SdfMetz: Extraction of Metrics and Graphs From Syntax Definitions[⋆] — Tool Demonstration —

Tiago L. Alves[1] and Joost Visser[2]

[1] University of Minho, Portugal, and
Software Improvement Group, The Netherlands
`t.alves@sig.nl`
[2] Software Improvement Group, The Netherlands
`j.visser@sig.nl`

Abstract. We developed SdfMetz, a tool for the extraction of metrics and graphs from syntax descriptions. SdfMetz supports various input languages, such as SDF and the syntax formalisms of DMS, ANTLR, and Bison. Among the extracted metrics are size and complexity metrics, feature metrics, and structure metrics. Some metrics are extracted directly from grammars, such as adaptations of the NPath and cyclomatic complexity metrics. Several structure metrics, such as tree impurity and recursiveness, are based on various kinds of grammar dependency graphs. The metrics and graphs can be emitted in several formats to allow their subsequent visualization and (statistical) analysis. We present the functionality of the tool, its implementation, and its use for grammar comparison and analysis.

1 Introduction

Grammars play a central role in language tool development. Their primary purpose is definition of surface syntax and parser generation, but they are likewise used to generate other language processing ingredients, such as AST traversal support and pretty-printers. Currently, renewed interest in domain-specific languages [1] (DSLs), e.g. in the context of model-driven engineering (MDE), again emphasises grammars as primary software artifacts.

Grammar engineering [2] aims to apply solid software engineering techniques to grammars. Such techniques include version control, static analysis, and testing. Through their adoption, the notoriously erratic and unpredictable process of developing and maintaining large grammars can become more efficient and effective, and can lead to results of higher-quality. In grammar engineering, quantification is an important instrument for understanding and controlling grammar evolution as well as for specifying and improving grammar quality attributes, just as for software artifacts and evolution in general.

[⋆] This is an extended version of a tool demo paper that was presented at LDTA 2006, but was not published in the proceedings.

Though grammars have been used in software engineering for decades, the systematic definition and application of grammar metrics is a more recent development. Power and Malloy [3] have defined a suite of metrics for attributes of grammars, such as *size*, *complexity*, and *structure*. Their definitions are given for grammars written in BNF, EBNF, or Yacc-style BNF dialects.

We have implemented a tool for grammar quantification and visualization, named SdfMetz. The suite of metrics calculated by SdfMetz is a significant extension of those defined by Power and Malloy, and apart from Yacc-like BNF dialects (Bison), the tool accepts ANTLR [4], SDF [5], and DMS [6] grammars. We also implemented *disambiguation* metrics, relevant for SDF only. The graphs constructed by SdfMetz for calculation of various metrics can also be exported and used for grammar visualization.

Elsewhere, we report on the use of SdfMetz for monitoring the development of an industrial strength grammar of the VDM-SL language [7].

2 Tool functionality

SdfMetz is a command line tool that accepts various grammar formalisms as input and emits either a metrics report or a graph. Currently the accepted input includes SDF, ANTLR, Bison, and DMS grammars. A total of thirty one metrics can be emitted:

– The twelve metrics defined in [3]: the counts of terminals, non-terminals, and production rules; cyclomatic complexity; average size of right-hand side; Halstead effort; tree impurity; number of grammatical levels, and normalised count of levels; count of non-singleton levels; count of non-terminals in the largest grammatical level; and maximum height.
– Twelve additional Hasltead metrics. Besides Halstead effort defined in [3], we additionally report: the underlying count of operators and operands, distinct and total; and derived metrics such as length, vocabulary, volume, difficulty, level, and time.
– An alternative tree impurity metric, applied to the successor graph of a grammar (instead of the transitive closure of that graph as defined in [3]).
– Three NPath metrics. We adapted the NPath metric [8] to grammars, which, like cyclometric complexity, measures possible paths, but gives more weight to nested choices. Additionally we implemented the average NPath per non-terminal and maximum NPath per non-terminal.
– Four ambiguity metrics, specific to SDF: counts of follow restrictions, associativity and preference attributes, reject and prioritized productions.

These metrics can be emitted either as nicely formatted textual reports for human consumption, or as comma-separated value files for further processing by spreadsheet or statistical tools. An example of this will be given in Section 4.

