Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 193 (2007) 3–18
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Executable Grammars in Newspeak

Gilad Bracha [1]

Cadence Design Systems
San Jose, California, USA

Abstract

We describe the design and implementation of a parser combinator library in Newspeak, a new language in the Smalltalk family. Parsers written using our library are remarkably similar to BNF; they are almost entirely free of solution-space (i.e., programming language) artifacts. Our system allows the grammar to be specified as a separate class or mixin, independent of tools that rely upon it such as parsers, syntax colorizers etc. Thus, our grammars serve as a shared executable specification for a variety of language processing tools. This motivates our use of the term *executable grammar*. We discuss the language features that enable these pleasing results, and, in contrast, the challenge our library poses for static type systems.

Keywords: Parser combinators, dynamic programming languages, Smalltalk, reflection

1 Introduction

Newspeak is a programming language derived from Smalltalk, currently under development. We have developed a parser combinator library in Newspeak, and find that Smalltalk and Newspeak have properties that make them very well suited for the implementation and use of such a library.

The concept of parser combinators has a long history in functional programming [11,13,21]. From an object-oriented perspective, the basic idea is to view the operators of BNF as methods, known as combinators, that operate on objects representing productions of a grammar. Each such object is a parser that accepts the language generated by a particular production. The results of the combinator invocations are also such parsers.

To make this concrete, lets look at a fairly standard rule for identifiers:

[1] Email: gilad@cadence.com

doi:10.1016/j.entcs.2007.10.004

```
id = letter (letter | digit)*
```

We can express this in Newspeak as:

```
id = letter, (letter | digit) star
```

Assume letter is a parser that accepts a single letter and digit is a parser that accepts a single digit. The subexpression letter | digit invokes the method | on the parser that accepts a letter, with an argument that accepts a digit. The result will be a parser that accepts either a letter or a digit.

We then invoke the method star on the result

```
(letter | digit) star
```

which yields a parser that accepts zero or more occurrences of either a letter or a digit.

We pass this parser as an argument to the method "," which we invoke on letter. The "," method is the sequencing combinator (which is implicit in BNF). It returns a parser that first accepts the language of the receiver and then accepts the language of its argument. In our example the result accepts a single letter, followed by zero or more occurrences of either a letter or a digit, just as we'd expect. Finally, we bind this result to id.

You should now have a basic intuition for the way parser combinators work, and a feel for what executable grammars look like in our framework. Here is a roadmap to the rest of this paper:

Section 2 shows a complete example of an executable grammar. Then, sections 3 through 6 discuss a number of design issues and features of our framework. In particular, section 4 showcases the modular separation between grammar and processing. The paper presents ideas for future improvements (section 7); In section 7.1, we analyze the type safety of grammars and parsers constructed using the library, and what mechanisms are needed to statically check them. We discuss related work (section 8), and conclude with section 9.

2 A Complete Example

We now consider a small grammar, and show how to represent it in our framework. In doing so, we'll highlight some interesting design issues. We will use the following grammar

```
expression = id
returnStatement = ^ expression.
```

This grammar relies on terminal symbols id, letter and digit defined as: