurgen Vinju, Tijs van der Storm — for collaborating on a grammar-related topic.
- Ralf Lämmel, Andrei Varanovich, Jean-Marie Favre — for collaborating on megamodelling topics.
- Zinovy Diskin — for insightful discussions during ETAPS'12 and MoDELS'12 on category theory and grammars in a broad sense.
- Anonymous reviewers of BX'12, SAC'12, LDTA'12, ECMFA'12, ICSM'12, SCAM'12, NordiCloud'12, FSE NIER'12, XM'12, MPM'12, NWPT'12, POPL'13, JUCS, EMSE and NWO Veni programme — for the effort that they have put into considering, assessing and reviewing my work.
- All presenters at PEM and WCN — for their contributions, the only essential component of the final product.
- All unmentioned colleagues and uncounted conference contacts — for fruitful discussions.
- All people who tried to contact me through email — for patience and unreasonable waiting times.

References

[AD89] H. Abramson and V. Dahl. *Logic Grammars*. Symbolic Computation: Artificial Intelligence. Springer, 1989, pp. I–XIV, 1–234. ISBN: 978-0-387-96961-9.

[Aho68] A. V. Aho. "Indexed Grammars. An Extension of Context-Free Grammars". In: *Journal of the ACM* 15.4 (Oct. 1968), pp. 647–671. ISSN: 0004-5411. DOI: 10.1145/321479.321488.

[ASU85] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1985.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1972. ISBN: 0-13-914556-7.

[Ajd35] K. Ajdukiewicz. "Die syntaktische Konnexität". In: *Studia Philosophica* 1 (1935), pp. 1–27. URL: http://www.ifispan.waw.pl/studialogica/s-p-f/volumina_i-iv/I-03-Ajdukiewicz-small.pdf.

[Alb+94] H. Alblas, R. op den Akker, P. Oude Luttighuis, and K. Sikkel. "A Bibliography on Parallel Parsing". In: *SIGPLAN Notices* 29.1 (Jan. 1994), pp. 54–65. ISSN: 0362-1340.

[AMUFVI09] F. J. Almeida-Martínez, J. Urquiza-Fuentes, and J. Á. Velázquez-Iturbide. "Visualization of Syntax Trees for Language Processing Courses". In: *Journal of Universal Computer Science* 15.7 (Apr. 2009), pp. 1546–1561.

[And91] J. Anderson. "Notional Grammar and the Redundancy of Syntax". In: *Studies in Language* 15.2 (1991), pp. 301–333. URL: http://www.fb10.uni-bremen.de/linguistik/dpng/pdf/Anderson1991NotionalGrammar.pdf.

[ABS11] B. J. Arnoldus, M. G. J. van den Brand, and A. Serebrenik. "Less Is More: Unparser-Completeness of Metalanguages for Template Engines". In: *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering*. GPCE '11. ACM, 2011, pp. 137–146. ISBN: 978-1-4503-0689-8. DOI: 10.1145/2047862.2047887.

[Bac60] J. W. Backus. "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference". In: *Proceedings of the International Conference on Information Processing*. Ed. by S. de Picciotto. Unesco, Paris, 1960, pp. 125–131. URL: http://www.softwarepreservation.org/projects/ALGOL/paper/Backus-Syntax_and_Semantics_of_Proposed_IAL.pdf.

[BH82] D. T. Barnard and R. C. Holt. "Hierarchic Syntax Error Repair for LR Grammars". In: *International Journal of Computer and Information Sciences* 11.4 (1982), pp. 231–258.

[Bas10] H. J. S. Basten. "Tracking Down the Origins of Ambiguity in Context-free Grammars". In: *Proceedings of the Seventh International colloquium conference on Theoretical Aspects of Computing*. ICTAC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 76–90.

[Bas11] H. J. S. Basten. "Ambiguity Detection for Programming Language Grammars". PhD thesis. Universiteit van Amsterdam, Dec. 2011.

[Bat10] F. Batarseh. "Java Nano Patterns: a Set of Reusable Objects". In: *Proceedings of the 48th Annual Southeast Regional Conference*. ACM SE '10. Oxford, Mississippi: ACM, 2010, 60:1–60:4. ISBN: 978-1-4503-0064-3.

[BN05] Eberhard Bertsch and Mark-Jan Nederhof. "Gap Parsing with LL(1) Grammars". In: *A Journal of Mathematical Research on Formal and Natural Languages* 8 (2005), pp. 1–16.

[BJV04] J. Bézivin, F. Jouault, and P. Valduriez. "On the Need for Megamodels". In: *OOPSLA & GPCE, Workshop on best MDSD practices* (2004). URL: http://www.softmetaware.com/oopsla2004/bezivin-megamodel.pdf.

[BAF97] R. Bolle, Y. Aloimonos, and C. Fermüller. "Toward motion picture grammars". In: *Computer Vision ACCV'98*. Ed. by Roland Chin and Ting-Chuen Pong. Vol. 1352. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1997, pp. 283–290. ISBN: 978-3-540-63931-2. DOI: 10.1007/3-540-63931-4_228.

[Bos+12] J. van den Bos, D. Landman, K. Blankers, P. Klint, A. van der Ploeg, T. van der Storm, J. Vinju, and V. Zaytsev. *LEGO Turing Machine*. http://www.legoturingmachine.org. 2012.

[Böt01] M. Böttner. "Peirce Grammar". In: *Grammars* 4 (1 2001), pp. 1–19. ISSN: 1386-7393. DOI: 10.1023/A:1011403527615.

[Bou98] P. Boullier. *Proposal for a Natural Language Processing Syntactic Backbone*. Research Report RR-3342. INRIA, 1998. URL: http://hal.inria.fr/inria-00073347/PDF/RR-3342.pdf.

[BSV97] M. G. J. van den Brand, M. P. A. Sellink, and C. Verhoef. "Obtaining a COBOL Grammar from Legacy Code for Reengineering Purposes". In: *Proceedings of the Second International Workshop on the Theory and Practice of Algebraic Specifications*. Ed. by M. P. A. Sellink. Berlin: Springer-Verlag, 1997, pp. 6–17. URL: http://www.cs.vu.nl/~x/coboldef/coboldef.html.

[BSV98] M. G. J. van den Brand, M. P. A. Sellink, and C. Verhoef. "Current Parsing Techniques in Software Renovation Considered Harmful". In: *Proceedings of the International Workshop on Program Comprehension (IWPC)*. 1998.

[BSV00] M. G. J. van den Brand, M. P. A. Sellink, and C. Verhoef. "Generation of Components for Software Renovation Factories from Context-Free Grammars". In: *Science of Computer Programming* 36.2–3 (2000), pp. 209–266.

[Bra+02] M. G. J. van den Brand, J. Scheerder, J. J. Vinju, and E. Visser. "Disambiguation Filters for Scannerless Generalized LR Parsers". In: *Compiler Construction 2002 (CC 2002)*. Ed. by N. Horspool. 2002, pp. 143–158.

[Bra61] H. Bratman. "An Alternate Form of the "UNCOL Diagram"". In: *Communications of the ACM* 4.3 (Mar. 1961). DOI: 10.1145/366199.366249.

[Cam10] D. A. O. Camacho. "A Product-Line for Families of Program Translators: A Grammar-Based Approach". PhD thesis. Louvain-la-Neuve, Belgium: UCL Presses Universitaires de Louvain, Aug. 2010. URL: http://dial.academielouvain.be/vital/access/services/Download/boreal:33289/PDF_01.

[Cam+10] D. O. Camacho, K. Mens, M. van den Brand, and J. Vinju. "Automated Generation of Program Translation and Verification Tools Using Annotated Grammars". In: *Science of Computer Programming* 75.1–2 (Jan. 2010), pp. 3–20. DOI: 10.1016/j.scico.2009.10.003.

[Cau07] D. Caucal. "Deterministic Graph Grammars". In: *Logic and Automata: History and Perspectives, Texts in Logic and Games*. Amsterdam University Press, 2007, pp. 169–250. URL: http://www-igm.univ-mlv.fr/~caucal/birthday.pdf.

