Iguana: A Practical Data-Dependent Parsing Framework

Ali Afroozeh   Anastasia Izmaylova

Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

{ali.afroozeh, anastasia.izmaylova}@cwi.nl

Abstract

Data-dependent grammars extend context-free grammars with arbitrary computation, variable binding, and constraints. These features provide the user with the freedom and power to express syntactic constructs outside the realm of context-free grammars, e.g., indentation rules in Haskell and type definitions in C. Data-dependent grammars have been recently presented by Jim *et al.* as a grammar formalism that enables construction of parsers from a rich format specification. Although some features of data-dependent grammars are available in current parsing tools, e.g., semantic predicates in ANTLR, data-dependent grammars have not yet fully found their way into practice.

In this paper we present Iguana, a data-dependent parsing framework, implemented on top of the GLL parsing algorithm. In addition to basic features of data-dependent grammars, Iguana also provides high-level syntactic constructs, e.g., for operator precedence and indentation rules, which are implemented as desugaring to data-dependent grammars. These high-level constructs enable a concise and declarative way to define the syntax of programming languages. Moreover, Iguana's extensible data-dependent grammar API allows the user to easily add new high-level constructs or modify existing ones. We have used Iguana to parse various real programming languages, such as OCaml, Haskell, Java, and C#. In this paper we describe the architecture and features of Iguana, and report on its current implementation status.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Parsing

Keywords Data-dependent grammars, GLL, disambiguation

1. Introduction

Parsing is the first step in many tasks, such as compiler construction and static analysis, that deal with source code. When building a (domain-specific) programming language, it is desirable to quickly build a parser and spend most of the effort in other phases such as name resolution and type checking. Despite the long investment in theory and practice of parsing, constructing parsers remains a difficult task, often left to the experts. Syntax of most programming languages cannot be directly expressed using current parsing tools that are based on pure (deterministic) context-free grammars, and there is a need for (manual) grammar modification, and various hacks in the lexer and parser [2].

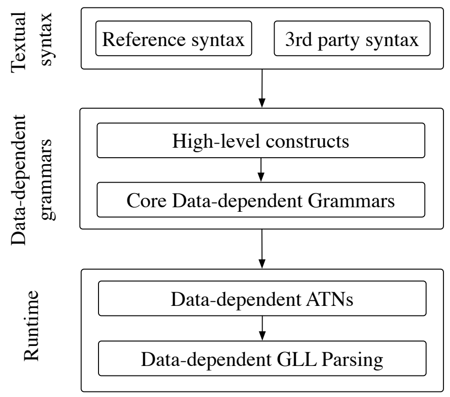

Figure 1. The architecture of Iguana.

We advocate a declarative approach [5] to syntax definition, where the user defines the syntax as a context-free grammar, and uses disambiguation constructs to specify (un)desired parse trees. In our earlier work [2] we described our vision of a parsing framework that can deal with many challenges of parsing programming languages. We base our parsing framework on data-dependent grammars [4], instead of pure context-free grammars. Data-dependent grammars extend context-free grammars with arbitrary computation, variable binding and constraints. In essence, these features allow the user to simulate handwritten parsers. Our data-dependent grammars are implemented on top of the Generalized LL (GLL) parsing algorithm [6]. In particular, we extended the original GLL algorithm to support environment manipulation and return values [2].

Data-dependent grammars are a rich format specification for building parsers. However, for expressing many aspects of programming languages, they are rather low-level. In our previous work [2], we proposed to use data-dependent grammars as an intermediate layer for parser-independent implementation of disambiguation mechanisms. We demonstrated how several high-level syntactic constructs, e.g., operator precedence and indentation-sensitive constructs [2, 3], can be desugared to data-dependent grammars. This provides a uniform view on various disambiguation mechanisms, and allows the user to add new syntactic constructs without the need to modify the machinery of a parsing algorithm.

In this paper we present Iguana, our data-dependent parsing framework. Iguana has been developed during the last two years and been used in evaluating our recent work [1–3] using grammars of real programming languages. We present the architecture and features of Iguana, and discuss its current implementation status.

CC'16, March 17–18, 2016, Barcelona, Spain
ACM. 978-1-4503-4241-4/16/03...$15.00
http://dx.doi.org/10.1145/2892208.2892234

2. Architecture

We designed Iguana to be extensible and flexible to serve as a standalone tool, and also as a parsing library for different language design and implementation tools. Figure 1 shows the architecture of Iguana, which consists of the following three layers:

Runtime Iguana is implemented on top of our version of GLL parsing. In our previous work [1], we proposed a modification to the Graph Structured Stack (GSS) of the original GLL algorithm, and showed that the new GSS makes GLL parsers significantly faster. To implement data-dependent features, we presented an extension of GLL that supports environment manipulation and return values [2]. Moreover, Iguana uses an interpretive version of GLL which operates on an in-memory representation of a grammar (data-dependent ATNs [2]). This eliminates the need for code generation and compilation cycles after every change to the grammar.

