nowebdoc-gems: Code Extractor

User Manual — GEMS/Depto. de Computação, PUC-SP

Italo S. Vega

June, 2016

Contents

Introduction

Knuth (1984) introduced an appŕoach to programming called *literate programming*: logic explanation interspersed with code chuncks. Years later, Ramsey (1994) designed `noweb`, a simple and extensible tool for literate programming. However, it is difficult to install `noweb` in my academic environment due to the dependencies of several libraries and administrative restrictions. When I decided to search for `noweb` alternatives, I found several good ones, especially: `leo` (Ream 2001), `SimpleLit` (Rock 2013) and `noweb.py` (Aquino 2015). The first one is too sophisticated for my needs. The second one uses a different markup language for code annotation, but was very suggestive in ideas. The last one contributed with interesting indications for simplifying the document processing.

But I need more than literate programming only! I need a tool that integrates with `pandoc` (MacFarlane 2006) and/or the `listings` LaTeX package (Moses and

Hoffmann 2006): `nowebdoc-gems` was born! I choose to develop `nowebdoc-gems` with Netbeans and ANTLR4.

1 nowebdoc-gems — Tool

1.1 Example

The following file, called `description.md`, contains the description of a logic implemented by the corresponding Java code snippet:

```
My Great Tool
=============
Logic description of the program with code blocks using
'pandoc' markers...

~~~
<<Foo.java>>=
class Foo {
    public static void main( String[] args ) {
        System.out.println( "Code extracted by 'nowebdoc-gems'" );
    }
}
~~~

Continuation...
```

Just run the command `nowebdoc-gems -R Foo.java description.md > Foo.java` to extract the snippet `Foo.java` from this description.

1.2 Activation Model

The ultimate goal of this project is to produce a Java application that processes an input literary program with code snippets (noweb/Pandoc format) and extract your code as output:

```
nowebdoc-gems -R NowebDocGems.java  noweb-pandoc.md > NowebDocGems.java
```

In this case, the computational effect produced by the application execution is as follows. From `noweb-pandoc.md` will extract the code called`NowebDocGems.java`.

Command line From the command line, the tool offers the following options:

```
nowebdoc-gems [--config <CONFIGURATION> | --version | --help] -R <FRAG> <SPECI
```

The `--config` option receives a configuration file as activation argument. The file should contain a pair of code delimiters used in the specification (generally is used `nowebdoc-gems.cfg` as the configuration file) with the following contents:

```
# Pandoc code markers:
code.begin=^\~\~\~.*
code.end=^\~\~\~
```

Lines starting with *sharp* are ignored by `--config` command. The last couple of valid delimiters define the corresponding processing values. No consistency checking occurs in processing this file. Many execution errors can be caused if the delimiters are invalid.

1.3 Primary Behavior

The behavioral model tool `nowebdoc-gems` was elaborated in accordance with the *TLA+* ideas proposed by Lamport (2002) and represented in UML (Object Management Group 2015). An `NowebDocGemSnippets` object determines the characteristic operation of this tool (see the UML state machine diagram showed in fig. 1). In a normal state (*Reading Description*), it is in a condition that indicates that it is not within a block of code. It also has an explicit representation of snippet name inexistent: $inCode = \textbf{false} \wedge snippetName = \varnothing$. When it finds a description line that matches the code start pattern, it passes to the code reading state $inCode = \textbf{true}$. When the line finally matches the code snippet identification pattern, it enters the state *Reading Snippet Name*. The condition changes to $snippetName = name \wedge snippetMap[name] = \varnothing$. The lines that follow are interpreted as lines of a code snippet, unless one of them matches the output code block pattern. Each line of code is added to the snippet list mapped by the *snippetName*.