[Cha70] S.-K. Chang. "The analysis of two-dimensional patterns using picture processing grammars". In: *Proceedings of the second annual ACM symposium on Theory of computing*. STOC '70. Northampton, Massachusetts, USA: ACM, 1970, pp. 206–216. DOI: 10.1145/800161.805166.

[Cho56] N. Chomsky. "Three Models for the Description of Language". In: *IRE Transactions on Information Theory* 2.2 (1956), pp. 113–123. URL: http://www.chomsky.info/articles/195609--.pdf.

[Chr05] H. Christiansen. "CHR Grammars". In: *The International Journal on Theory and Practice of Logic Programming* 5.4–5 (2005). URL: http://arxiv.org/abs/cs/0408027.

[CD09] H. Christiansen and V. Dahl. "Abductive Logic Grammars". In: *Logic, Language, Information and Computation*. Ed. by H. Ono, M. Kanazawa, and R. de Queiroz. Vol. 5514. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, pp. 170–181. ISBN: 978-3-642-02260-9. DOI: 10.1007/978-3-642-02261-6_14.

[Cic+08] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. "Automating Co-evolution in Model-Driven Engineering". In: *12th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2008)*. IEEE Computer Society, 2008, pp. 222–231.

[CH06] A. Cleve and J.-L. Hainaut. "Co-transformations in Database Applications Evolution". In: *Post-proceedings of the First International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'05)*. Vol. 4143. LNCS. Springer, 2006, pp. 409–421.

[Col78] A. Colmerauer. "Metamorphosis Grammars". In: *Natural Language Communication with Computers*. Ed. by Leonard Bolc. Vol. 63. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1978, pp. 133–188. ISBN: 978-3-540-08911-7. DOI: 10.1007/BFb0031371.

[Con58] M. E. Conway. "Proposal for an UNCOL". In: *Communications of the ACM* 1.10 (Oct. 1958), pp. 5–8. DOI: 10.1145/368924.368928.

[Cox03] A. Cox. "Syntactic Approximation Using Iterative Lexical Analysis". In: *Proceedings of the International Workshop on Program Comprehension*. 2003, pp. 154–163.

[Čre+05] M. Črepinšek, M. Mernik, F. Javed, B. R. Bryant, and A. Sprague. "Extracting Grammar from Programs: Evolutionary Approach". In: *SIGPLAN Notices* 40.4 (2005), pp. 39–46. DOI: 10.1145/1064165.1064172.

[Čre+10] M. Črepinšek, T. Kosar, M. Mernik, J. Cervelle, R. Forax, and G. Roussel. "On Automata and Language Based Grammar Metrics". In: *Computer Science and Information Systems* 7.2 (2010).

[CV+95] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, and Gh. Păun. "Grammar Systems: a Grammatical Approach to Distribution and Cooperation". In: *Automata, Languages and Programming*. Ed. by Z. Fülöp and F. Gécseg. Vol. 944. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, pp. 429–443. ISBN: 978-3-540-60084-8. DOI: 10.1007/3-540-60084-1_94.

[Cza+09] Krzysztof Czarnecki, J. Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James Terwilliger. "Bidirectional Transformations: A Cross-Discipline Perspective". In: *Theory and Practice of Model Transformations*. Ed. by Richard Paige. Vol. 5563. LNCS. Springer Berlin / Heidelberg, 2009, pp. 260–283. ISBN: 978-3-642-02407-8.

[DTL97] V. Dahl, P. Tarau, and R. Li. "Assumption Grammars for Processing Natural Language". In: *Proceedings of the Fourteenth International Conference on Logic Programming*. MIT Press, 1997, pp. 256–270. URL: http://reference.kfupm.edu.sa/content/a/s/assumption_grammars_for_processing_natur_114394.pdf.

[DFP95] J. Dassow, R. Freund, and Gh. Păun. "Cooperating Array Grammar Systems". In: *International Journal of Pattern Recognition and Artificial Intelligence* 9.6 (1995), pp. 1029–1053. DOI: 10.1142/S0218001495000420.

[Dea+03] T. R. Dean, J. R. Cordy, A. J. Malton, and K. A. Schneider. "Agile Parsing in TXL". In: *Journal of Automated Software Engineering* 10.4 (2003), pp. 311–336.

[DeR74] F. DeRemer. "Transformational Grammars". In: *Compiler Construction*. Ed. by F. Brauer, J. Eickel, F. De Remer, M. Griffiths, U. Hill, J. Horning, C. Koster, W. McKeeman, P. Poole, and W. Waite. Vol. 21. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1974, pp. 121–145. ISBN: 978-3-540-06958-4. DOI: 10.1007/3540069585_50.

[DK99] A. van Deursen and T. Kuipers. "Building Documentation Generators". In: *Proceedings of the 15th IEEE International Conference on Software Maintenance (ICSM'99)*. 1999, pp. 40–49. URL: http://www.st.ewi.tudelft.nl/~arie/papers/icsm99.pdf.

[D'H+01] T. D'Hondt, K. De Volder, K. Mens, and R. Wuyts. "Co-evolution of Object-Oriented Software Design and Implementation". In: *Software Architectures and Component Technology*. Ed. by M. Akşit. Proceedings of SACT 2000. Kluwer Academic Publisher, Jan. 2001, pp. 207–224. ISBN: 0792375769.

[Dre+01] F. Drewes, S. Ewert, R. Klempien-Hinrichs, and H.-J. Kreowski. "Computing Raster Images from Grid Picture Grammars". In: *Implementation and Application of Automata*. Ed. by S. Yu and A. Păun. Vol. 2088. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 113–121. ISBN: 978-3-540-42491-8. DOI: 10.1007/3-540-44674-5_9.

[FLV12] J.-M. Favre, R. Lämmel, and A. Varanovich. In: *Proceedings of the ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2012)*. Ed. by R. B. France, J. Kazmeier, R. Breu, and C. Atkinson. Vol. 7590. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 151–167. ISBN: 978-3-642-33665-2. DOI: 10.1007/978-3-642-33666-9_11. URL: http://softlang.uni-koblenz.de/mega.

[FN04] J.-M. Favre and T. NGuyen. "Towards a Megamodel to Model Software Evolution through Transformations". In: *Electronic Notes in Theoretical Computer Science, Proceedings of the SETra Workshop* 127.3 (2004). DOI: 10.1016/j.entcs.2004.08.034.

[Fer+02] M. Fernández, Y. Kadiyska, D. Suciu, A. Morishima, and W.-Ch. Tan. "SilkRoute: A Framework for Publishing Relational Data In XML". In: *ACM Transactions on Database Systems (TODS)* 27.4 (Dec. 2002), pp. 438–493. ISSN: 0362-5915. DOI: 10.1145/582410.582413.

[FV99] Joe Fialli and Sekhar Vajjhala. *Java Specification Request 31: XML Data Binding Specification*. 1999. URL: http://jcp.org/en/jsr/detail?id=031.

[FLZ12] B. Fischer, R. Lämmel, and V. Zaytsev. "Comparison of Context-free Grammars Based on Parsing Generated Test Data". In: *Post-proceedings of the Fourth International Conference on Software Language Engineering (SLE 2011)*. Ed. by U. Aßmann and A. Sloane. Vol. 6940. LNCS. Springer, Heidelberg, 2012, pp. 324–343. DOI: 10.1007/978-3-642-28830-2_18. URL: http://dx.doi.org/10.6084/m9.figshare.91426.

[For04] B. Ford. "Parsing Expression Grammars: a Recognition-Based Syntactic Foundation". In: *Proceedings of the Symposium on Principles of Programming Languages*. Jan. 2004. URL: http://pdos.csail.mit.edu/papers/parsing:popl04.pdf.

[GM77] L. Geurts and L. Meertens. "Keyword Grammars". In: *Proceedings of the Fifth Annual International Conference on the Implementation and Design of Algorithmic Languages*. Ed. by J. André and J.-P. Banâtre. Guidel, 1977, pp. 1–12.

