PTG(beta) User Manual

Egill Búi Einarsson

March 19, 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

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 PTG repository 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
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 src will be compiled and added to the PTG jar file created by running make.

4 Missing Features

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

- Either LR(1) 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.
- No table outputed for tiny state machines.

- -ALL is missing items.
- The help text is outdated.

5 Command Line Arguments

The syntax for command line execution is

```
java -jar PTG Grammar Options Output
```

where

Grammar The file containing the grammar to be parsed. Required and must be the first argument.

Options A few options that define the grammar. Any number of them may be used and must be followed by a String, see Settings for String.

-START Sets the grammar's start variable to String. Default is the first variable listed in Grammar.

-END Sets the grammar's end of input token to String. Default is "$".

-EMPTY Sets the grammar's empty string token as String. Default is "<e>".

Output This is the output request. All of them can be followed by Settings, if a String is defined then it will be the output file name, default is the filename of the grammar.

-ALL Performs all of the following outputs.

-FIRST Creates a First table.

-FOLLOW Creates a Follow table.

-LL1 Creates a LL(1) table.

-SLR1 Creates a SLR(1) table.

-LR1 Creates a LR(1) table.

-LALR1 Creates a LALR(1) table.

-LR0M Creates a LR(0) state machine.

-LR1M Creates a LR(1) state machine.

-LALRM Creates a LALR(0) state machine.

Settings Any and all settings are optional, some are not applicable but cause no error.

HTML The table being defined will only have HTML output.

LATEX The table being defined will only have LaTeXoutput.

GZ The state machine being defined will be in GraphViz's DOT format.

TIKZ The state machine being defined will be in Tikz format.

TINY The state machine being defined will contain only a label, will produce a table for more information.

SMALL The state machine being defined will have a decreased description if possible.

LARGE The state machine being defined will have a full description.

LR The state machine being defined will go from left to right rather than top to bottom.

String Any string of characters that is not one of the above defined keywords, pretty much anything is accepted but might cause catastrophic failure.

Note that the keywords are in capital letters for clarity but in face PTG is case insensitive in regards to keywords. Here are a few example calls:

```
java -jar PTG example.gra -all
java -jar PTG example.gra -start Var2 -end EoF -empty e -lll html
```

```
java -jar PTG Grammar [-Start symbol] [-End symbol] [-Empty symbol]
  ( [-table [table_type] [out]] | [-machine [] [out]] )+
```

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

The Input Grammar File is a text file that contains the grammar to parse. There are no intended constraints on the filename or it's extension, the filename(that is without the dot and extension) will be used as a default prefix for any output filenames unless a new name is given in the input arguments. In the file each line is one rule, each rule contains a left-hand side, a seperator and a right-hand side in that order. The left-hand side and the seperator are one symbol each whereas the right-hand side is a string of one or more symbols. A symbol is a whitespace terminated string of characters.

So a rule is a string of three or more symbols where the first is the left-hand side and the second is a seperator. PTG will interpret any left-hand side symbols as being a variable in the grammar and the left-hand side of the first rule is the default start variable. Any symbols in a right-hand side that is not a variable is a terminal symbol. There is one special symbol which is the empty string symbol, in a rule that uses the empty symbol the right-hand side should consist of only the empty string symbol. The default empty string symbol is <e> but a new one can be defined as an input argument. To learn more about rules see this Wikipedia article.

Lastly a grammar is terminated by an empty line. This allows a comment to follow after the grammar for whatever reason. An example of a grammar is shown below, this example is used in the section An Example below and the

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 ebe10@HI.is and hopefully I can fix it or explain it. // If you can't wait, then feel free to modify the source code in hopes of fixing the bug.// Also feel free to send me a line with comments or complaints regarding PTG.

9 An Example

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