The calculation of several metrics requires the construction of a grammar successor graph, its transitive closure, or its corresponding strong connected component graph. SdfMetz can emit these graphs in the `dot` format [9]. An example of a strong connected component graph is given in Figure 1.

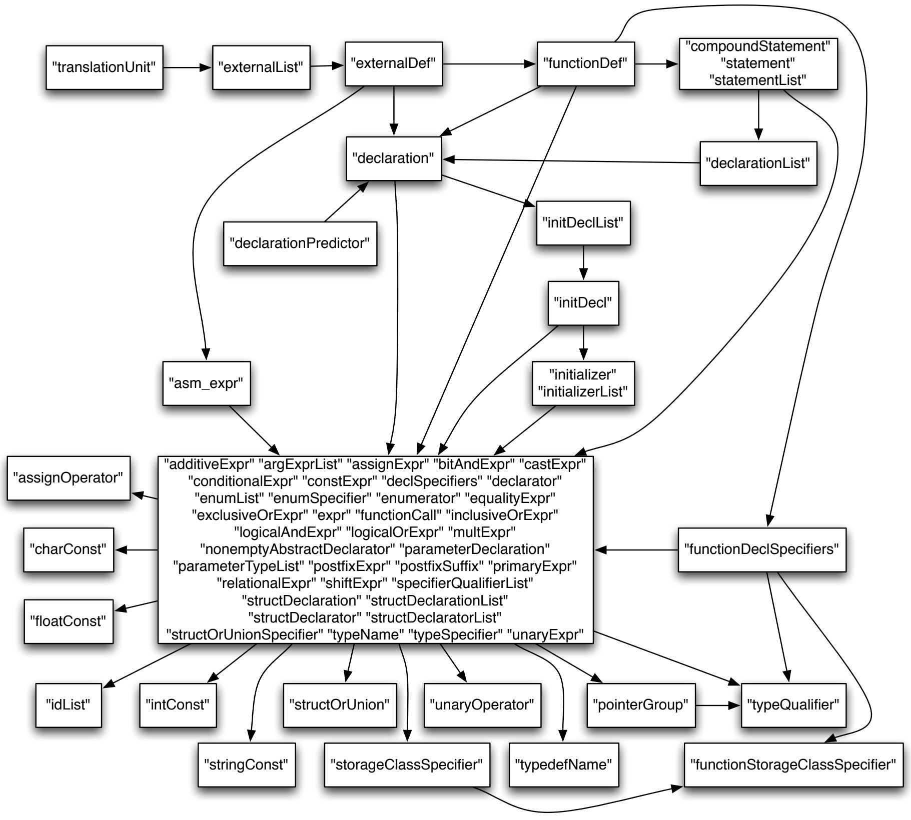
"translationUnit"
"externalList"
"externalDef"
"functionDef"
"compoundStatement" "statement" "statementList"
"declaration"
"declarationList"
"declarationPredictor"
"initDeclList"
"initDecl"
"initializer" "initializerList"
"asm_expr"
"additiveExpr" "argExprList" "assignExpr" "bitAndExpr" "castExpr" "conditionalExpr" "constExpr" "declSpecifiers" "declarator" "enumList" "enumSpecifier" "enumerator" "equalityExpr" "exclusiveOrExpr" "expr" "functionCall" "inclusiveOrExpr" "logicalAndExpr" "logicalOrExpr" "multExpr" "nonemptyAbstractDeclarator" "parameterDeclaration" "parameterTypeList" "postfixExpr" "postfixSuffix" "primaryExpr" "relationalExpr" "shiftExpr" "specifierQualifierList" "structDeclaration" "structDeclarationList" "structDeclarator" "structDeclaratorList" "structOrUnionSpecifier" "typeName" "typeSpecifier" "unaryExpr"
"assignOperator"
"charConst"
"floatConst"
"functionDeclSpecifiers"
"idList"
"intConst"
"structOrUnion"
"unaryOperator"
"pointerGroup"
"typeQualifier"
"stringConst"
"storageClassSpecifier"
"typedefName"
"functionStorageClassSpecifier"

Fig. 1. Excerpt of a strong connected component graph for an ANTLR grammar of the C language, generated with SdfMetz.