[GM05] J. Gil and I. Maman. "Micro Patterns in Java Code". In: *Proceedings of OOPSLA'05*. San Diego, CA, USA: ACM, 2005, pp. 97–116.

[Giu+10] F. Giunchiglia, H. Xu, A. Birukou, and R. Chenu. "Scientific Knowledge Object Patterns". In: *Proceedings of the 15th European Conference on Pattern Languages of Programs*. EuroPLoP '10. Irsee, Germany: ACM, 2010, 15:1–15:6. ISBN: 978-1-4503-0259-3. DOI: 10.1145/2328909.2328928.

[Goe07] Th. Goetz. "Freeing the Dark Data of Failed Scientific Experiments". In: *Wired Magazine* 15.10 (2007). URL: http://www.wired.com/science/discoveries/magazine/15-10/st_essay.

[Gra07a] B. Graaf. "Model-Driven Evolution of Software Architectures". In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR'07)*. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 357–360. DOI: 10.1109/CSMR.2007.35.

[Gra07b] B. Graaf. "Model-Driven Evolution of Software Architectures". PhD thesis. Delft University of Technology, 2007. URL: http://www.st.ewi.tudelft.nl/~arie/phds/Graaf.pdf.

[Gre96] U. Grenander. *Elements of Pattern Theory: A Catalogue Raisonne*. Johns Hopkins Series in the Mathematical Sciences. John Hopkins University Press, 1996. ISBN: 9780801851889. URL: http://books.google.co.uk/books?id=1xBDcFvFFvsC.

[Gru71] J. Gruska. "Complexity and Unambiguity of Context-free Grammars and Languages". In: *Information and Control* 18.5 (1971), pp. 502–519. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(71)90519-5.

[Har01] S. Harnad. "The Self-Archiving Initiative". In: *Nature* 410.6832 (2001), pp. 1024–1025. ISSN: 0028-0836. DOI: 10.1038/35074210.

[Hil+10] R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione. "Realizing Architecture Frameworks Through Megamodelling Techniques". In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10)*. Antwerp, Belgium: ACM, 2010, pp. 305–308. ISBN: 978-1-4503-0116-9. DOI: 10.1145/1858996.1859057.

[ISO96] ISO/IEC 14977:1996(E). *Information Technology. Syntactic Metalanguage. Extended BNF*. 1996. URL: http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf.

[JZ12a] D. Jansen and V. Zaytsev, eds. *Post-proceedings of the Sixth Wikimedia Conference Netherlands (WCN 2012)*. In Dutch. ISSN: 2213-9400. Wikimedia Foundation Netherlands, Nov. 2012, pp. 1–16. URL: http://commons.wikimedia.org/wiki/File:WCN_2012_Proceedings.pdf.

[JZ12b] D. Jansen and V. Zaytsev, eds. *Proceedings of the Sixth Wikimedia Conference Netherlands (WCN 2012)*. In Dutch. ISSN: 2212-5485. Wikimedia Foundation Netherlands, Nov. 2012, pp. 1–13.

[Joh75] S. C. Johnson. *YACC—Yet Another Compiler Compiler*. Murray Hill, New Jersey: Computer Science Technical Report 32, AT&T Bell Laboratories, 1975.

[JO04] H. A. de Jong and P.A. Olivier. "Generation of Abstract Programming Interfaces from Syntax Definitions". In: *Journal of Logic and Algebraic Programming* 59.1–2 (2004), pp. 35–61. ISSN: 1567-8326. DOI: 10.1016/j.jlap.2003.12.002.

[Jon02] M. de Jonge. "Pretty-Printing for Software Reengineering". In: *Proceedings of the 18th IEEE International Conference on Software Maintenance (ICSM'02)*. Montreal, Canada: IEEE Computer Society Press, 2002, pp. 550–559. URL: http://reference.kfupm.edu.sa/content/p/r/pretty_printing_for_software_reengineeri_98908.pdf.

[JM01] M. de Jonge and R. Monajemi. "Cost-Effective Maintenance Tools for Proprietary Languages". In: *Proceedings of 17th International Conference on Software Maintenance (ICSM 2001)*. IEEE, 2001, pp. 240–249.

[JKY69] A. K. Joshi, S. R. Kosaraju, and H. Yamada. "String Adjunct Grammars". In: *Proceedings of the 10th Anniversary Symposium on Switching and Automata Theory (SWAT 1969)*. IEEE Computer Society, 1969, pp. 245–262. DOI: 10.1109/SWAT.1969.23.

[JLT75] A. K. Joshi, L. S. Levy, and M. Takahashi. "Tree Adjunct Grammars". In: *Journal of Computer and System Sciences* 10.1 (1975), pp. 136–163. DOI: 10.1016/S0022-0000(75)80019-5.

[Jou+08] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. "ATL: A Model Transformation Tool". In: *Science of Computer Programming* 72.12 (2008). Special Issue on Experimental Software and Toolkits (SCP EST2), pp. 31–39. ISSN: 0167-6423. DOI: 10.1016/j.scico.2007.08.002.

[Kal01] L. Kallmeyer. "Local Tree Description Grammars". In: *Grammars* 4 (2 2001), pp. 85–137. ISSN: 1386-7393. DOI: 10.1023/A:1011431526022.

[Kat+09] L. C. L. Kats, M. de Jonge, E. Nilsson-Nyman, and E. Visser. "Providing Rapid Feedback in Generated Modular Language Environments. Adding Error Recovery to SGLR Parsing". In: *Proceedings of the 24th ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA 2009)*. Ed. by Gary T. Leavens. Orlando, Florida, USA: ACM Press, Oct. 2009. URL: http://swerl.tudelft.nl/twiki/pub/Main/TechnicalReports/TUD-SERG-2009-020.pdf.

[Kel81] A. Kelemenová. "Grammatical Levels of the Position Restricted Grammars". In: *Proceedings on Mathematical Foundations of Computer Science*. Springer-Verlag, 1981, pp. 347–359. ISBN: 3-540-10856-4.

[Kli93] P. Klint. "A Meta-Environment for Generating Programming Environments". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2.2 (1993), pp. 176–201.

[KLV05a] P. Klint, R. Lämmel, and C. Verhoef. "Toward an Engineering Discipline for Grammarware". In: *ACM Transactions on Software Engineering Methodology (TOSEM)* 14.3 (2005), pp. 331–380. URL: http://www.cs.vu.nl/grammarware/agenda.

[KSV11] P. Klint, T. van der Storm, and J. Vinju. "EASY Meta-programming with Rascal". In: *Post-proceedings of the Third International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2009)*. Ed. by J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva. Vol. 6491. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, Jan. 2011, pp. 222–289. URL: http://homepages.cwi.nl/~paulk/publications/rascal-gttse-final.pdf.

[Kli+12] Paul Klint et al. *Rascal Tutor*. http://tutor.rascal-mpl.org. SWAT, CWI. 2012.

[KL03] S. Klusener and R. Lämmel. "Deriving Tolerant Grammars from a Base-line Grammar". In: *Proceedings of the 19th IEEE International Conference on Software Maintenance (ICSM'03)*. Los Alamitos, CA, USA: IEEE Computer Society, Sept. 2003, pp. 179–188. URL: http://homepages.cwi.nl/~ralf/icsm03/.

[KLV05b] S. Klusener, R. Lämmel, and C. Verhoef. "Architectural Modifications to Deployed Software". In: *Science of Computer Programming* 54 (2005), pp. 143–211.

[Knu64] D. E. Knuth. "Backus Normal Form vs. Backus Naur Form". In: *Communications of the ACM* 7.12 (1964), pp. 735–736. DOI: 10.1145/355588.365140.

[Knu90] D. E. Knuth. "The Genesis of Attribute Grammars". In: *WAGA*. Ed. by P. Deransart and M. Jourdan. Vol. 461. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 1990, pp. 1–12. URL: http://www.dcs.warwick.ac.uk/~sk/cs325/gag.pdf.

