Towards Formal Grammars As Institutions

Mikhail Barash and Magne Haveraaen

Bergen Language Design Laboratory, University of Bergen, Norway
mikhail.barash@uib.no, magne.haveraaen@uib.no

1 Introduction

Phrase structure grammars, introduced by Chomsky in 1950s, have found their applications across many areas of computer science, especially in specification of syntax of programming languages. Many specialized and extended grammar models have been used for this purpose: Floyd argued [10] about the expressive power of context-free grammars soon after Chomsky's seminal paper, van Wijngaarden et al. [16] constructed a two-level grammar to specify syntax and static semantics of Algol 68, Cleveland and Uzgalis [6] presented a two-level grammar that additionally defined the dynamic semantics of Algol 68. Two-level grammars, as well as many other grammar models are, however, computationally universal, which makes parsing algorithms for them unpractical. This caused the research on formal grammars to focus primarily on the family of context-free grammars and parsing algorithms for them.

Context-free grammars are a core component in many language-related *tools*, such as parsers, compilers, Integrated Development Environments (IDEs), parser generators, and language workbenches [9, 4]. While these tools *start off* of a grammar that specifies the syntax of a programming language in question, they *augment* the grammar in order to solve the task at hand. Thus, parsers may construct symbol tables or may annotate a grammar with type information, IDEs may implement a special treatment for error recovery or may have type annotations different from what the underlying parser has, language workbenches may transform grammars to equivalent syntax definitions in other formalisms [4], and so on. Despite the fact these tools are ubiquitous, the interaction between grammars and such tools has not been systematically presented. Of interest are questions of grammar and tool co-evolution, grammar reuse [7], composition [8], transformation [17].

Our proposal is to use the theory of institutions [11] to systematically explore the relationship between various grammar formalisms. Institutions give a ground for structured mechanisms to express augmentations of grammars (such as applying set-theoretic operations on them or specifying grammar composition), and are instrumental to the study of modularity in grammar specifications.

2 Institution of Abstract Syntax Grammars

Context-free grammars are tuples of the form $G = (\Sigma, N, R, S)$, where Σ is a finite alphabet of language symbols ("terminal symbols"), N is a finite alphabet of language constructs ("non-terminal symbols"), R is a finite set of rules of the form $A \to \alpha$ with $A \in N$, $\alpha \in (\Sigma \cup N)^*$, and $S \in N$ is the start symbol.

In this extended abstract, we consider a subclass of context-free grammars—*abstract syntax grammars*—that allow trivial parsing: the right-hand side part of every rule starts with a unique terminal symbol, that is, every rule is of the form $A \to \sigma_A \alpha$, with $\sigma_A \in \Sigma$, $\alpha \in (\Sigma \cup N)^*$, $A \in N$. This setting allows us to lift our focus from the problem of parsing; we can then

extend the results discussed in the present extended abstract to the general case of context-free grammars.

Example 1. Consider a language of arithmetic expressions in prefix notation; sample strings in this language are "**;** **int** x **;** **int** y **;** **sum** (**val** x)(**mul** (**val** x)(**val** y))" and "**;** **bool** b**;** **bool** c **;** **sum** (**val** b)(**val** c)". This language can be defined by the following abstract syntax grammar.

$S \to$	**;** E $S \mid \varepsilon$	
$E \to$	**int** I	*integer variable declaration*
$\mid$	**bool** I	*Boolean variable declaration*
$\mid$	**sum** $(E)(E)$	*sum / disjunction*
$\mid$	**mul** $(E)(E)$	*multiplication / conjunction*
$\mid$	**val** I	*value of a variable*

An *institution* [11, 12] is a tuple $I = (\mathbf{Sig}, F, M, \models)$, where **Sig** is a category of morphisms, $F : \mathbf{Sig} \to \mathbf{Set}$ is a functor of formulae, $M : \mathbf{Sig}^{op} \to \mathbf{CAT}$ is a contravariant functor of models, and $\models$ is a satisfaction relation, which for every object $\mathcal{S}$ of **Sig** defines a relation $\models_{\mathcal{S}} \subseteq M(\Sigma) \times F(\Sigma)$, such that the following satisfaction relation holds: for every morphism $s : \mathcal{S} \to \mathcal{S}'$ in **Sig**, $\Big(M(s^{op} : \mathcal{S}' \leftarrow \mathcal{S})\Big)(\mu') \models_{\mathcal{S}} \varphi \quad \Leftrightarrow \quad \mu' \models_{\mathcal{S}'} \Big(F(s : \mathcal{S} \to \mathcal{S}')\Big)(\varphi)$, where μ' is an object of $M(\mathcal{S}')$, and $\varphi \in F(\mathcal{S})$.

We can construct various institutions based on a perspective in question. For example, **Sig** can be a category of morphisms on the symbols of the grammar, M can represent the set of strings generated by this grammars, and F can define the grammar formalism. The satisfaction relation $\models$ shall then represent the fact that a string, which can be parsed according to an original grammar, can also be parsed according to a modified grammar, based on a morphism on an original grammar, which induces a corresponding morphism on a modified grammar.

