DRAFT 15 February 2007

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

Tiago L. Alves [1]

European Space Agency - ESOC, Darmstadt, Germany

Joost Visser [2]

DI-CCTC, Universidade do Minho, Braga, Portugal

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 input formalisms of DMS, ANTLR, and Yacc. 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 analysis and comparison.

Key words: Metrics, Graphs, Syntax definition.

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 [7] (DSLs), e.g. in the context of model-driven engineering (MDE), again emphasises grammars as primary software artifacts.

Grammar engineering [4] aims to apply solid software engineering techniques to grammars. Such techniques include version control, static analysis,

[1] Email: `tiago.alves@esa.int`

[2] Email: `joost.visser@di.uminho.pt`

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.

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 [9] 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.

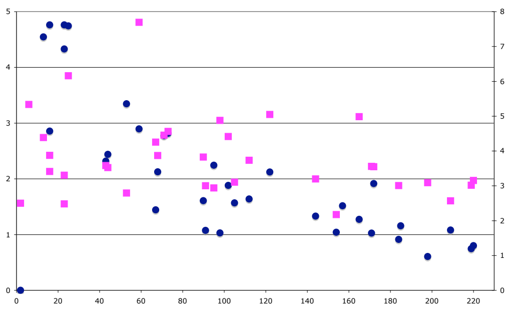
We have implemented a tool for grammar quantification and visualization, named SdfMetz. The suite of metrics calculated by SdfMetz is a significant extension of those defined by Power and Malloy, and apart from Yacc-like BNF dialects, the tool accepts ANTLR, SDF [3], and DMS [2] grammars. We also implemented *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 [1].

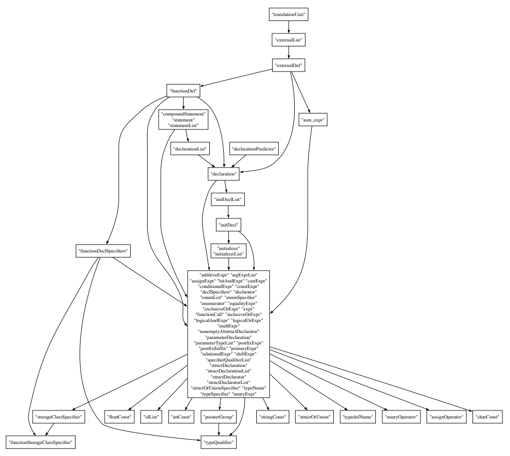
2 Tool functionality

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, Yacc, Bison, and DMS grammars. A wide range of metrics can be emitted:

- All the metrics defined in [9]: 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; and count of non-terminals in the largest grammatical level.
- The Halstead Effort metric of [9], as well as futher Halstead metrics: the underlying count of operators and distinct operands, distinct and total; and other derived metrics such as length, volume, difficulty, level, and time.
- The tree impurity metric, applied to the transitive closure of the successor graph [9], as well as tree impurity on the successor graph itself
- An adaptation to grammars of the NPath metric [8], which, like cyclometric complexity measures possible paths, but gives more weight to nested choices.
- Ambiguity metrics, specific to SDF: counts of follow restrictions, associativity and preference attributes, reject and prioritized productions.



(a) Scatter plot of non-terminals (x-axis), tree impurity without transitive closure (left y-axis, circles), and NPath (right y-axis, squares) for 42 grammars.



"translationUnit"
"externalList"
"externalDef"
"functionDef"
"compoundStatement" "statement" "statementList"
"asm_expr"
"declarationList"
"declarationPredictor"
"declaration"
"initDeclList"
"initDecl"
"functionDeclSpecifiers"
"initializer" "initializerList"
"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"
"storageClassSpecifier"
"floatConst"
"idList"
"intConst"
"pointerGroup"
"stringConst"
"structOrUnion"
"typedefName"
"unaryOperator"
"assignOperator"
"charConst"
"functionStorageClassSpecifier"
"typeQualifier"

(b) Strong connected component graph for an ANTLR grammar of the C language.

Fig. 1. Examples of output processed with (a) Excel and (b) `dot`.

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. Figure 1a gives an example.

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 format of `dot` [5]. An example is given in Figure 1b.

3 Tool implementation

SdfMetz was developed in Haskell and SDF, making essential use of the Strafunski bundle [6] 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, bags and graphs and for exporting to `dot` format. The tool was extensively unit tested using the HUnit framework.

4 Tool demonstration

The demonstration of the tool will consist of the following elements.

Run We will 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 will 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 will explain the various metrics emitted by SdfMetz. We will explain how these metrics can be visualized and interpreted.

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

Availability The SdfMetz tool is developed as open source software and is available from: `http://wiki.di.uminho.pt/wiki/bin/view/PURe/SdfMetz`.

References

[1] T. Alves and J. Visser. Development of an industrial strength grammar for VDM. Technical Report DI-PURe-05.04.29, Universidade do Minho, 2005.

[2] I. Baxter, P. Pidgeon, and M. Mehlich. DMS: Program transformations for practical scalable software evolution. In *Proceedings of the International Conference on Software Engineering*. IEEE Press, 2004.

[3] J. Heering, P.R.H. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism SDF — Reference manual. *SIGPLAN Notices*, 24(11):43–75, 1989.

[4] P. Klint, R. Lämmel, and C. Verhoef. Toward an engineering discipline for grammarware. *ACM Trans. Softw. Eng. Methodol.*, 14(3):331–380, 2005.

[5] E. Koutsofios. Drawing graphs with *dot*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, November 1996.

[6] R. Lämmel and J. Visser. A Strafunski Application Letter. In V. Dahl and P. Wadler, editors, *Proc. of Practical Aspects of Declarative Programming (PADL'03)*, volume 2562 of *LNCS*, pages 357–375. Springer-Verlag, 2003.

[7] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[8] B.A. Nejmeh. NPATH: a measure of execution path complexity and its applications. *Commun. ACM*, 31(2):188–200, 1988.

[9] J.F. Power and B.A. Malloy. A metrics suite for grammar-based software. In *Journal of Software Maintenance and Evolution*, volume 16, pages 405–426. Wiley, November 2004.