[Kop97] R. Koppler. "A Systematic Approach to Fuzzy Parsing". In: *Software—Practice & Experience* 27.6 (1997), pp. 637–649.

[Kor11] A. Kornai. "Probabilistic Grammars and Languages". In: *Journal of Logic, Language and Information* 20 (3 2011), pp. 317–328. ISSN: 0925-8531. DOI: 10.1007/s10849-011-9135-z.

[Kos91] C. H. A. Koster. "Affix Grammars for Programming Languages". In: *Attribute Grammars, Applications and Systems*. Ed. by H. Alblas and B. Melichar. Vol. 545. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1991, pp. 358–373. ISBN: 978-3-540-54572-9. DOI: 10.1007/3-540-54572-7_13. URL: http://www.cs.ru.nl/~kees/home/papers/agpl.pdf.

[KBA02] I. Kurtev, J. Bézivin, and M. Akşit. "Technological Spaces: an Initial Appraisal". In: *Proceedings of CoopIS, DOA'2002, Industrial track*. 2002. URL: http://eprints.eemcs.utwente.nl/10206/01/0363TechnologicalSpaces.pdf.

[Lam58] J. Lambek. "The Mathematics of Sentence Structure". In: *The American Mathematical Monthly* 65.3 (Mar. 1958), pp. 154–170. URL: http://aix1.uottawa.ca/~rblute/COURSE2/lambek.pdf.

[Lam08] J. Lambek. "Pregroup Grammars and Chomsky's Earliest Examples". In: *Journal of Logic, Language and Information* 17.2 (2008), pp. 141–160. DOI: 10.1007/s10849-007-9053-2.

[Läm01] R. Lämmel. "Grammar Adaptation". In: *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*. Vol. 2021. LNCS. Springer-Verlag, 2001, pp. 550–570.

[Läm05] R. Lämmel. "The Amsterdam Toolkit for Language Archaeology". In: *Electronic Notes in Theoretical Computer Science (ENTCS)* 137.3 (Sept. 2005). Proceedings of the Second International Workshop on Metamodels, Schemas and Grammars for Reverse Engineering (ATEM'04), pp. 43–55.

[Läm07] R. Lämmel. "Style Normalization for Canonical X-to-O Mappings". In: *Proceedings of the ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation (PEPM'07)*. ACM Press, 2007.

[LS06] R. Lämmel and W. Schulte. "Controllable Combinatorial Coverage in Grammar-Based Testing". In: *Proceedings of the 18th IFIP TC6/WG6.1 International Conference on Testing of Communicating Systems (TestCom'06)*. Vol. 3964. LNCS. Springer, 2006, pp. 19–38.

[LV99] R. Lämmel and C. Verhoef. *Browsable Grammars*. 1999. URL: http://www.cs.vu.nl/grammarware/browsable.

[LV01a] R. Lämmel and C. Verhoef. "Cracking the 500-Language Problem". In: *IEEE Software* (Nov. 2001), pp. 78–88. URL: http://www.cs.vu.nl/grammarware/500/.

[LV01b] R. Lämmel and C. Verhoef. "Semi-automatic Grammar Recovery". In: *Software—Practice & Experience* 31.15 (Dec. 2001), pp. 1395–1438. URL: http://www.cs.vu.nl/grammarware/ge/.

[LZ09a] R. Lämmel and V. Zaytsev. "An Introduction to Grammar Convergence". In: *Proceedings of the Seventh International Conference on Integrated Formal Methods (iFM 2009)*. Ed. by M. Leuschel and H. Wehrheim. Vol. 5423. LNCS. Berlin, Heidelberg: Springer-Verlag, Feb. 2009, pp. 246–260. DOI: 10.1007/978-3-642-00255-7_17. URL: http://grammarware.net/writes/\#Convergence2009.

[LZ09b] R. Lämmel and V. Zaytsev. "Recovering Grammar Relationships for the Java Language Specification". In: *Proceedings of the Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2009)*. Best Paper Award. Edmonton, Canada: IEEE, Sept. 2009, pp. 178–186. DOI: 10.1109/SCAM.2009.29. URL: http://grammarware.net/writes/\#JLS-SCAM2009.

[LZ11] R. Lämmel and V. Zaytsev. "Recovering Grammar Relationships for the Java Language Specification". In: *Software Quality Journal (SQJ)* 19.2 (Mar. 2011), pp. 333–378. ISSN: 0963-9314. DOI: 10.1007/s11219-010-9116-5. URL: http://arxiv.org/abs/1008.4188.

[Läm08] Ralf Lämmel. *Software Language Processing Samples (SLPS)*. MSDN Blog post on *Grammarware, Haskellware, XMLware*. May 2008. URL: http://blogs.msdn.com/b/ralflammel/archive/2008/05/23/software-language-processing-samples-slps.aspx.

[Lan88] B. Lang. "Parsing Incomplete Sentences". In: *Proceedings of the 12th Conference on Computational linguistics, Volume 1*. COLING '88. Budapest, Hungry: Association for Computational Linguistics, 1988, pp. 365–371.

[LR01] A. Lecomte and C. Retoré. "Extending Lambek Grammars: a Logical Account of Minimalist Grammars". In: *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. ACL '01. Toulouse, France: Association for Computational Linguistics, 2001, pp. 362–369. DOI: 10.3115/1073012.1073059.

[Lee67] John A. N. Lee. *The Anatomy of a Compiler*. Van Nostrand, 1967.

[LS68] P. M. Lewis II and R. E. Stearns. "Syntax-Directed Transduction". In: *Journal of the ACM* 15.3 (July 1968), pp. 465–488. ISSN: 0004-5411. DOI: 10.1145/321466.321477.

[Llo08] R. Lloyd. "Era of Scientific Secrecy Near End". In: *Live Science* (2008). URL: http://www.livescience.com/2825-era-scientific-secrecy.html.

[Łuk77] L. Łukaszewicz. "Functional Grammars". In: *Information Processing Letters* 6.5 (1977), pp. 146–150. DOI: 10.1016/0020-0190(77)90011-4.

[Łuk82] L. Łukaszewicz. "Universal Grammars". In: *Information Processing Letters* 15.2 (1982), pp. 76–80. ISSN: 0020-0190. DOI: 10.1016/0020-0190(82)90111-9.

[Man06] P. B. Mann. "A Translational BNF Grammar Notation". In: *ACM SIGPLAN Notices* 41.4 (2006), pp. 16–23.

[Mas62] A. H. Maslow. *Toward a Psychology of Being*. Princeton, N.J.: D. Van Nostrand Co., 1962.

[MHW70] W. M. McKeeman, J. J. Horning, and D. B. Wortman. *A Compiler Generator*. Englewood Cliffs, N.J.: Prentice-Hall, 1970.

[Mei90] H. Meijer. "The Project of Extended Affix Grammars at Nijmegen". In: *Attribute Grammars and Their Applications*. Ed. by P. Deransart and M. Jourdan. Vol. 461. Lecture Notes in Computer Science. Springer, 1990. DOI: 10.1007/3-540-53101-7_10.

[MV11] B. Meyers and H. Vangheluwe. "A Framework for Evolution of Modelling Languages". In: *Science of Computer Programming* 76.12 (2011). Special Issue on Software Evolution, Adaptability and Variability, pp. 1223–1246. ISSN: 0167-6423. DOI: 10.1016/j.scico.2011.01.002.

[MS67] W. F. Miller and A. C. Shaw. *A Picture Calculus*. PUB-0358. SLAC, 1967. URL: http://www.slac.stanford.edu/pubs/slacpubs/0000/slac-pub-0358.html.

[Mon70] R. Montague. "Universal Grammar". In: *Theoria* 36.3 (1970), pp. 373–398. ISSN: 1755-2567. DOI: 10.1111/j.1755-2567.1970.tb00434.x.

[Moo01] L. Moonen. "Generating Robust Parsers using Island Grammars". In: *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01)*. IEEE Computer Society Press, Oct. 2001, pp. 13–22. DOI: 10.1109/WCRE.2001.957806.

