PTG(beta) User Manual

Egill Búi Einarsson

April 1, 2013

abstract

PTG is a command-line program that generates files containing parsing tables and state machines for a formal language defined by an input 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 the TIKZ format is used. Alternativily state machines can use Graphviz's digraph automaton format which is intended 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 context insensitive formal grammars. A grammar's FIRST, FOLLOW, LL(1), SLR(1), LALR(1) and LR(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 or TIKZ format for use directly in LaTeX. The idea is when creating a document studying formal languages, that features some or all of these tables and machines, instead of writting theentites directly or copying a generated file into the master file PTG generated files can be dynamically linked in the master file. The result is that if the formal languages change then only the PTG grammar file needs to be updated, PTG run on the new grammar and Graphviz Dot on any digraph automaton format. The later two steps can of curse be automated with a make or batch file.

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.jar ...
```

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 that was 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:

- LALR(1) machines are incorrect(besides not existing), resulting in incorrect LALR(1) tables.
- Tikz is (often) displaying terrible machines. (Not caused by Tikz!)

5 Command Line Arguments

The syntax for command line execution is:

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

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, the filename of the grammar is used as a default.

-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 fact PTG is case insensitive in regards to keywords. Here are a few example calls:

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

```
java -jar PTG.jar Grammar [-Start symbol] [-End symbol] [-Empty symbol]
```

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 supplied by 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 consisting 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 Example section below. Here the default start variable is S.

```
S -> S ( S )
S -> <e>


S -> ( S ) S
S -> <e>

A comment starts here.
This is an example grammar for PTG and is used to generate
the example tables and state machines in the manual.

E -> T Em
Em -> v T Em
Em -> <e>
T -> ekki T
T -> ^
T -> ( E )
T -> 0
T -> 1
```

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

For now the recommendation is copy-pasting the contents of the generated file into the master file or just hyperlinking the generated file as an individual page from the master file. There should be a work-around with Javascript and it is quite simple to fix the problem with PHP.

7.3 TIKZ Statemachines

```
\usepackage{tikz}
\usetikzlibrary{arrows,automata}
\usetikzlibrary{shapes.multipart}
\usetikzlibrary{shapes.misc}
```

Add these commands to the document header, between the documentclass and document directive, if missing. Now a generated Tikz statemachine contained in the file exampleMachine.tex can be added with:

```
\input{exampleMachine.tex}
```

7.4 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/dotguide.pdf

8 I found a bug, what should I do?

Go to the PTG repository and check if it is a known issue, if not added it there. Remember to supply all the necessary information, that is what you intended to do, what PTG did, the commandline argument you used and lastly the input grammar file.
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

Here is an example to demonstrate the use of PTG. The text for the example is in the example.gra in the examples directory of the PTG repository and is as follows:

```
S -> S ( S )
S -> <e>


S -> ( S ) S
S -> <e>

A comment starts here.
This is an example grammar for PTG and is used to generate
the example tables and state machines in the manual.

E -> T Em
Em -> v T Em
Em -> <e>
T -> ekki T
T -> ^
T -> ( E )
T -> 0
T -> 1
```

The FIRST, FOLLOW and LL(1) parse tables can be generated by the command:

```
java -jar PTG.jar examples/example.gra -first -follow -LL1
```

X	FIRST(X)
S	ϵ (

X	FOLLOW(X)
S	( ) $

	(	)	$
S	ϵ S (S)	ϵ S (S)	ϵ S (S)

Table 1: FIRST, FOLLOW and LL(1) parsing of example/example.gra

Now let's generate LR(0) state machine in Graphviz format and create a png image file with DOT.

