PTG(beta) User Manual

Egill Búi Einarsson

March 18, 2013

abstract

PTG is a command-line program that generates files containing parsing tables and state machines for a given grammar. Generated tables are in either LaTeX or HTML format which eases automated inserting of generated elements. Likewise the state machines can be inserted directly when TIKZ format is used. Alternativily state machines can use Graphviz's digraph automaton format for use with Graphviz DOT.

1 Copyleft

Table of Contents

1 Copyleft **1**

2 Introduction **1**

3 Setup **1**
3.1 Compiling the Source 2

4 Missing Features **2**

5 Command Line Arguments **2**

6 Preparing the Input Grammar File **3**

7 Implementing Generated Files **3**
7.1 LaTeX Tables 3
7.2 HTML Tables 3
7.3 Graphviz Statemachines 3

8 I found a bug, what should I do? **3**

9 An Example **3**

2 Introduction

PTG is run from a command-line terminal and generates parsing tables and state machines for LL(1), LR(0) and LR(1) grammars. A grammar's FIRST, FOLLOW, LL(1), SLR(1) and LALR(1) parsing tables can be generated in either LaTeX or HTML format along with LR(0) and LR(1) state machines in Graphviz's digraph automaton format for use with DOT. The idea is when creating a document for a particular grammar that features some or all of these tables, instead of copying into the master file the generated files are dynamically linked. The result is that the document can be updated [...]

No guarantee is made that this software will run smoothly, without flaws, without destroying other files, without corrupting data, without crashing your system, at all... etc. Please show the appropiate caution when running any software from the internet. Furthermore no guarantee is made on the correctness of the output. The second to last section of this manual details what to do if an error is encountered.

3 Setup

Setup is simple as long as a JRE(Java Runtime Environment) has been setup. Go to the https://github.com/EgillEinarss/PTG on Github.com and download PTG.jar. Place this file in the current command line directory and run

```
java -jar PTG [...]
```

along with any relevant arguments(detailed in the next section).

3.1 Compiling the Source

Required programs are git and make. Use a command line. If needed, go to your projects directory and get the project with the command

```
git clone https://github.com/EgillEinarss/PTG.git
```

and enter the directory with

```
cd PTG
```

Alternatively go to the PTG repository on Github.com and download what you want or require.

Now you have the developement environment setup used to create PTG. Now simply run the compilation process with the command

```
make
```

Simple.

Feel free to modify the source code in the directory src, also any Java source files added into the directory src will be compiled and added to the PTG jar file created by running the `makefile`.

4 Missing Features

Some features are still waiting to be implemented or are improperly implented. This manual might even describe these features as implemented. A partial list of features or components in limbo:

- Either LR(0) or LALR(1) machines are incorrect.
- The LR option does nothing.
- Tikz is displaying terrible machines.
- This manual is far from resembling a useful manual.
- Tables for machines with memory, print an entry for each item in memory.

5 Command Line Arguments

The syntax for command line execution is

```
   java PTG Grammar [-Start symbol] [-End symbol] [-Empty symbol]        ([-tabl
or
   java PTG grammar [-start symbol] [-end symbol] [-empty symbol] -all [out]
```

grammar is the file that contains the grammar that will be parsed, it is a required argument and needs to be the first one supplied.
start is an optional parameter which sets the start variable of the grammar to the supplied symbol. The default is the first variable listed in the grammar.
end is an optional parameter which sets the end of input variable of the grammar to the supplied symbol. The default is $.
empty is an optional parameter which sets the empty string of the grammar to the supplied symbol. Default is <e>.
symbol a string that should avoid the symbol # along with all whitespace.
table should be replaced with one of

```
   first    follow    LL1    SLR1    LALR1    LR1
```

6 Preparing the Input Grammar File

7 Implementing Generated Files

7.1 LaTeX Tables

There are two simple ways to use generated LaTeX tables, either by copy pasting or using the input command. PTG generates a tabular environment as opposed to an actual table environment, this is because a table (or other container, for example a figure) has commands that relate to placement of the environment in the document. The input command will allow the use of generated tex files directly. This allows a user to update a grammar file, run PTG for that grammar and then recompile the document. A table or figure environment can be used to contain the input command. Below is an example of how to insert a file named exampleTable.tex into a figure:

```
\begin{table}
\centering
%\input{exampleTable.tex}
\caption{A caption for the table}
\label{table:TableLabel}
\end{table}
```

Both the caption and label commands are optional and the arguments supplied for them are nonsensical, \centering is also optional.

7.2 HTML Tables

7.3 Graphviz Statemachines

Graphviz statemachine files have the extension .gz and should be rendered using Graphviz DOT. An example command to render exampleSM.gz as a png image file named exampleSM.png would be:

```
   .\dot -Tpng exampleSM.gz -o exampleSM.png
```

Check the DOT documentation for more details. http://www.graphviz.org/pdf/dotgu

8 I found a bug, what should I do?

Sit tight, don't worry, we'll get through this. Send a few line describing what you were doing or intending to do, the command line inputs you used and the input grammar to [...]. // If you can't wait, then feel free to modify the source code in hopes of fixing the bug.

9 An Example

Table 1: FIRST, FOLLOW and LL(1) parsing of the example