3 Tool implementation

SdfMetz was developed in Haskell and SDF, making essential use of the Strafunski bundle [10] for generating Haskell code from SDF grammars and for generic AST traversal. From the SDF grammar of each input language we generated AST, serialization, traversal, and pretty-printer components.

After parsing an input grammar, it is first translated to SDF, which is thus used as a universal syntax definition language. This allows all metrics to have a single implementation. Some metrics were defined directly over the AST. Other metrics require the construction of a successor graph, and subsequent calculation of its transitive closure or strong components. We used a graph library based on finite maps for the representation and manipulation of such graphs. Further librares were used, such as datatype libraries for sets, and bags, and for exporting to `dot` format. The tool was extensively unit tested using the HUnit framework [11].

4 Tool demonstration

The demonstration of the tool consists of the following elements.

Run We show how to invoke SdfMetz on various different input formalisms, and how to configure it to obtain different kinds of output.

Graphs By means of instructive examples, we explain which kinds of graphs are emitted, and how they can be visualized and interpreted.

Metrics Using grammars of well-known languages as examples, we explain the various metrics emitted by SdfMetz. We explain how these metrics can be visualized and interpreted.

Census We present the result of running SdfMetz on a large suite of syntax definitions, of various sizes and written in various grammar formalisms. We explain how simple statistical instruments can be used to interpret the huge amount of resulting measurement data.

As example we describe two scenarios where we applied SdfMetz: grammar comparison and statistical study of the tree impurity metric.

4.1 Grammar comparison

The main motivation to support different grammar formalisms in SdfMetz was to initiate the research of grammar quality. Our main focus is to identify an important set of metrics that can be used to assess grammar quality. Not only are we interested in investigating quality of grammars specified in the same formalism (e.g. quality of ANTLR grammars) but also quality of grammars in different formalisms.

As starting step for that purpose we show how SdfMetz can be used to compute and collect metrics for different formalisms. Additionally, we show an excerpt of the collected metrics and provide comments about our findings.

Grammar Collection The first step to analyze grammars is to collect grammars for different formalisms. We collected grammars from different sources. ANTLR grammars were download from the ANTLR website [12]. Bison grammars were obtained from GNU GCC [13] and from several internet pages. DMS grammars were kindly provided by Dr. Ira Baxter, from Semantic Designs (SD) [14]. SDF grammars were downloaded from GrammarBase web site (GB) [15] and provided by the authors. Additional grammars were kindly provided by Dr. Tobias Kuipers from the Software Improvement Group (SIG) [16].

Metric computation To compute and collect metrics we execute SdfMetz for a set of grammars in the same formalism. The execution of SdfMetz is done as following:

```
$ SdfMetz -f antlr -c -o AntlrMetrics.csv -i AntlrGrammars/*.g
$ SdfMetz -f bison -c -o BisonMetrics.csv -i BisonGrammars/*.y
$ SdfMetz -f dms -c -o DMSMetrics.csv -i DMSGrammars/*.def
$ SdfMetz -c -o SDFMetrics.csv -i SDFGrammars/*.bnf
```

Table 1. Sampled grammars and some of their properties.