[MN95] G. C. Murphy and D. Notkin. "Lightweight Source Model Extraction". In: *Proceedings of the Third ACM SIGSOFT symposium on Foundations of Software Engineering*. SIGSOFT '95. Washington, D.C., United States: ACM, 1995, pp. 116–127. ISBN: 0-89791-716-2.

[Nel08] S. R. Nelson. *Steel Drivin' Man. John Henry, The Untold Story of an American Legend*. Oxford University Press, 2008. ISBN: 978-0195341195.

[Nie+07] O. Nierstrasz, M. Kobel, T. Girba, M. Lanza, and H. Bunke. "Example-Driven Reconstruction of Software Models". In: *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*. IEEE, 2007, pp. 275–286. DOI: 10.1109/CSMR.2007.23.

[NNEH09] E. Nilsson-Nyman, T. Ekman, and G. Hedin. "Practical Scope Recovery Using Bridge Parsing". In: *Post-proceedings of the Second International Conference on Software Language Engineering*. Ed. by Dragan Gašević, Ralf Lämmel, and Eric Wyk. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 95–113. DOI: 10.1007/978-3-642-00434-6_7.

[Niv+91] M. Nivat, A. Saoudi, K. G. Subramanian, R. Siromoney, and V. R. Dare. "Puzzle Grammars and Context-free Array Grammars". In: *International Journal of Pattern Recognition and Artificial Intelligence* 5 (1991), pp. 663–676. DOI: 10.1142/S0218001491000399.

[Okh01] A. Okhotin. "Conjunctive Grammars". In: *Journal of Automata, Languages and Combinatorics* 6.4 (2001), pp. 519–535. URL: http://users.utu.fi/aleokh/papers/conjunctive.pdf.

[Okh04] A. Okhotin. "Boolean Grammars". In: *Information and Computation* 194.1 (2004), pp. 19–48. ISSN: 0890-5401. DOI: 10.1016/j.ic.2004.03.006. URL: http://users.utu.fi/aleokh/papers/boolean_grammars_ic.pdf.

[O'N08] E. J. O'Neil. "Object/Relational Mapping 2008: Hibernate and the Entity Data Model (EDM)". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 1351–1356. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376773.

[Oph97] J. L. Ophel. *Breadth-First Parsing*. Department of Computing and Electrical Engineering Research Memo 97/12. Dec. 1997.

[Ove05] P. Overell. *RFC 4234. Augmented BNF for Syntax Specifications: ABNF*. Oct. 2005. URL: http://tools.ietf.org/html/rfc4234.

[PW86] F Pereira and D Warren. "Definite Clause Grammars for Language Analysis". In: *Readings in Natural Language Processing*. Ed. by Barbara J. Grosz, Karen Sparck-Jones, and Bonnie Lynn Webber. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, pp. 101–124. ISBN: 0-934613-11-7. URL: http://cgi.di.uoa.gr/~takis/pereira-warren.pdf.

[Pol84] C. Pollard. "Generalized Phrase Structure Grammars, Head Grammars and Natural Language". PhD thesis. Stanford University, 1984.

[PM04] J. F. Power and B. A. Malloy. "A Metrics Suite for Grammar-based Software". In: *Journal of Software Maintenance and Evolution: Research and Practice* 16 (6 Nov. 2004), pp. 405–426. ISSN: 1532-060X. DOI: 10.1002/smr.293.

[Pra71] T. W. Pratt. "Pair Grammars, Graph Languages and String-to-Graph Translations". In: *Journal of Computer and System Sciences* 5.6 (Dec. 1971), pp. 560–595. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(71)80016-8.

[RVSW95] O. Rambow, K. Vijay-Shanker, and D. Weir. "D-Tree Grammars". In: *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. ACL '95. Cambridge, Massachusetts, USA: Association for Computational Linguistics, 1995, pp. 151–158. DOI: 10.3115/981658.981679.

[RWVS01] O. Rambow, D. Weir, and K. Vijay-Shanker. "D-Tree Substitution Grammars". In: *Journal of Computational Linguistics* 27.1 (Mar. 2001), pp. 89–121. ISSN: 0891-2017. URL: http://acl.ldc.upenn.edu/J/J01/J01-1004.pdf.

[RP05] S. C. Reghizzi and M. Pradella. "Tile Rewriting Grammars and Picture Languages". In: *Journal of Theoretical Computer Science* 340.2 (June 2005), pp. 257–272. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.03.041.

[Ric85] Helmut Richter. "Noncorrecting Syntax Error Recovery". In: *ACM Transactions on Programming Languages and Systems* 7.3 (July 1985), pp. 478–489. ISSN: 0164-0925.

[Ruc96] M. Ruckert. "Conservative Pretty Printing". In: *SIGPLAN Notices* 23.2 (1996), pp. 39–44. DOI: 10.1145/251621.251630.

[San08] K. Sanderson. "Data on Display". In: *Nature* 455.7211 (2008), p. 273. ISSN: 0028-0836. DOI: 10.1038/455273a.

[SAJ88] Y. Schabes, A. Abeille, and A. K. Joshi. "Parsing Strategies With 'Lexicalized' Grammars: Application to Tree Adjoining Grammars". In: *Proceedings of the 12th Conference on Computational Linguistics, Volume 2*. COLING '88. Budapest, Hungary: Association for Computational Linguistics, 1988, pp. 578–583. ISBN: 963 8431 56 3. DOI: 10.3115/991719.991757.

[Sch95] A. Schürr. "Specification of Graph Translators With Triple Graph Grammars". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by E. W. Mayr, G. Schmidt, and G. Tinhofer. Vol. 903. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, pp. 151–163. ISBN: 978-3-540-59071-2. DOI: 10.1007/3-540-59071-4_45.

[SJ10] E. Scott and A. Johnstone. "GLL Parsing". In: *Electronic Notes in Theoretical Computer Science* 253.7 (2010). Proceedings of the Ninth Workshop on Language Descriptions Tools and Applications (LDTA 2009), pp. 177–189. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2010.08.041.

[SV99] Alex Sellink and C. Verhoef. "Generation of Software Renovation Factories from Compilers". In: *Proceedings of the 15th IEEE International Conference on Software Maintenance (ICSM'99)*. 1999, pp. 245–255.

[SV00] M. P. A. Sellink and C. Verhoef. "Development, Assessment, and Reengineering of Language Descriptions". In: *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering (CSMR 2000)*. Ed. by J. Ebert and C. Verhoef. IEEE Computer Society, Mar. 2000, pp. 151–160. URL: http://staff.science.uva.nl/~x/cale/cale.pdf.

[Sik97] K. Sikkel. *Parsing Schemata — a Framework for Specification and Analysis of Parsing Algorithms*. Springer, 1997, pp. I–XVI, 1–365. ISBN: 978-3-540-61650-4.

[SS12] E. Silkensen and J. G. Siek. "Well-typed Islands Parse Faster". In: *ACM Computing Research Repository (CoRR)* abs/1201.0024 (2012). URL: http://arxiv.org/abs/1201.0024.

[Sim+11] J. Simon, A. Birukou, F. Casati, R. Casati, and M. Marchese. *Liquid Publications Green Paper*. LiquidPub. 2011. URL: https://dev.liquidpub.org/svn/liquidpub/papers/deliverables/LPGreenPaper.pdf.

[Sin08] S. Singh. "India Takes an Open Source Approach to Drug Discovery". In: *Cell* 133.2 (2008), pp. 201–203. ISSN: 0092-8674. DOI: 10.1016/j.cell.2008.04.003.

[SC12] A. Stevenson and J. R. Cordy. "Grammatical Inference in Software Engineering: an Overview of the State of the Art". In: *Pre-proceedings of the Fifth International Conference on Software Language Engineering (SLE 2012)*. Ed. by K. Czarnecki and G. Hedin. Dresden, Germany: Fakultät Informatik, Technische Universität Dresden, July 2012, pp. 206–225. URL: http://research.cs.queensu.ca/~cordy/Papers/SC_SLE12_GramInf.pdf.