Data-dependent grammars This layer provides a data type and API to construct an abstract representation of a data-dependent grammar. A data-dependent grammar can be directly defined using this API and later transformed to a data-dependent ATN to be interpreted. This layer also provides an API for transformation of data-dependent grammars, for example, EBNF-to-BNF transformation. Our data-dependent grammars support all the features of the original data-dependent grammars [4], and some useful extensions such as return values [2]. We call these features the *core* features. In addition, we provide a set of high-level syntactic constructs, e.g., for operator precedence and indentation rules, and transformations that desugar them to the core data-dependent grammars. Our parsing framework is extensible: the language engineer can add new high-level constructs, by using our API and providing the necessary desugaring to data-dependent grammars.

Textual syntax This layer provides a textual (concrete) syntax for defining data-dependent grammars. Our textual syntax is faithful to the original syntax of data-dependent grammars and provides a grammar-centric, clean way of defining data-dependent grammars. Tool builders who wish to design their own syntax for grammar definition can integrate Iguana into their tool, as a parsing backend for syntax definition, using our abstract representation of data-dependent grammars. Iguana will also be used in the future as the parsing library for the Rascal meta-programming language[1].

3. Features

Full context-free grammars Iguana supports all context-free grammars. This frees the user from the expressivity limitations of a deterministic parsing technique, such as LL(k), and enables modularity and composability.

Data-dependent grammars Iguana supports the following core features of data-dependent grammars: parametrized nonterminals, variable binding, and constraints. In addition, Iguana supports labeled symbols and return values.

Single-phase parsing Iguana employs a single-phase parsing strategy [2], in which there is no separate lexing phase. Using Iguana, terminals can be defined to the level of characters or using regular expressions, providing scannerless parsing [8] or context-aware scanning [7], respectively. Single-phase parsing effectively presents the parsing context to the lexing phase, avoiding the problems of a separate lexer in determining the type of tokens that have different meanings in different contexts.

High-level disambiguation constructs Iguana supports the following high-level (disambiguation) constructs that are implemented as desugaring to data-dependent grammars [2]: (1) lexical disambiguation filters, such as follow and precede restrictions and keyword exclusion; (2) operator precedence and associativity constructs, such as `>`, **`left`**, **`right`**, and **`nonassoc`**; and (3) indentation sensitive constructs, such as **`offside`**, **`align`** and **`ignore`**.

Other features Iguana also supports some other useful features, such as global variables. For example, to deal with type definitions in C, which require maintaining a map of definitions during parsing, it is more convenient to define a global variable, rather than passing and returning values to/from all reachable nonterminals. We also demonstrated how data dependency can be used to deal with C# conditional directives during parsing, without the need for a separate preprocessor. All of these features are implemented via desugaring to the core data-dependent grammars [2].

4. Implementation Status and Future Work

Iguana has been our research platform for the last two years, and is under active development. Iguana is implemented mainly in Java, and partly in Scala, for the parts that require transformation and traversal, e.g., grammars and parse trees. The core part of Iguana (data-dependent GLL parsing) is stable, and most effort are now put in developing an IntelliJ plugin. Iguana is available as an open source project at `http://iguana-parser.github.io`, with documentation, links to the Github repositories and grammars.

Grammars We have written Iguana grammars for some major programming languages (from their reference manuals), such as Java, C#, C, OCaml, Haskell, and XML. We used these grammars in the evaluation of our previous work [1–3]. The grammars are written in the Rascal front-end (we implemented the first textual front-end for Iguana as an extension of Rascal), and we are gradually converting them to the reference syntax as well.

Performance The results of running Iguana on grammars of real programming languages are shown in our previous work [1–3]. Iguana runs nearly linearly on grammars of programming languages, and keeps the cubic bound on highly ambiguous grammars.

Tool support We are developing an IntelliJ plugin to facilitate the development of Iguana grammars. The plugin provides common features such as syntax highlighting, navigation, outline views, basic refactoring and some static grammar validation.

Future work Currently Iguana reports the source location of a parse error, but does not provide error recovery facilities. Error recovery is future work. We also plan to work on an incremental version of Iguana, especially for use in an interactive grammar development environment.

References

[1] A. Afroozeh and A. Izmaylova. Faster, Practical GLL Parsing. In *Compiler Construction*, CC'15, pages 89–108. Springer, 2015.

[2] A. Afroozeh and A. Izmaylova. One Parser to Rule Them All. In *Onward! 15*, pages 151–170. ACM, 2015.

[3] A. Afroozeh and A. Izmaylova. Operator Precedence for Data-Dependent Grammars. In *PEPM'16*, pages 13–24. ACM, 2016.

[4] T. Jim, Y. Mandelbaum, and D. Walker. Semantics and Algorithms for Data-dependent Grammars. In *POPL'10*, pages 417–430, 2010.

[5] L. C. Kats, E. Visser, and G. Wachsmuth. Pure and Declarative Syntax Definition: Paradise Lost and Regained. In *OOPSLA'10*, pages 918–932. ACM, 2010.

[6] E. Scott and A. Johnstone. GLL Parse-tree Generation. *Science of Computer Programming*, 78(10):1828–1844, Oct. 2013.

[7] E. R. Van Wyk and A. C. Schwerdfeger. Context-aware Scanning for Parsing Extensible Languages. GPCE '07, pages 63–72. ACM, 2007.

[8] E. Visser. Scannerless Generalized-LR Parsing. Technical report, University of Amsterdam, 1997.

[1] `http://www.rascal-mpl.org`