Language	Origin	Type	TERM	VAR	PROD	MCC	NPATH	E	TIMPi	TIMP	LEV	CLEV	NSLEV	DEP	HEI
XPath	GCC	Bison	47	28	86	58	86	7.8	2.3	93.7	9	32.1	1	20	4
BibTeX	GB	SDF	20	6	16	21	32	8.9	20.0	100.0	6	100.0	0	1	6
MatLab	DK	Bison	44	34	92	58	92	13.0	4.0	62.3	15	44.1	2	16	6
Fortran 77	GB	SDF	41	16	41	32	58	18.9	4.8	42.9	15	93.8	1	2	7
C	P&M	BNF	86	65	-	149	-	51	-	64.1	22	33.8	3	38	13
C	ALR	ANTLR	122	67	67	197	285	59.4	1.4	94.9	27	40.9	3	37	13
Java 1.3	ALR	ANTLR	104	68	68	171	263	74.0	2.1	86.1	24	35.3	1	45	9
EcmaScript	SD	DMS	144	90	344	254	344	80.8	1.6	83.7	29	32.2	3	57	10
Ada	HF	Bison	93	238	458	220	458	87.4	0.8	63.3	156	65.5	4	42	24
Java	P&M	BNF	100	149	-	213	-	95	-	32.7	89	59.7	4	33	23
PHP 5	SD	DMS	159	112	418	306	418	96.5	1.6	76.1	55	49.1	2	37	15
Java 1.5	ALR	ANTLR	108	102	102	245	450	132.2	1.9	93.1	24	23.5	2	73	9
SDL	GB	SDF	89	91	174	170	273	132.5	1.1	39.8	76	83.5	2	13	14
Java	SD	DMS	99	144	460	316	460	137.9	1.3	93.9	41	28.5	2	87	17
C	GB	SDF	102	24	148	162	196	146.4	5.9	75.9	15	62.5	1	10	10
C++	P&M	BNF	116	141	-	368	-	173.0	-	85.8	21	14.9	1	121	4
EcmaScript	ALR	ANTLR	185	198	198	406	612	175.6	0.6	43.5	104	52.5	4	89	10
DB2	SIG	SDF	214	98	292	311	478	185.0	1.0	42.6	69	71.1	1	29	16
C#	ALR	ANTLR	133	219	219	408	660	202.3	0.7	55.6	159	72.9	4	30	27
PL/SQL	ALR	ANTLR	196	157	157	449	4683	215.5	1.5	45.6	119	76.3	6	15	21
C#	P&M	BNF	138	245	-	466	-	228	-	29.7	159	64.9	5	44	28
VDM-SL	TA	SDF	143	71	227	232	316	247.6	2.8	78.7	35	49.3	3	27	13
Java 1.5	GB	SDF	105	122	327	318	616	265.4	2.1	79.8	53	43.4	2	63	10
Ada	ALR	ANTLR	99	209	209	326	537	295.4	1.1	56.9	143	68.4	5	36	19
VB.net	JV	SDF	170	206	446	466	1554	390.0	1.0	42.9	154	74.8	6	25	26
Veriog 2001	SD	DMS	232	488	1248	760	1248	455.7	0.5	46.1	266	54.5	10	117	19
SDL	ALR	ANTLR	174	461	461	822	2043	708.9	0.6	35.4	357	77.4	6	43	28
PL/SQL	SIG	SDF	456	499	1094	888	1564	710.9	0.3	24.5	434	87.0	2	38	29
Cobol	GB	SDF	479	774	1330	1122	2194	909.4	0.2	22.2	627	81.2	7	70	26
C++	SD	DMS	173	436	2122	1686	2122	1300.6	0.7	96.3	42	9.6	2	393	10

GB	=	The online Grammar Base [15].	JV	=	Joost Visser.
SIG	=	Software Improvement Group [16].	TA	=	Tiago Alves.
SD	=	Semantic Designs [14].	DL	=	Danny Luk.
GCC	=	GNU C Compiler [13].	HF	=	Herman Fischer.
ALR	=	ANTLR web site [12].	P&M	=	Power and Malloy [3].

The `-f` argument indicates the grammar formalism to be analyzed. In case of an SDF grammar, this argument can be omitted. The `-c` argument indicates that the output is done in Comma Separated Values (CSV) format. The `-o` argument indicates the output file where the results are stored, and the `-i` argument indicates the files where grammars can be found. SdfMetz can handle a single file or multiple files of the same formalism, as show in the code execution example.

Metric analysis To analyze the metrics we imported the four generated files into Excel for statistical analysis. An excerpt of the analyzed grammars and computed metrics is presented in Table 1. Additionally, for comparison, we manually included the values for the grammars reported in [3]. The grammars are ordered by the Halstead effort metric.

Focusing on the McCabe's cyclomatic complexity (MCC) we can observe that it is below 500 for 25 of the 30 grammars. From the remaining ones, the two highest scoring grammars are the Cobol grammar from the GB (MCC=1122) and the C++ grammar from the SD(MCC=1686), which double and triple this value, respectively. A possible explanation for these high values is that the Cobol grammar contains an embedded SQL grammar and the C++ grammar includes macro parsing.