[SCL12] T. van der Storm, W. R. Cook, and A. Loh. "Object Grammars: Compositional & Bidirectional Mapping Between Text and Graphs". In: *Pre-proceedings of the Fifth International Conference on Software Language Engineering (SLE 2012)*. Ed. by K. Czarnecki and G. Hedin. Dresden, Germany: Fakultät Informatik, Technische Universität Dresden, July 2012, pp. 4–23. URL: http://www.cs.utexas.edu/~wcook/Drafts/2012/ensogrammars.pdf.

[SSK72] G. Stromoney, R. Siromoney, and K. Krithivasan. "Abstract Families of Matrices and Picture Languages". In: *Computer Graphics and Image Processing* 1.3 (1972), pp. 284–307. ISSN: 0146-664X. DOI: 10.1016/S0146-664X(72)80019-4.

[SSC06] K. G. Subramanian, R. Saravanan, and P. H. Chandra. "Cooperating Basic Puzzle Grammar Systems". In: *Combinatorial Image Analysis*. Ed. by R. Reulke, U. Eckardt, B. Flach, U. Knauer, and K. Polthier. Vol. 4040. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 354–360. ISBN: 978-3-540-35153-5. DOI: 10.1007/11774938_28.

[SCD03] N. Synytskyy, J.R. Cordy, and T.R. Dean. "Robust Multilingual Parsing using Island Grammars". In: *Proceedings CASCON'03, 13th IBM Centres for Advanced Studies on Collaborative Research*. Toronto, Ontario, Canada: IBM Press, 2003, pp. 149–161.

[Vig+11] A. Vignaga, F. Jouault, M. Bastarrica, and H. Brunelière. "Typing Artifacts in Megamodeling". In: *Software and Systems Modeling* (2011), pp. 1–15. ISSN: 1619-1366. DOI: 10.1007/s10270-011-0191-2.

[Vis97] E. Visser. *Scannerless Generalized-LR Parsing*. Tech. rep. P9707. University of Amsterdam, July 1997.

[WM83] D. A. Watt and O. L. Madsen. "Extended Attribute Grammars". In: *The Computer Journal* 26.2 (1983). URL: http://comjnl.oxfordjournals.org/content/26/2/142.full.pdf.

[Wij65] A. van Wijngaarden. *Orthogonal Design and Description of a Formal Language*. MR 76. SMC, Oct. 1965. URL: http://www.fh-jena.de/~kleine/history/languages/VanWijngaarden-MR76.pdf.

[Wij74] A. van Wijngaarden. "The Generative Power of Two-Level Grammars". In: *Automata, Languages and Programming*. Ed. by J. Loeckx. Vol. 14. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1974, pp. 9–16. ISBN: 978-3-540-06841-9. DOI: 10.1007/3-540-06841-4_48.

[Wil97] David S. Wile. "Abstract Syntax from Concrete Syntax". In: *Proceedings of the 19th International Conference on Software Engineering*. ICSE '97. Boston, Massachusetts, United States: ACM, 1997, pp. 472–480. ISBN: 0-89791-914-9.

[Wir77] N. Wirth. "What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions?" In: *Communications of the ACM* 20.11 (1977), pp. 822–823. DOI: 10.1145/359863.359883.

[Wu97] D. Wu. "Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora". In: *Journal of Computational Linguistics* 23.3 (Sept. 1997), pp. 377–403. ISSN: 0891-2017. URL: http://acl.ldc.upenn.edu/J/J97/J97-3002.pdf.

[Zay05] V. Zaytsev. "Correct C# Grammar too Sharp for ISO". In: *Participants Workshop, Part II of the Pre-proceedings of the International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2005)*. Extended abstract. Braga, Portugal: Technical Report, TR-CCTC/DI-36, Universidade do Minho, July 2005, pp. 154–155. URL: http://grammarware.net/writes/\#Too-Sharp2005.

[Zay10] V. Zaytsev. "Recovery, Convergence and Documentation of Languages". PhD thesis. Amsterdam, The Netherlands: Vrije Universiteit, 2010. URL: http://grammarware.net/writes/\#Zaytsev-Thesis2010.

[Zay11a] V. Zaytsev. "Language Convergence Infrastructure". In: *Post-proceedings of the Third International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2009)*. Ed. by J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva. Vol. 6491. LNCS. Berlin, Heidelberg: Springer-Verlag, Jan. 2011, pp. 481–497. DOI: 10.1007/978-3-642-18023-1_16. URL: http://grammarware.net/writes/\#LCI2011.

[Zay11b] V. Zaytsev. "MediaWiki Grammar Recovery". In: *Computing Research Repository (CoRR)* abs/1107.4661 (July 2011), pp. 1–47. URL: http://arxiv.org/abs/1107.4661.

[Zay12d] V. Zaytsev. "BNF WAS HERE: What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions". In: *Programming Languages Track, Volume II of the Proceedings of the 27th ACM Symposium on Applied Computing (SAC 2012)*. Ed. by S. Ossowski and P. Lecca. Riva del Garda, Italy: ACM, Mar. 2012, pp. 1910–1915. ISBN: 978-1-4503-0857-1. DOI: 10.1145/2245276.2232090. URL: http://dx.doi.org/10.6084/m9.figshare.91423.

[Zay12k] V. Zaytsev. "Guided Grammar Convergence. Full Case Study Report. Generated by converge::Guided". In: *ACM Computing Research Repository (CoRR)* abs/1207.6541 (July 2012), pp. 1–44. URL: http://arxiv.org/abs/1207.6541.

[Zay12p] V. Zaytsev. "Language Evolution, Metasyntactically". In: *Pre-proceedings of the First International Workshop on Bidirectional Transformation (BX 2012)*. Ed. by F. Hermann and J. Voigtländer. Institute of Cybernetics at Tallinn University of Technology, Mar. 2012. DOI: 10.6084/m9.figshare.91427.

[Zay12r] V. Zaytsev. "Language Evolution, Metasyntactically". In: *Electronic Communications of the European Association of Software Science and Technology (EC-EASST)* 49 (2012), pp. 1–17. ISSN: 1863-2122. URL: http://journal.ub.tu-berlin.de/eceasst/article/view/708.

[Zay12u] V. Zaytsev. "Negotiated Grammar Transformation". In: *Extreme Modeling Workshop (XM 2012)*. Ed. by J. De Lara, D. Di Ruscio, and A. Pierantonio. Dipartimento di Informatica, Università degli Studi dell'Aquila, Oct. 2012. URL: http://www.di.univaq.it/diruscio/sites/XM2012/xm2012_submission_11.pdf.

[Zay12v] V. Zaytsev. "Negotiated Grammar Transformation". In: *Post-proceedings of the Extreme Modeling Workshop (XM 2012)*. Ed. by J. De Lara, D. Di Ruscio, and A. Pierantonio. ACM Digital Library, Nov. 2012. URL: http://grammarware.net/writes/\#Negotiated2012.

[Zay12x] V. Zaytsev. "Notation-Parametric Grammar Recovery". In: *Post-proceedings of the 12th International Workshop on Language Descriptions, Tools, and Applications (LDTA 2012)*. Ed. by A. Sloane and S. Andova. ACM Digital Library, June 2012. URL: http://grammarware.net/writes/\#NPGR2012.

[Zay12y] V. Zaytsev. "Notation-Parametric Grammar Recovery". In: *Pre-proceedings of the 12th International Workshop on Language Descriptions, Tools, and Applications (LDTA 2012)*. Ed. by A. Sloane and S. Andova. Institute of Cybernetics at Tallinn University of Technology, Mar. 2012, pp. 105–118. DOI: 10.6084/m9.figshare.91425.

