A Parser in my Pocket

Martin McClure

ESUG 2010

GemStone®

vmware®

Once upon a time...

Parsing Expression Grammars: A Recognition-Based Syntactic Foundation

Bryan Ford
Massachusetts Institute of Technology
Cambridge, MA
baford@mit.edu

Abstract

For decades we have been using Chomsky's generative system of grammars, particularly context-free grammars (CFGs) and regular expressions (REs), to express the syntax of programming languages and protocols. The power of generative grammars to express ambiguity is crucial to their original purpose of modelling natural languages, but this very power makes it unnecessarily difficult both to express and to parse machine-oriented languages using CFGs. Parsing Expression Grammars (PEGs) provide an alternative recognition-based formal foundation for describing machine-oriented languages which solves the ambiguity problem by not introducing ambiguity in the first place. Where CFGs express nondeterministic choice between alternatives, PEGs instead use *prioritized* choice. PEGs address frequently felt expressiveness limitations of

1 Introduction

Most language syntax theory and practice is based on *generative* systems, such as regular expressions and context-free grammars, in which a language is defined formally by a set of rules applied recursively to generate strings of the language. A *recognition-based* system, in contrast, defines a language in terms of rules or predicates that decide whether or not a given string is in the language. Simple languages can be expressed easily in either paradigm. For example, $\{s \in \mathtt{a}^* \mid s = (\mathtt{aa})^n\}$ is a generative definition of a trivial language over a unary character set, whose strings are "constructed" by concatenating pairs of a's. In contrast, $\{s \in \mathtt{a}^* \mid (|s| \bmod 2 = 0)\}$ is a recognition-based definition of the same language, in which a string of a's is "accepted" if its length is even.

While most language theory adopts the generative paradigm, most practical language applications in computer science involve the opposite and structural decomposition, or *parsing*, of strings. ... generative definitions to practical recogniz... ...ounding library of parsing algo... ...ars [9].

Executable Grammars in Newspeak

Gilad Bracha

August 18, 2007

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.

Smalltalk PEG-Based Parsers

- Xstreams PEG parser
- OMeta
- PetitParser
- ...?

The rest of this talk

- Grammars, PEGs
- OMeta2 with Examples
- Parser Combinators and PetitParser

Context-free grammars

conditional ::=

“if” cond “then” statement

| “if” cond “then” statement “else” statement

Works for generation,

but not sufficient for recognition

PEG
Ordered Choice

conditional =

“if” cond “then” statement “else” statement

| “if” cond “then” statement

Makes recognition unambiguous

Example 1

File Server Incremental Backup


OPEN
RESET
MODE
A
MODE
B

```
time ( rsync -aHxi --numeric-ids --delete
--link-dest=/mnt/backup/caboodle1/2010-08-03
/caboodle1/ /mnt/backup/caboodle1/2010-09-08 )
```

For more information on Ometa:

http://tinlizzie.org/ometa/
http://tinlizzie.org/~awarth/ometa/ometa2.html
http://www.squeaksource.com/OMeta/

http://www.lukas-renggli.ch/blog/petitparser-1

http://source.lukas-renggli.ch/petit.html

BNF

ID ::= letter { letter | digit } ;

BNF

ID ::= letter { letter | digit } ;

Ometa2

id = letter (letter | digit)*

BNF

```
ID ::= letter { letter | digit } ;
```

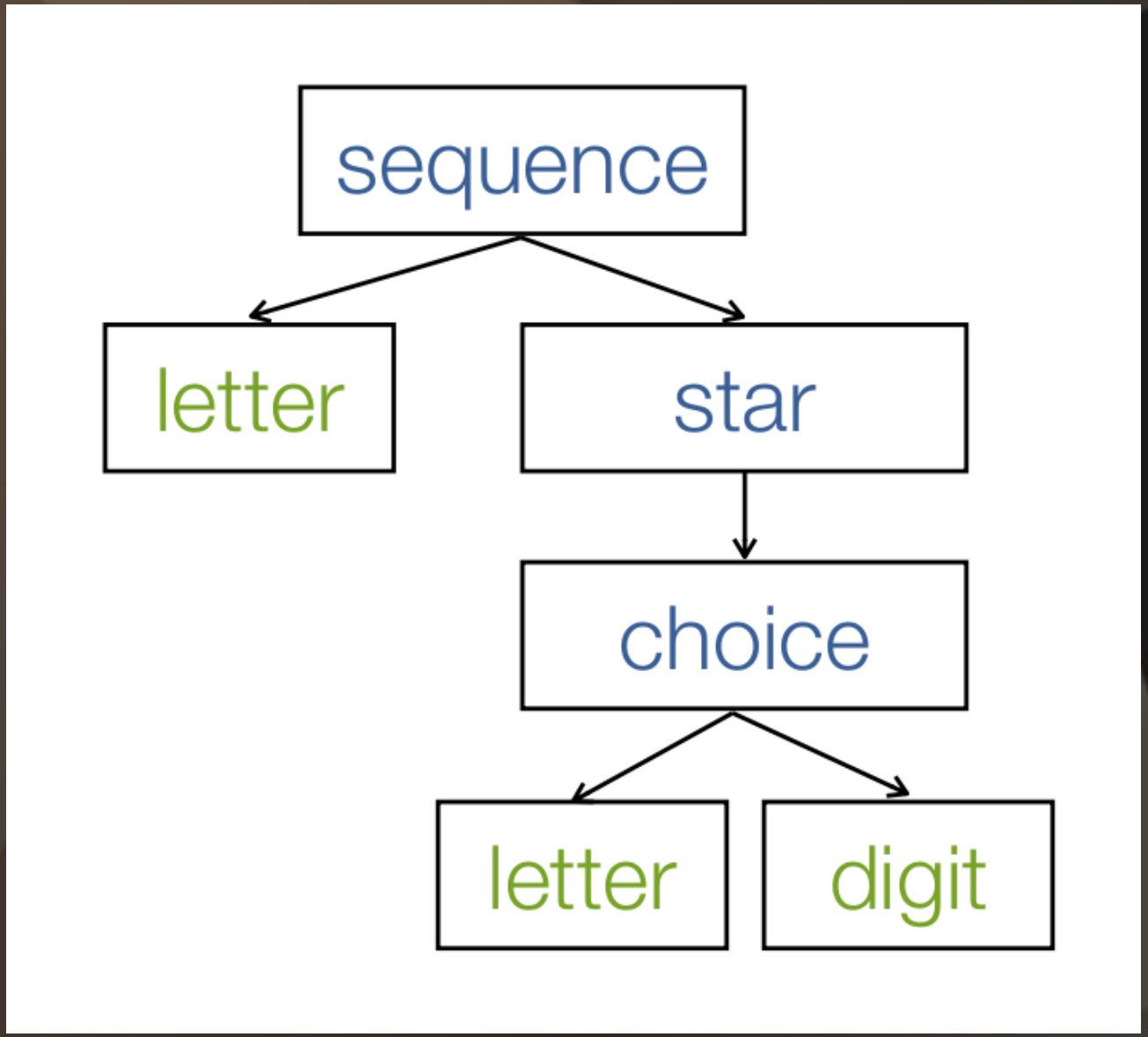
Ometa2

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

PetitParser

```
id := #letter asParser ,
    (#letter asParser / #digit asParser) star
```

```
id := #letter asParser ,
    (#letter asParser / #digit asParser) star
```



¿questions?

Martin McClure

ESUG 2010

GemStone®

vmware®