PTG User Manual

Egill Búi Einarsson

June 3, 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 into the LaTeX source files when the TIKZ format is used. Alternatively state machines can use Graphviz's digraph automaton format which is intended for use with Graphviz DOT.

Copyleft

Contents

1 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 writing the entites 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 and then the tables and state machines can be represented by using PTG, Graphviz, etc.. This process can of course be automated with a make or batch file with the exception of updating the grammar file.

2 Setup

Setup is simple as long as a JRE (Java Runtime Environment) has been setup. Go to the PTG repository[1] 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).

2.1 Compiling the Source

Required programs are git[2] and make[3]. 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[4] and download what you want or require.

Now the developement environment used to create PTG has been set up. To compile the source and create the jarfile used to run PTG, type in the command:

```
make
```

PTG is an open source program and the source code can be found in the dirctory `src`. Feel free to modify the source code. Any new Java source files can be added into the `src` directory without causing problems. They will be compiled and added to the PTG jar file when make is run without needing to modify the makefile.

3 Command Line Arguments

The syntax for command line execution is:

```
java -jar PTG.jar <Grammar> <Options> <Table> <State machine> <All>
```

<Grammar> defines what file is to be used as the PTG grammar file. <Options> define some of the grammar's parameters. <Table>, <State machine> and <All> define output options for PTG. Each of the output options will create one or more files using the base filename, followed by a suffix and lastly

[1] https://github.com/EgillEinarss/PTG
[2] http://git-scm.com/
[3] http://www.gnu.org/software/make/
[4] https://github.com/

an appropriate file extension. Which suffix is added reflects what type of parsing table or state machine was being generated. Each of these parameters is detailed below in their own subsection. Note that all keywords are given in capital letters, although PTG is case-insensitive.

3.1 <Grammar> Parameter

This is a required parameter and must be the first one supplied. It will be used as the filename of the PTG grammar file to be parsed. Furthermore all characters up to the first dot will be used as the default base filename for any output files generated by PTG.

3.2 <Options> Parameters

These options help define the grammar. Any number of them may be used but each must be followed by a string of non-whitespace ASCII characters. Below, this string is called <token>.

- **START <token>** Sets the grammar's *start variable* to <token>. The first variable listed in the supplied grammar is used as a default parameter.
- **END <token>** Sets the grammar's *end of input* to <token>. The default is "$".
- **EMPTY <token>** Sets the grammar's *empty string* to <token>. The default is "<e>".

Setting more than one of these parameters with the same <token> will result in untested and undefined behaviour.

3.3 <Table> Parameters

Any number of <Table> parameters can be supplied in an argument. Each parameter defines one or two new files that PTG should create and determines which of the parsing tables it should contain. The syntax for each is:

```
<Table type> <Table option> <Filename>
```

<Table type> Argument

This is a required argument of each <Table> parameter and the possible values are:

- **FIRST** Creates the First table. `FIRST` will be concatenated to the base filename.
- **FOLLOW** Creates the Follow table. `FOLLOW` will be concatenated to the base filename.
- **LL1** Creates the LL(1) table. `LL1` will be concatenated to the base filename.
- **SLR1** Creates the SLR(1) table. `SLR1` will be concatenated to the base filename.
- **LR1** Creates the LR(1) table. `LR1` will be concatenated to the base filename.
- **LALR1** Creates the LALR(1) table. `LALR` will be concatenated to the base filename.

<Table option> Argument

This is an optional argument which limits the number of output files to one.

HTML The parsing table generated will be represented in HTML and have the file extension `.html`.

LATEX The parsing table generated will be a LaTeX tabular environment and have the file extension `.tex`.

If this optional argument is not present then PTG will interpret it as though the user wants both formats to be generated.

<Filename> Argument

The last optional argument is to override the default base filename of any generated file.