[Zay12ab] V. Zaytsev. "Renarrating Linguistic Architecture: A Case Study". In: *Sixth International Workshop on Multi-Paradigm Modeling (MPM 2012)*. Ed. by C. Hardebolle, E. Syriani, J. Sprinkle, and T. Mészáros. Automatizálási és Alkalmazott Informatikai Tanszék, Budapesti Műszaki és Gazdaságtudományi Egyetem, Oct. 2012. URL: http://avalon.aut.bme.hu/mpm12/papers/paper\%2015.pdf.

[Zay12ac] V. Zaytsev. "Renarrating Linguistic Architecture: A Case Study". In: *Post-proceedings of the Sixth International Workshop on Multi-Paradigm Modeling (MPM 2012)*. Ed. by C. Hardebolle, E. Syriani, J. Sprinkle, and T. Mészáros. ACM Digital Library, Nov. 2012. ISBN: 978-1-4503-1805-1. URL: http://grammarware.net/writes/\#Renarration2012.

[ZL11] V. Zaytsev and R. Lämmel. "A Unified Format for Language Documents". In: *Post-proceedings of the Third International Conference on Software Language Engineering (SLE 2010)*. Ed. by B. A. Malloy, S. Staab, and M. G. J. van den Brand. Vol. 6563. LNCS. Berlin, Heidelberg: Springer-Verlag, Jan. 2011, pp. 206–225. DOI: 10.1007/978-3-642-19440-5_13. URL: http://grammarware.net/writes/\#TestMatch2012.

[Zay+08] V. Zaytsev, R. Lämmel, T. van der Storm, L. Renggli, and G. Wachsmuth. *Software Language Processing Suite*[32]. http://grammarware.github.com. Contains, among other works: *XBGF Manual: BGF Transformation Operator Suite v.1.0* (Zaytsev, 2010), http://grammarware.github.com/xbgf; *Grammar Zoo* (Zaytsev, 2009–2012), http://grammarware.github.com/zoo; *Grammar Tank* (Zaytsev, 2011–2012), http://grammarware.github.com/tank. 2008–2012.

[32]The authors are given according to the list of contributors at http://github.com/grammarware/slps/graphs/contributors.

Unpublished work

[Zay12a] V. Zaytsev. *Automated Reuse-driven Grammar Restructuring*. Veni application 016.133.021. Classified as "very good" and rejected. NWO, 2012, pp. 1–17.

[Zay12h] V. Zaytsev. "Guided Grammar Convergence". Submitted to the Eighth European Conference on Modelling Foundations and Applications (ECMFA 2012). Rejected. Feb. 2012.

[Zay12i] V. Zaytsev. "Guided Grammar Convergence". Submitted to the 28th IEEE International Conference on Software Maintenance (ICSM 2012). Rejected. Apr. 2012.

[Zay12j] V. Zaytsev. "Guided Grammar Convergence". Submitted to the 40th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2013). Rejected. June 2012.

[Zay12m] V. Zaytsev. "Incremental Replications of Grammarware Engineering Experiments". Submitted to the Special Issue on Experimental Replications of the Journal of Empirical Software Engineering (EMSE). Rejected. June 2012.

[Zay12n] V. Zaytsev. "Islands in the Cloud". Submitted to the 12th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2012). Rejected. May 2012.

[Zay12o] V. Zaytsev. "Islands in the Cloud". Submitted to the Nordic Symposium on Cloud Computing and Internet Technologies (NordiCloud 2012). Rejected. June 2012.

[Zay12t] V. Zaytsev. "Metasyntactic Footprints and Signatures". Submitted to the 24th Nordic Workshop on Programming Theory (NWPT). Rejected. Extended abstract. Nov. 2012.

[Zay12ae] V. Zaytsev. "Reusing a Community-Created Grammar". Submitted to the Special Issue of the Journal of Universal Computer Science on Software Components, Architectures and Reuse (JUCS). Rejected. Apr. 2012.

[Zay12ah] V. Zaytsev. "Trends in Grammar Transformation". Submitted to the 13th International Symposium on Trends in Functional Programming (TFP 2012). Rejected. Mar. 2012.

[Zay12ai] V. Zaytsev. "Unparsing Techniques". Submitted to the New Ideas Track of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2012 NIER). Rejected. June 2012.

[Zay12aj] V. Zaytsev. "Visualization of Grammarware Engineering Processes". Submitted to the New Ideas Track of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2012 NIER). Rejected. June 2012.

[Zay13] V. Zaytsev. *Grammar Zoo: A Repository of Experimental Grammarware*. Submitted to the Fifth Special issue on Experimental Software and Toolkits of Science of Computer Programming (SCP EST5). Under review. 2013. URL: http://grammarware.net/writes/\#Zoo2013.

Presentations

[Bag12] A. H. Bagge. "Design of the Magnolia IDE and Compiler". Colloquium presentation at the *Programming Environment Meeting*, 20 December. 2012.

[BBB12] C. Bast, W. Bast, and T. Brus. "Towards a programming language that really abstracts away from time". Colloquium presentation at the *Programming Environment Meeting*, 5 October. 2012.

[Bos12a] J. van den Bos. "A Little Language: Little Maintenance?" Colloquium presentation at the *Software Engineering Meeting*, 23 November. 2012.

[Bos12b] J. van den Bos. "Domain-Specific Optimization in Digital Forensics". Colloquium presentation at the *Software Engineering Meeting*, 16 March. 2012.

[Bru12] M. Bruntink. "Back in Business Science". Colloquium presentation at the *Software Engineering Meeting*, 9 November. 2012.

[Coo12] W. Cook. "Ensō". Keynote presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012. URL: http://enso-lang.org/blog/2012/07/18/presentations-on-enso/.

[Cza12] K. Czarnecki. "Re-Envisioning Lightweight Modeling". Colloquium presentation at the *Programming Environment Meeting*, 15 June. 2012.

[Dem12] S. Demeyer. "Agile Quality Assurance". Colloquium presentation at the *Programming Environment Meeting*, 25 January. 2012. URL: http://event.cwi.nl/pem/2012/2012-01-25-Demeyer.pdf.

[Din12] T. B. Dinesh. "Xpaths + Semantic Tweets = Social Semantic Web?" Colloquium presentation at the *Programming Environment Meeting*, 13 July. 2012.

[Dor12] J. Dormans. "Engineering Tools for Game Design". Colloquium presentation at the *Programming Environment Meeting*, 25 October. 2012.

[Ger12] D. M. German. "The Software Engineer Meets the Lawyer: The Legal Challenges of Software Reuse". Colloquium presentation at the *Programming Environment Meeting*, 11 April. 2012.

[God12] M. Godfrey. "Requirements Engineering: Parts I–IV". Colloquium presentation at the *Programming Environment Meeting*, 13, 25, 27 April, 29 June. 2012.

[Hav12] M. Haveraaen. "The Inliner: a Core Engine for Refactoring and Module Systems". Colloquium presentation at the *Programming Environment Meeting*, 18 June. 2012.

[HB12] M. Haveraaen and A. H. Bagge. "Magnolia: A Mouldable Language". Colloquium presentation at the *Programming Environment Meeting*, 18 June. 2012.

[Hil12a] M. Hills. "Meta-Language Support for Type-Safe Access to External Resources". Colloquium presentation at the *Software Engineering Meeting*, 7 September. 2012.

[Hil12b] M. Hills. "Modularity of Language Definitions". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Hil12c] M. Hills. "Type Checking Rascal in Rascal". Colloquium presentation at the *Programming Environment Meeting*, 11 May. 2012. URL: http://event.cwi.nl/pem/2012/2012-05-11-Hills.pdf.

[Izm12] A. Izmaylova. "TyMoRe: Towards a Modular, Language-Parametric Refactoring Framework". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Joo12] S. Joosten. "Large-Scale Information Systems Using a Simple Algebraic Input Language". Colloquium presentation at the *Programming Environment Meeting*, 2 November. 2012.

[Kat12] L. Kats. "Software Language Engineering in the Cloud". Colloquium presentation at the *Programming Environment Meeting*, 24 August. 2012.

[Lag12] P. Lago. "Supporting Global Software Engineering Communities as Agile Service Networks". Colloquium presentation at the *Programming Environment Meeting*, 16 November. 2012.

