Appendix B

Experimental Tool User Manual

In order to make practical tests and comparisons with other LR methods, an experimental tool has been implemented, which generates parsing tables for discriminating-reverse LR(1), and some funny variants as LR(0) or full-stack reverse recognizers. The program also includes generation of direct canonical LR(1), LR(0), SLR(1), and LALR(1) (uncompressed-)table-driven parsers, and even an automatic sentence generator, which can be used to obtain easily nontrivial random sentences for test purposes.

Efficiency was not amongst the objectives for these implementations, i.e., this is not a production-quality tool, since the main interest is to have a code that could be easily modified and clear to implement directly the algorithms and minimize the possibility of programming errors. It has been found convenient also for didactic purposes.

This Appendix presents the user manual for the latest version of the tool. First, the general layout is presented, showing how the different modules interact, the file organization, the grammar and sentence formats, and the command interface for the different modules. Finally, an example session is reproduced.

B.1 Using the tool

The tool is invoked at the command line prompt with the program name `aut`, followed by a functioning mode, asking for one of the following things:

`aut gr` K To *g*enerate the discriminating-*r*everse parsing tables

`aut gd` K To *g*enerate the usual *d*irect parsing tables

K represents the parser lookahead length, which may be specified as `0` or `1` (default).

`aut s` LENGTH To generate one or more random sentences from the grammar. If LENGTH is specified, sentence(s) with the nearest possible length are generated; otherwise random length sentences are produced.

`aut pr` To *p*arse an input sequence with the discriminating-*r*everse parsing tables

`aut pd` To *p*arse an input sequence with the *d*irect parsing tables

First, one of the parser generators is invoked. Then, the sentence generator can optionally be invoked to produce a random sentence, or an input text file can be manually constructed by using any text editor. Finally, the corresponding parser is invoked with the input file.

B.1.1 Files used by the tool

The tool uses the following four files —the default names in the current directory are shown, although at invocation time other (path) name(s) can be given:

`g.aut` ASCII text file containing the specification grammar in Backus-Naur Form (BNF). This is the input to the generators (see the section below for the format of this file).

`v.aut, p.aut` Vocabulary and parsing-table file, respectively, both in internal format. They are produced by the parser generators and read by the parsers.

`s.aut` ASCII text file containing a sequence to be parsed. It can be manually written or automatically generated by `aut s` (see below for the format of this file).

The use of default names allows to comfortably work without caring about the internal files of the tool. That is, the user can write with an editor the `g.aut` text file containing the grammar —two programs, `yacc2aut` and `aut2yacc`, are available to convert `yacc` grammars to the tool format and vice versa— and successively invoke the different modes, without any new command or edition, to generate parsers or sentences and verify parses.

As previously stated, the user can specify, when appropriate, the precise file (path) names, with the following command line options:

`-gf` FILEPATHNAME *G*rammar *f*ile.

`-vf` FILEPATHNAME *V*ocabulaty *f*ile.

`-pf` FILEPATHNAME *P*arsing-table *f*ile.

`-sf` FILEPATHNAME Input *f*ile to the parser containing a *s*equence of terminals symbols.

The grammar file

The grammar file is the input to the parser or sentence generators. It specifies the structure of legal sentences of the language. The file format is an ASCII text file containing the grammar in BNF notation. More specifically, a grammar is a sequence of rules, where each rule has the form:

```
nonterminal = rightpart { | rightpart } .
```

That is, a nonterminal name, an equal sign, and one or more rightparts (sequences of symbols) separated by vertical bars, all ended with a dot. One rightpart may be empty. Terminal symbols should appear between quotes (") containing the strings as they will literally appear in sentences, formed by any sequence of characters (quotes are represented in the sequence by a pair of quotes). Nonterminal symbols begin with any alphabetical upper or lower case characters or underscore (_), and may also contain numerical characters. The start symbol is the first to appear in the file.

Grammars should be well-formed (usually called "reduced" in the literature), otherwise an error is signalled when they are processed. The following tests are made:

- No rightpart can be repeated for the same nonterminal.
- Every terminal symbol should be generated from the start symbol.
- Every nonterminal symbol should derive sequences consisting only of terminals or the empty sequence.

Note also that recursive rules as "`A = A`" are useless and make the grammar ambiguous.

The input files to parse

After the parser has been successfully generated, we might want to test it with an input file. The format of this file to parse is just an ASCII text file containing the terminal symbols exactly as they appear in the grammar (without quotes), separated by spaces or newlines. This is precisely the format produced by the automatic sentence generator.

This simple scheme allows an easy construction of sentences, with no other purpose than to test the parsers. The parser is intended to be integrated within the Cigale system, then standard lexical analyzer features will be available.

B.1.2 Other command line options

This section describes other command line options. Some of them are significant only in some functioning mode, some in several modes. Although this

should be evident from the explanations, you can look at the beginning of the example session below to see which case is appropriate.

`-`NUMBER Specify the number of sentences to display (or redirect to a file with `>`). Only the last one is saved to the sentence file. By default NUMBER is 1.

`-c` In case of reverse *c*onflicts for non-LR(K) grammars, interactively ask which action is preferred. Note that conflicting situations are detected in pairs, and so the user is asked.

`+c` By default the generator stops state construction from situations in conflict. With this option the curious user makes the generator to *c*ontinue building states beyond conflicts until bottom-of-stack states.

`+e` A technique using a heuristic estimate of typical l*e*ngths of nonterminal phrases is used by default to generate sentences, with the intention to produce human-like sentences. With this option, that computation is disabled, and sentences are generated purely at random, but then results are typically very poor in comparison with the default.