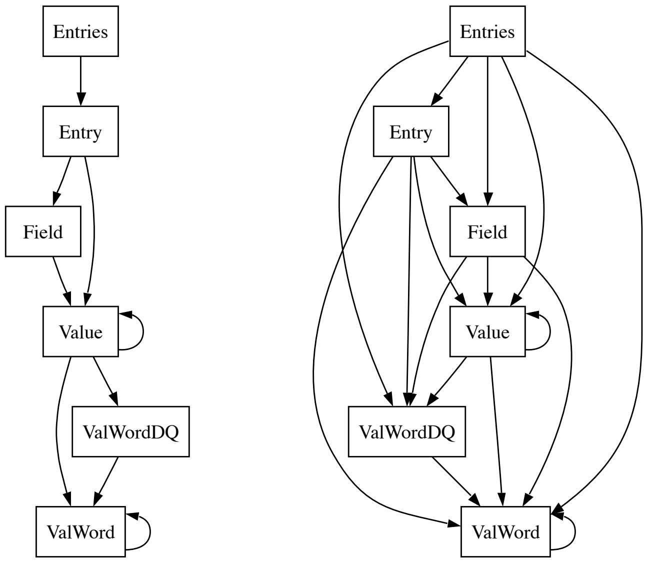

Fig. 2. Immediate successors graph and transitive closure graph for BibTeX.

From Table 1 we can also observe that grammars for the same language have different metric values. Taking as example the two Java 1.5 grammars specified in ANTLR and SDF, we notice that the number of terminals are different, however this difference is small. The same can be also observed for the C# grammar from ANTLR when compared with the values reported in [3]. In contrast, when comparing the two C++ grammars, from SD and as reported in [3] we note that the former contains about 50% more terminals than the latter. This provides stronger evidence that the two grammars in fact parse different language subsets.

4.2 Tree Impurity metric analysis: TIMP vs. TIMPi

In table 1 we can observe that there are significant differences between the two tree impurity metrics.

Tree impurity metric [17] measures to which extent the graph deviates from a tree structure. A value of 0% means that the graph is a tree, while a value of 100% means the graph is a fully connected graph. The TIMP metric follows the definition of [3], where the formula is applied to the transitive closure of the grammar graph. TIMPi, introduced in SdfMetz, implements the same formula but applied to the successor graph instead of the transitive closure of that graph.

For tree impurity for immediate successor graphs (TIMPi), the values lie between 0.2% and 5.9%, except for BibTeX grammar. For tree impurity for transitive successor graphs (TIMP), the values range between 22.2% and 100.0%.

For the BibTeX grammar, we can observe that TIMP has a value 100% where TIMPi reports just 20%. Hence, according to the definition presented in [3], the

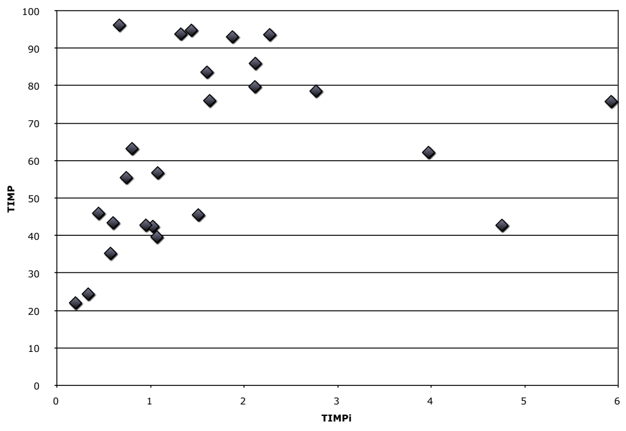

Fig. 3. Scatter plot between TIMP and TIMPi for all grammars except BibTeX.

BibTeX grammar is 100% tree "impure". To visualize this we used the SdfMetz, and the execution is done as following:

```
$ SdfMetz -g -o BibTeX-Graph.dot -i SDFGrammars/bibtex.def
$ SdfMetz -t -o BibTeX-TcGraph.dot -i SDFGrammars/bibtex.def
```

On the first execution, the `-g` argument was passed to request the graph of the grammar in `dot` format. On the second execution, we passed the `-t` argument which computes the transitive closure of the graph and outputs the result in `dot` format. The result of these graphs is presented in Figure 2.

From the graph of the immediate successors we can see that two of ten nodes have multiple dependencies (Value and ValWord) which explain the deviation from the tree shape and a result of 20% for tree impurity. In contrast, in the transitive closure graph, we can observe that all nodes are connected which explains the value of 100%.