3.4 <State machine> parameters

Any number of <State machine> parameters can be supplied in an argument. Each parameter defines one or two new files that PTG should create and determines which of the state machines it should contain. The syntax for each is:

```
<SM type> <SM option> <Size> <Orientation> <Filename>
```

<SM type> Argument

This is a required argument of each <State machine> parameter and the possible values are:

−**LR0M** Creates the LR(0) state machine. `LR0M` will be concatenated to the base filename.

−**LR1M** Creates the LR(1) state machine. `LR1M` will be concatenated to the base filename.

−**LALRM** Creates the LALR(0) state machine. `LALRM` will be concatenated to the base filename.

<SM options> Argument

This is an optional argument which limits the number of output files to one.

GZ The state machine output will adhere to Graphviz's digraph automaton format and have the file extension `.gz`.

TIKZ The state machine ouput will adhere to the TIKZ automata format and should be fully compatible with LaTeX. It will have the file extension `.tex`.

If this optional argument is not present then PTG will interpret it as though the user wants both formats to be generated.

<Size> Argument

The optional <Size> argument defines the amount of text contained in each state of the state machine, although it will always contain a label (an identifier). The default state size is TINY.

TINY Each state will contain only a label.

LARGE Besides a label, each state will show which rules can be used to continue parsing an input string.

SMALL The same as LARGE state size but redundant rules are omitted.[5]

When the state size is TINY, two extra files will be generated containing tables with the extra information that was omitted in comparison to the state size being LARGE. One file will use HTML syntax and add `label.html` to the base filename whereas the other will contain a LaTeX tabular environment and add `label.tex` to the base filename.

<Orientation> Argument

This is another optional argument that changes the orientation of the generated state machine.

LR The state machine will grow from left to right.

If the option is unused then the state machine will grow from top to bottom.

<Filename> Argument

The last optional argument is to override the default base filename of any generated file.

3.5 <All> parameter

This will create all of the possible output options of PTG. The syntax is:

```
-ALL <Table option> <SM option> <Size> <Orientation> <Filename>
```

All of these options behave as detailed in the <Table> Parameters and <State machine> Parameters subsections.

3.6 Examples of Command Line Arguments

Below are three examples:

```
java -jar PTG.jar example.gra -all
java -jar PTG.jar example.gra -all latex tikz large out
java -jar PTG.jar filename -start Var2 -end EoF -empty e -lll html
```

The first generates all possible parsing tables in both LaTeX and HTML format and state machines in both Graphviz DOT and TIKZ format with tiny state sizes using `example` as the base filename from the PTG grammar file `example.gra`. The second generates all possible parsing tables in LaTeX format and state machines in TIKZ format with large state sizes using `out` as the base filename. It uses the same PTG grammar file as the first. The third and last generates the LL(1) parsing table in HTML format for the the PTG grammar defined in the file `filename`. For this grammar the *start variable* is `Var2`, the *end of input* is `EOF` and the *empty string* is denoted by `e`. The output file is named `filenameLL1.html`.

4 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 base name for any output filenames unless a new name is supplied by command line 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.

To reiterate, 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 top rule is the default *start variable*. Any symbols in a right-hand side that is not a *variable* is a *terminal symbol*.

Care should be taken with the *empty string* symbol. All rules that uses the *empty string* symbol should have a variable, followed by a seperator and then finally the *empty string* symbol. The *empty string* symbol cannot be a *variable*. The default *empty string* symbol is <e> but a new one can be defined in the command line arguments. To learn more about rules, see this Wikipedia article[6].

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 PTG grammar is shown below, this is the same example used in the Example section. Here the default *start variable* is S.

```
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.
```

5 Using Generated Files

After using PTG to generate files, a new problem arises regarding how to use them. In this section, possible ways of displaying the generated files are discussed.

5.1 LaTeX Tables

There are two simple ways to use generated LaTeX tables, either by pasting them into the master file or by using the input command. PTG generates only a tabular environment instead of an actual table environment, this is because a table (or any other container, for example a figure) has commands that relate to placement of the environment in the document.[7] The input command will allow the use of generated LaTeX 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 table environment:

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

Both the caption and label commands are optional and the arguments supplied for them are nonsense. The centering command is also optional.