2 Instalation

The next two scripts are found in the root path of this project. The tool activation command follows:

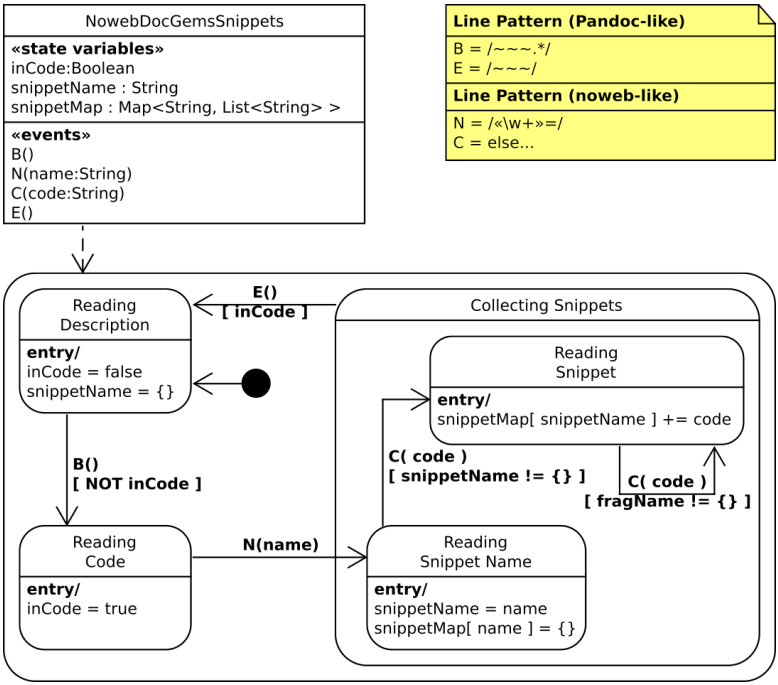

Figure 1: Tool primary behavior

```
<<nowebdoc-gems>>=
#! /bin/bash
java -jar /usr/local/lib/nowebdoc-gems/nowebdoc-gems.jar $*
```

This script installs the tool in `/usr/local/lib` and copies the activation command in `/usr/local/bin`:

```
<<install-nowebdoc-gems>>=
#! /bin/bash
rm -fR /usr/local/lib/nowebdoc-gems
mkdir /usr/local/lib/nowebdoc-gems
cp ../nowebdoc-gems/dist/nowebdoc-gems.jar /usr/local/lib/nowebdoc-gems

rm -fR /usr/local/lib/nowebdoc-gems/lib
mkdir /usr/local/lib/nowebdoc-gems/lib
cp ../nowebdoc-gems/dist/lib/* /usr/local/lib/nowebdoc-gems/lib

cp ./nowebdoc-gems /usr/local/bin
chmod u+x /usr/local/bin/nowebdoc-gems
```

The final step copies a configuration file in the home directory:

```
rm -fR ~/cfg/nowebdoc-gems
mkdir ~/cfg/nowebdoc-gems
cp ./nowebdoc-gems.cfg ~/cfg/nowebdoc-gems
```

Bibliography

Aquino, Jonathan. 2015. “Noweb.py.” https://github.com/JonathanAquino/noweb.py.

Knuth, Donald E. 1984. “Literate Programming.” *The Computer Journal* 27 (2). Oxford University Press: 97–111.

Lamport, Leslie. 2002. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison.

MacFarlane, John. 2006. “Pandoc a Universal Document Converter.” http://pandoc.org/.

Moses, Brooks, and Jobst Hoffmann. 2006. “Listings — Typeset Source Code

Listings Using LaTeX." http://ctan.org/pkg/listings.

Object Management Group. 2015. *Unified Modeling Language*. V. 2.5. http://www.omg.org/spec/UML/2.5.

Ramsey, Norman. 1994. "Literate Programming Simplified." *IEEE Software* 11 (September): 97–105.

Ream, Edward K. 2001. "Leo's Home Page." http://leoeditor.com/.

Rock, Andrew. 2013. "The SimpleLit System for Literate Programming." Technical Report. Griffith University.