So, for a single grammar we have observed an important difference between the two tree impurity metrics. But how are these grammars related in general? To investigate this, we have decided to use the metrics computed from all grammars except from BibTeX and then plot the TIMP metric against TIMPi. BibTeX was not included because it is an outlier, tree impurity is very high due to the small number of non-terminals. The result of this is shown in Figure 3. From the scatter plot we can observe that there is high dispersion. To quantify the dispersion we have determine correlation using the Spearman rank correlation analysis. The Spearman rank correlation coefficient between TIMP and TIMPi is 0.401 with high significance ($p < 0.01$), which means that the two metrics are very weakly correlated.

We can conclude that, in fact, the two metrics measure different things. The BibTeX example indicates that TIMP is more sensitive to long chains of dependencies. In our opinion, the TIMPi metric is more intuitive and more useful to quantify the degree of connectedness in a grammar.

5 Conclusion

We have presented SdfMetz, a tool for grammar quantification and visualization, supporting a total of thirty one metrics for four different formalisms (SDF, DMS, Bison and ANTLR). We described the tool functionality, implementation and how to use the tool for grammar comparison and analysis. Through two scenarios, we have reported metrics for 30 grammars, and showed how the tool can be valuable to gain insight about grammar properties. Furthermore, we have contributed to the grammar engineering body of knowledge showing how metrics can be analyzed and validated.

Availability The SdfMetz tool is developed as open source software and is available from: http://wiki.di.uminho.pt/twiki/bin/view/Research/PURe/SdfMetz. A spreadsheet containing metrics values for more than 50 grammars and the full metrics for the grammars used in the paper can be found in the first author's web page: http://wiki.di.uminho.pt/twiki/bin/view/Personal/Tiago/Publications.

Acknowledgements The first author is supported by the *Fundação para a Ciência e a Tecnologia*, grant SFRH/BD/30215/2006.

References

1. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4) (2005) 316–344
2. Klint, P., Lämmel, R., Verhoef, C.: Toward an engineering discipline for grammarware. ACM Trans. Softw. Eng. Methodol. **14**(3) (2005) 331–380
3. Power, J.F., Malloy, B.A.: A metrics suite for grammar-based software: Research articles. J. Softw. Maint. Evol. **16**(6) (2004) 405–426
4. Parr, T.J., Quong, R.W.: ANTLR: A predicated-LL(k) parser generator. Software & Practice and Experience **25**(7) (1995) 789–810
5. Heering, J., Hendriks, P., Klint, P., Rekers, J.: The syntax definition formalism SDF — Reference manual. SIGPLAN Notices **24**(11) (1989) 43–75
6. Baxter, I., Pidgeon, P., Mehlich, M.: DMS: Program transformations for practical scalable software evolution. In: Proceedings of the International Conference on Software Engineering, IEEE Press (2004) 625–634
7. Alves, T., Visser, J.: Development of an industrial strength grammar for VDM. Technical Report DI-PURe-05.04.29, Universidade do Minho (2005)
8. Nejmeh, B.: NPATH: a measure of execution path complexity and its applications. Commun. ACM **31**(2) (1988) 188–200
9. Koutsofios, E.: Drawing graphs with *dot*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA (November 1996)
10. Lämmel, R., Visser, J.: A Strafunski Application Letter. In Dahl, V., Wadler, P., eds.: Proc. of Practical Aspects of Declarative Programming (PADL'03). Volume 2562 of LNCS., Springer-Verlag (2003) 357–375

11. Herington, D.: Hunit - haskell unit testing http://hunit.sourceforge.net/.
12. Parr, T.J.: ANTLR parser generator http://www.antlr.org/.
13. : GCC, the GNU compiler collection http://gcc.gnu.org/.
14. Baxter, I.D.: Semantic Designs, Inc., home page http://www.semdesigns.com/.
15. de Jonge, M.: The online Grammar Base http://web.archive.org/web/20050908044013/http://www.cs.uu.nl/~mdejonge/grammar-base/.
16. Kuipers, T.: Software Improvement Group home page http://www.sig.nl/.
17. Fenton, N., Pfleeger, S.: Software metrics: a rigorous and practical approach. PWS Publishing Co., Boston, MA, USA (1997) 2nd edition, revised printing.