[6] http://en.wikipedia.org/wiki/Formal_grammar\#Context-free_grammars

[7] The tabular environment can be used without a container.

5.2 HTML Tables

For now the recommendation is 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.[8] There should be a work-around with Javascript and it is quite simple to fix the problem with PHP.

5.3 TIKZ State Machines

To use TIKZ state machines some commands are required in the document's preamble, the preamble consists of everything before the \begin{document} command. These commands are:

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

Now a generated Tikz statemachine contained in the file exampleMachine.tex can be added with:

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

Check the TIKZ website[9] for more details on how to use TIKZ.

5.4 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[10] for more details.

6 I found a bug, what should I do?

Go to the PTG repository and check if it is a known issue, if not add it there. Remember to supply all the necessary information to recreate and fix the bug. This includes what you intended to do, what PTG did, the command line arguments 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 through the repository with comments or complaints regarding PTG.

[8] The page will be lacking many of the frills associated with the HTML standard, for example a header.
[9] http://www.texample.net/tikz/
[10] http://www.graphviz.org/pdf/dotguide.pdf

7 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>

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.
```

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 tables for 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
```

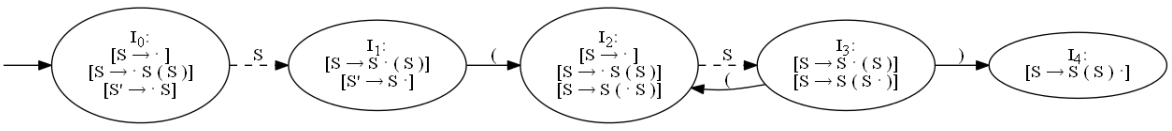


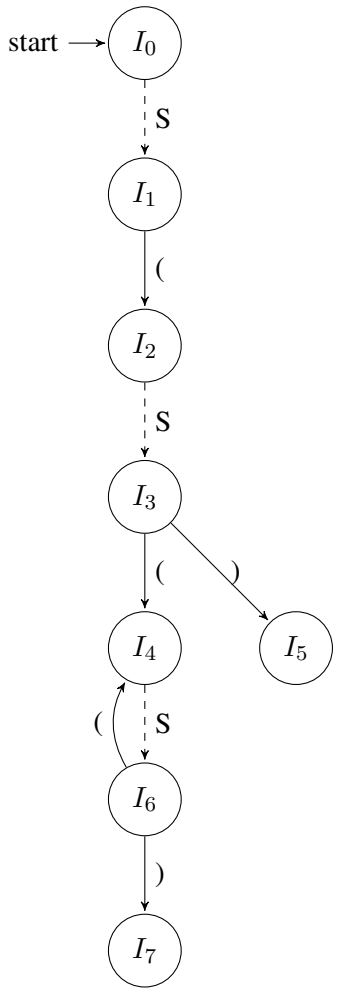
Figure 1: LR(0) statemachine for examples/example.gra

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) parsing table for examples/example.gra

Lastly let's generate the LR(1) state machine with TINY states along with a table of the omitted information for each state:

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



State	Current Rule Set
I_0	S → · ϵ, $ S → · ϵ, (S → · S (S), $ S → · S (S), (S' → · S, $
I_1	S → S · (S), $ S → S · (S), (S' → S · , $
I_2	S → · ϵ, (S → · ϵ,) S → · S (S), (S → · S (S),) S → S (· S), $ S → S (· S), (
I_3	S → S · (S), (S → S · (S),) S → S (S ·), $ S → S (S ·), (
I_4	S → · ϵ, (S → · ϵ,) S → · S (S), (S → · S (S),) S → S (· S), (S → S (· S),)
I_5	S → S (S) · , $ S → S (S) · , (
I_6	S → S · (S), (S → S · (S),) S → S (S ·), (S → S (S ·),)
I_7	S → S (S) · , (S → S (S) · ,)

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

In parting, I wish to point out that the full LaTeX source file for this user manual can be found in `doc/UserManual.tex` in the PTG repository.