```
java -jar PTG.jar examples/example.gra -LR0M large gz
dot -Tpng examples/exampleLR0M.gz -o examples/exampleLR0M.png
```

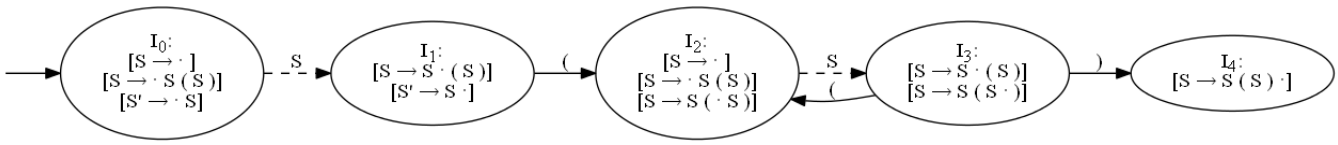


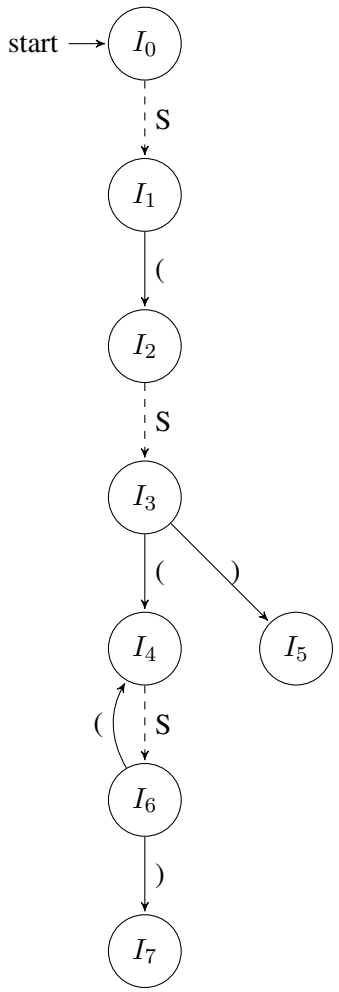
Figure 1: LR(0) of examples/example.gra

State	(	)	$	S
I_0	reduce S $\rightarrow \epsilon$	reduce S $\rightarrow \epsilon$	reduce S $\rightarrow \epsilon$	I_1
I_1	shift I_2		reduce S' $\rightarrow$ S	
I_2	reduce S $\rightarrow \epsilon$	reduce S $\rightarrow \epsilon$	reduce S $\rightarrow \epsilon$	I_3
I_3	shift I_2	shift I_4		
I_4	reduce S $\rightarrow$ S (S)	reduce S $\rightarrow$ S (S)	reduce S $\rightarrow$ S (S)	

Table 2: SLR(1) of examples/example.gra

Lastly let's generate the tiny LR(1) state machine along with the labels table:

```
java -jar PTG.jar examples/example.gra -LR1M tiny tikz
```



State	Current Rule Set
I_0	S $\rightarrow \cdot \epsilon$, $ S $\rightarrow \cdot \epsilon$, (S $\rightarrow \cdot$ S (S), $ S $\rightarrow \cdot$ S (S), (S' $\rightarrow \cdot$ S, $
I_1	S $\rightarrow$ S $\cdot$ (S), $ S $\rightarrow$ S $\cdot$ (S), (S' $\rightarrow$ S $\cdot$, $
I_2	S $\rightarrow \cdot \epsilon$, (S $\rightarrow \cdot \epsilon$,) S $\rightarrow \cdot$ S (S), (S $\rightarrow \cdot$ S (S),) S $\rightarrow$ S ($\cdot$ S), $ S $\rightarrow$ S ($\cdot$ S), (
I_3	S $\rightarrow$ S $\cdot$ (S), (S $\rightarrow$ S $\cdot$ (S),) S $\rightarrow$ S (S $\cdot$), $ S $\rightarrow$ S (S $\cdot$), (
I_4	S $\rightarrow \cdot \epsilon$, (S $\rightarrow \cdot \epsilon$,) S $\rightarrow \cdot$ S (S), (S $\rightarrow \cdot$ S (S),) S $\rightarrow$ S ($\cdot$ S), (S $\rightarrow$ S ($\cdot$ S),)
I_5	S $\rightarrow$ S (S) $\cdot$, $ S $\rightarrow$ S (S) $\cdot$, (
I_6	S $\rightarrow$ S $\cdot$ (S), (S $\rightarrow$ S $\cdot$ (S),) S $\rightarrow$ S (S $\cdot$), (S $\rightarrow$ S (S $\cdot$),)
I_7	S $\rightarrow$ S (S) $\cdot$, (S $\rightarrow$ S (S) $\cdot$,)

Figure 2: LR(0) statemachine for examples/example.gra with tiny states along with labels table.

To end this example, the full LaTeX source file can be read in doc/UserManual.tex in the PTG repository.