Example 2. Consider a language L_1 of nested balanced parentheses, where each parenthesis additionally has an index, and consider a variation L_0 of this language, where the indexes are insignificant. An example of a valid string in the language L_1 is $({}_a({}_{bb})({}_{cc})_a)$, and an example of a valid string in L_0 is $({}_a({}_{bc})({}_{de})_f)$. The language L_1 abstracts well-formed XML documents, and L_0 abstracts XML documents where opening and closing tags are not necessarily matched, but documents are still well-formed tree structures. In an institution, let F represent a specification of a grammar that defines the language L_1, M represent a set of terms for a grammar that defines L_0. Then $\models$ represents the fact that a string, which can be parsed according to a grammar for L_0, can also be parsed according to a grammar for L_1.

This same institution can be used to explore a correspondence between grammars for an untyped and a typed version of a language for arithmetical expressions from Example 1. There, a morphism on grammar rules would specify how rules of the original grammar have to be changed in order to incorporate the typing information directly. We anticipate that polynomially parsable extensions of context-free grammars, such as conjunctive [13], Boolean [14], and grammars with contexts [5], can be of a benefit in this setting.

Other institutions may be constructed to explore modular grammar specifications and language composition [8]. Yet another direction is to construct institutions for morphisms on the symbols of the grammar considered together with bananas [1]; this would allow exploring the setting of language evolution [8].

Using Syntactic Theory Functors [12] would enable creating syntactic structuring mechanisms for a given grammar specification formalism in order to, for example, extend signatures or change notation in arbitrary ways; this would allow combining "incompatible" layers of morphisms.

Considering language tools on top of the institutions will enable treating tools first-class. It would be possible to translate the behaviour of the tools between institutions, or compose tools' behaviour: given a language L defined by grammar G, a tool T based on G, and an evolved language L' defined by a modified grammar $G' = \tau(G)$, then the tool T' for L' could be obtained by performing a composition-like[1] operation on tool T and the grammar G'. Conversely, given an evolved tool T' and an original grammar G, the tool for G can be obtained as follows: $T = T' \circ \tau^{op}$.

References

[1] J. Andersen, C. Brabrand, D. R. Christiansen, *Banana Algebra: Compositional syntactic language extension*. Sci. Comput. Program. 78:10. 1845–1870. 2013.

[2] M. Barash, *Programming language specification by a grammar with contexts*. NCMA 2013. 51–67.

[3] M. Barash, *A New Life for Legacy Language Definition Approaches?* STAF Workshops 2020. 75–84.

[4] M. Barash, *Vision: The Next 700 Language Workbenches*, SLE 2021, to appear.

[5] M. Barash, A. Okhotin, *An extension of context-free grammars with one-sided context specifications*. Inf. Comput. 237. 268–293. 2014.

[6] J. Cleaveland, R. Uzgalis, *Grammars for programming languages*. Elsevier. 1977.

[7] T. Cleenewerck, K. Czarnecki, J. Striegnitz, M. Völter, *Evolution and Reuse of Language Specifications for DSLs (ERLS)*. ECOOP Workshops 2004. 187–201.

[8] S. Erdweg, P. G. Giarrusso, T. Rendel, *Language composition untangled*. LDTA 2012. 7.

[9] S. Erdweg, T. van der Storm, M. Völter, L. Tratt, R. Bosman, et al., *Evaluating and comparing language workbenches: Existing results and benchmarks for the future*. Comput. Lang. Syst. Struct. 44. 24–47. 2015.

[10] R. W. Floyd, *On the nonexistence of a phrase structure grammar for ALGOL 60*. Commun. ACM 5(9). 483–484. 1962.

[11] J. A. Goguen, R. M. Burstall, *Institutions: Abstract Model Theory for Specification and Programming*. J. ACM 39(1). 95–146. 1992.

[12] M. Haveraaen, M. Roggenbach, *Specifying with syntactic theory functors*. J. Log. Algebraic Methods Program. 113. 100543. 2020.

[13] A. Okhotin, *Conjunctive Grammars*. J. Autom. Lang. Comb. 6(4). 519–535. 2001.

[14] A. Okhotin, *Boolean grammars*. Inf. Comput. 194(1). 19–48. 2004.

[15] A. Okhotin, *On the existence of a Boolean grammar for a simple programming language*. AFL 2005.

[16] A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, Cornelis H. A. Koster, et al., *Revised Report on the Algorithmic Language ALGOL 68*. Acta Informatica 5. 1–236. 1975.

[17] V. Zaytsev, A. H. Bagge, *Parsing in a Broad Sense*. MoDELS 2014. 50–67.

[1] This operation should be more expressive than a mere composition, which is not well-defined since the morphism τ is not surjective.