[Lan12a] D. Landman. "`git` session". Colloquium presentation at the *Software Engineering Meeting*, 3 August. 2012.

[LB12] D. Landman and J. van den Bos. "Lego Turing Machine". Colloquium presentation at the *Software Engineering Meeting*, 1 June. 2012.

[Lan12b] W. Langdon. "Genetic Programming to Improve Software". Colloquium presentation at the *Programming Environment Meeting*, 11 October. 2012. URL: http://event.cwi.nl/pem/2012/2012-10-11-Langdon.pdf.

[Loh12a] A. Loh. "An Algebra for Interpreter Combinators". Colloquium presentation at the *Programming Environment Meeting*, 3 September. 2012.

[Loh12b] A. Loh. "Ensō: a Self-Describing Language Workbench". Colloquium presentation at the *Programming Environment Meeting*, 6 April. 2012.

[Loh12c] A. Loh. "Language Composition Defined". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Pac12] S. Pachidi. "Software Operation Data Mining". Colloquium presentation at the *Programming Environment Meeting*, 16 February. 2012.

[Plo12a] A. van der Ploeg. "Declarative Generalized 2D Graphics". Colloquium presentation at the *Software Engineering Meeting*, 21 September. 2012.

[Plo12b] A. van der Ploeg. "On the Timely Drawing of Tidy Trees". Colloquium presentation at the *Software Engineering Meeting*, 20 January. 2012.

[Pos12] R. Poss. "Don't Forget the Hardware!" Colloquium presentation at the *Programming Environment Meeting*, 7 December. 2012. URL: http://event.cwi.nl/pem/2012/2012-12-07-Poss.pdf.

[Roz12a] R. van Rozen. "Language-Centric Approaches for Game Development: Part I". Colloquium presentation at the *Software Engineering Meeting*, 9 March. 2012.

[Roz12b] R. van Rozen. "Static Analysis of Lua Using Interface Models". Colloquium presentation at the *Software Engineering Meeting*, 14 September. 2012.

[Sch12] A. Scherpenisse. "Erlang: Software for a Concurrent World". Colloquium presentation at the *Programming Environment Meeting*, 14 December. 2012.

[Ser12] A. Serebrenik. "Measuring and Mining Evolution of Software Projects". Colloquium presentation at the *Programming Environment Meeting*, 10 February. 2012. URL: http://event.cwi.nl/pem/2012/2012-02-10-Serebrenik.pdf.

[Sim12] S. Simon. "Choosing Products in Social Networks". Colloquium presentation at the *Programming Environment Meeting*, 6 July. 2012.

[Slo12] A. Sloane. "The Kiama Language Processing Library". Colloquium presentation at the *Programming Environment Meeting*, 2 March. 2012. URL: http://event.cwi.nl/pem/2012/2012-03-02-Sloane.pdf.

[Ste12a] M. Steindorfer. "Extraction, Storage and Retrieval of Partial Code Snippets". Colloquium presentation at the *Programming Environment Meeting*, 1 March. 2012.

[Ste12b] M. Steindorfer. "Requirements for Language Composition". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Ste12c] M. Steindorfer. "The Ninth LASER Summer School on Software Engineering". Colloquium presentation at the *Software Engineering Meeting*, 28 September. 2012.

[Ste12d] L. Stevens. "Automatically Evaluating the Consistency and Conciseness of Class Names". Colloquium presentation at the *Software Engineering Meeting*, 8 June. 2012.

[Sto12a] T. van der Storm. "Language Design Smells". Colloquium presentation at the *Software Engineering Meeting*, 24 February. 2012.

[Sto12b] T. van der Storm. "Towards IDE Composition". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Sto12c] T. van der Storm. "Understanding Data Abstraction". Colloquium presentation at the *Software Engineering Meeting*, 23 March. 2012.

[Tes12] T. Tesfay. "REST ASSURED". Colloquium presentation at the *Programming Environment Meeting*, 5 October. 2012.

[Tip12] F. Tip. "Automated Repair of HTML Generation Errors in PHP Applications Using String Constraint Solving". Colloquium presentation at the *Programming Environment Meeting*, 4 December. 2012.

[Var12] A. Varanovich. "Modeling the Linguistic Architecture of Software Products". Colloquium presentation at the *Programming Environment Meeting*, 11 October. 2012.

[Vin12a] J. Vinju. "Eyeballing the Cyclomatic Complexity Metric". Colloquium presentation at the *Software Engineering Meeting*, 20 July. 2012.

[Vin12b] J. Vinju. "Lies, Damn Lies and Modularity". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012.

[Wal12] J. Wales. "Wikipedia, Education and the Internet". Keynote presentation at the *Sixth Wikimedia Conference Netherlands (WCN 2012)*, 9 November. 2012.

[Zay12b] V. Zaytsev. "Bidirectional Grammar Transformations". Seminar presentation at the *Fifth Seminar Series on Advanced Techniques and Tools for Software Evolution (SATToSE)*, 20 August. 2012. URL: http://grammarware.net/talks/\#BGX2012.

[Zay12c] V. Zaytsev. "Bidirectional Transformations and Grammarware". Colloquium presentation at the *Software Engineering Meeting*, 3 February. 2012. URL: http://grammarware.net/talks/\#SEM2012BX.

[Zay12e] V. Zaytsev. "Experimental Replications". Colloquium discussion at the *Programming Environment Meeting*, 25 May. 2012. URL: http://grammarware.net/talks/\#Replications2012.

[Zay12f] V. Zaytsev. "Grammar Composition and Extension". Symposium presentation at the *Symposium on Language Composability and Modularity*, 10 August. 2012. URL: http://grammarware.net/talks/\#Decomposition2012.

[Zay12g] V. Zaytsev. "Grammar Convergence". Presentation of recent research results at the *Software Languages Team*, 25 July. 2012. URL: http://grammarware.net/talks/\#Guided-Convergence2012.

[Zay12l] V. Zaytsev. "History and Future of the PEM Colloquium". Colloquium discussion at the *Programming Environment Meeting*, 20 January. 2012. URL: http://grammarware.net/talks/\#PEM2012.

[Zay12q] V. Zaytsev. "Language Evolution, Metasyntactically". Paper presentation at the *First International Workshop on Bidirectional Transformations*, 25 March. 2012. URL: http://grammarware.net/talks/\#BX2012.

[Zay12s] V. Zaytsev. "Megamodelling Language Design: User Experiences and Ad Hoc Megamodelling". Local talk about ongoing research results in joint work at the *Software Languages Team*, 18 July. 2012. URL: http://grammarware.net/talks/\#MegaL2012.

[Zay12w] V. Zaytsev. "Negotiated Grammar Transformation". Paper presentation at the *Extreme Modeling Workshop*, 1 October. 2012. URL: http://grammarware.net/talks/\#Negotiated-XM2012.

[Zay12z] V. Zaytsev. "Open Notebook Computer Science". Presentation at the *Software Freedom Day "Open Education"*, 19 September. 2012. URL: http://grammarware.net/talks/\#Open2012.

[Zay12aa] V. Zaytsev. "Renarrating Linguistic Architecture". Poster presentation at the *Workshop on Multi-Paradigm Modeling*, 1–5 October. 2012. URL: http://grammarware.net/talks/\#Renarration-MPM2012.

[Zay12ad] V. Zaytsev. "Renarration of Megamodels". Presentation of upcoming research results at the *Software Languages Team*, 25 July. 2012. URL: http://grammarware.net/talks/\#Renarration-SLT2012.

[Zay12af] V. Zaytsev. "Subatomic Scientific Knowledge Objects". Remote presentation at the *Software Languages, Software Linguistics and Software Language Engineering Body of Knowledge*, 25 September. 2012. URL: http://grammarware.net/talks/\#Subatomic2012.

[Zay12ag] V. Zaytsev. "Tolerance in Grammarware". Colloquium presentation at the *Programming Environment Meeting*, 4 May. 2012. URL: http://grammarware.net/talks/\#Tolerance2012.