Panini, the programming language

Sam M W

October 18, 2014

1 Introduction

This book introduces and explains how a grammar and dictionary is programmed into Panini.

2 Brief introduction to the Backus-Naur form

The Backus-Naur Form describes a reduction of a long phrase into shorter ones. In short, that is what is known as a context-free grammar. Here is a possible grammar that matches some simple sentences, described in Backus-Naur:

```
          sentence ::= <subject> <predication> <punct>
       predication ::= <verb-intransitive>
       predication ::= <verb-transitive> <object>
  verb-transitive ::= "love"
  verb-transitive ::= "see"
  verb-transitive ::= "dig"
  verb-transitive ::= "speak"
verb-intransitive ::= "weep"
verb-intransitive ::= "weep"
           subject ::= "I"
           subject ::= "you"
            object ::= "me"
            object ::= "you"
            object ::= "him"
            object ::= "Norwegian"
             punct ::= "."
             punct ::= "?"
             punct ::= "!"
```

Note a few things: anything on the left of ::= is a *symbol*; that is, a thing that must somehow be reduced down to simpler things. Anything to the right of ::= is the thing or combination of things that the symbol may be reduced to. These may include both terminals and non-terminals. A *terminal* is something

that does not need further decomposition. Also note that one non-terminal may decompose in several ways, and may be defined several times.

Here is how this may describe a sentence. We will look up "sentence" above, and see that it consists of a subject and a predication. Looking up "subject", we see that it is one of "I" and "you". Let's choose "I". The next thing in the decomposition of a sentence is the predication. We see that that it may either be a transitive verb plus an object, or an intransitive verb. Again, we'll choose one of the options. We will look at a transitive verb and an object. Look up verb-transitive, and choose one: "speak". The next thing in the decomposition of a predication is the object. One object is "Norwegian". The last thing that the sentence decomposed to is some punctuation mark. We'll choose ".". We have composed, or decomposed, the sentence "I speak Norwegian.", a perfectly valid English sentence.

The grammar given above will also generate these sentences:

- I weep.
- I love.
- I love him.
- You love Norwegian?
- You see Norwegian?
- You speak him.

As you can see, the grammar above produced some good English sentence, some dubious ones and some terrible ones. The reason for this is that Backus-Naur describes a context-free grammar. Context is what says, for example, that Norwegian is spoken, but not seen. And that we can see or love "him", but not speak "him".

3 Brief introduction to Panini, the programming language

Like Backus-Naur, Panini also describes grammar. But the difference is that Panini describes the contextual implications of a grammatical rule too. It also looks a bit different. Earlier, you saw a definition for "sentence" in Backus-Naur. Here is the same definition in Panini:

```
(df sentence
        (call subject)
        (call predication)
        (call punct))
```

Note that everything is wrapped in parentheses. The first word in the parentheses is *df*, which is short for "definition". The next word is the symbol that we

want to define, *sentence*. After that come any number of instructions enveloped by parentheses. In this example, there are instructions, each beginning with the word *call*. These are instructions to try to parse a constituent named by the second word inside the parentheses.

Let's define a few more symbols:

```
(df predication
        (call verb intransitive))
(df predication
        (call verb transitive)
        (call object))
```

Here, you can see that the *call* instruction is followed by sometimes one word, and sometimes two. In the previous paragraph I said that *call* tries to resolve a symbol named by the second word in the parentheses. The third (and there may be many more after this) are called flags, and serve to define more narrowly which symbols may be applied. The rules may contain the *flags* instruction, together with any token you want to flag the rule with. When it is called, it is checked to see that all the flags in the *call* instruction are also in the flag list in the rule. If you have another look at the Backus-Naur form above, you can see that we needed to define the verb "love" twice, because it occurs with an object or without one. Panini offers a more elegant solution, because you may flag a verb with both transitive and intransitive. Here's how:

```
(df verb
        (flags transitive intransitive)
        (lit love))
```

(the *lit* instruction matches the given word to the input, or appends it to the output).

In this way, Panini does exactly the same as the Backus-Naur notation. In the next section, you will see how Panini also provides more than context-free grammar.

4 Introduction to namespaces, scopes and variables

A variable is a small piece of information that is also addressed by name, and may optionally have some value attached to it. Variables are kept in a scope, which may also contain other scopes. A scope can also be addressed by name. Several scopes may exist inside each other. In this way, we can build up information and representations of arbitrary complexity.

Scopes live in namespaces. Namespaces have specific purposes. We will introduce two here. One namespace, called *rection*, is, by convention, used to build up grammatical information about a phrase. This may be the gender of a noun, or the number and person of a subject, for instance. This information

may be used again later on, to ensure that the correct suffix or article is used with the noun, or to ensure that a verb is conjugated correctly, for instance.

Another namespace is called *seme*. This is used to represent the meaning of a phrase. The variables' names and values should match from language to language, so that they can be translate from one to another or so that a program may take its input in any language that has been included in the Panini Project.

You will be told about the specifics of these and other namespaces as and when they are talked about in this documentation.

5 Scoping

Because natural and artificial languages are recursive, so that a predication may take several arguments, and those arguments may themselves take other arguments, the Panini Project must provide a way to scope the various information in sensible and flexible ways.