`+f` Disable *f*ull rightparts computation, that is, do not consider in reverse automata whether or not the lookahead part is covered by the portion of the rightpart to the left of the marker. Without special options activated, this option asks the generator to produce only the discriminating section of the automaton, not the additional states that would be used to verify the presence of the handle on top of the stack.

`-g` DEBUGNUMBER Activate the debug flag DEBUGNUMBER. For developing purposes only.

`-i` With this option the reverse parser is generated from an *i*nitial state interactively specified by the user. State situations need to be specified according to internal codes, which can be displayed using option `-dgv` (see below).

`+l` Produce a (direct) LALR(1) parser instead of an full LR(1) one, by merging states with the same core (non-lookahead) part. Apart from this, no further compression is made, contrary to what is usual in other LALR(1) parser generators, e.g., `yacc`.

`-l` When producing a direct LR(1) parser for a non-LALR(1) grammar, signal states that after merger with previous LR(1) states would lead to LALR(1) conflicts.

`+n` SEED By default, every time the sentence generator is invoked, a different random sentence is produced. Sometimes, repeatability is interesting,

and then this option can be used to always produce the same random sentence(s). A SEED value can be optionally specified to obtain more (repeatable) random sentences. The default value of SEED is 0.

`-r a` A full reverse recognizer for the whole stack is produced, which can be compared with the usual discriminating reverse automaton. Either LR(0) or LR(1) algorithm can be used. If suboption "`a`" is not specified, situation actions are not considered during automaton construction, which yields a smaller full-stack recognizer, although useless for action discrimination.

`+s` Produce a (direct) SLR(1) parser instead of an full LR(1) one, by augmenting items within the LR(0) automaton states with lookaheads in the Follow sets. The resulting states correspond to LALR(1) ones, but sometimes the SLR(1) parsing tables have more entries.

`+t` CHAR Produce a file named `paut.tex` containing the parsing table in text form, which can be displayed, browsed with an editor, or printed. If CHAR is specified (usually a dot), then it is used instead of a blank space for empty table entries. See also option `-x`.

`+v` The program produces by default some friendly output. This can be avoided with this option, which may be of interest when the tool is executed from scripts.

`-x` To help in the preparation of documents on this subject, with this option the generator produces displayed data in LaTeX notation, as well as a file named `paut.tex` containing the parsing table in suitable form to be included in a LaTeX document.

Display options

There are several display options that the user can ask for in the command line to show some results or listings. All of them begin with `-d` and then a list of letters telling the display options wanted. As with other options, display options are significant only in some modes. See the example session below for their meaning.

B.2 A sample session

In this section, an example session is shown to give an idea of the dynamic of working with the tool, and about the kind of results that can be obtained.

When the program is run without arguments, the following help is displayed —also, when an error is encountered in the command line, the user is asked whether he wants the help to be displayed.

```
$ aut

LR(1) discriminating reverse automaton experimental tool (v10.0.3 9-10-1998)

Usage:

  aut g r|d[#] {-|+c -d{ailbfgsv} +f -g[#] -i +|-l -r[a] +s -t. +v -x file}
  aut s[#] {-# -d{bdfgvem} +e -g[#] +n[#] +v file}
  aut p r|d {-d{psu} -g[#] +v file}

 Functioning modes:
  g      parser Generation (reads gf; writes vf, pf)
  s      random Sentence generation (reads gf; writes sf), with random or
         given (#) length (quite approximately)
  p      Parse (reads vf, pf, sf) with mostly manual error recovery

 Automaton classes:
  r      Reverse discriminating
  d      Direct (classical)
  #      lookahead length: 0 or 1 (default)

 Files:
  -gf path  BNF Grammar File (default g.aut) with rules like:
               proc_list = "proc" heading ";" proc_list | .
  -sf path  Sentence text File (default s.aut) to parse; spaces separate
            terminals.
  -pf path  Parsing table File (default p.aut) in internal format.
  -vf path  Vocabulary File (default v.aut) in internal format.

 Options (some do not produce a usable parser):
  -#     Number of sentences
  -c     resolve (reverse) Conflicts (by asking)
  +c     Continue (reverse) state construction beyond conflicts
  -d     Display options:
    a     Automaton states; suboptions:
     i     state situations (also called Items)
     l     List actions for (stack symbol, lookahead) pairs
    b      intput BNF grammar
    d      sentential forms during Derivation
    f      First and follow sets
    g      internal Grammar
    p      (action counts and) stack windows during Parsing
    s      Statistics
    u      strict parsing User time
    v      Vocabulary; suboptions:
     e      estimated typical lEnghts of phrases
     m      Minimum and maximum lenghts of phrases
  +e     disable use of typical lEngths of phrases
  +f     disable Full rightparts computation in situations
  -g[#]  debuGging [option; default 0]
  -i     ask (reverse) Initial state
  +l     build (direct) LALR(1) automaton
  -l     signal (direct) LALR(1) conflicts in LR(1) automaton
  +n[#]  seed [Number; default 0]: disable randomize
```

```
  -r[a]  full (reverse) Recognizer [consider Actions]
  +s     build (direct) SLR(1) automaton
  -t.    produce parsing table in plain Text using '.' for empty entries
  +v     disable Verbose output
  -x     produce output and parsing table in LaTeX notation

Author address:  <Fortes Gálvez, José> jfortes@dis.ulpgc.es.
```

Before using the tool, let us have a look at the grammar with which we are going to work:

```
$ cat g.aut
E = E "+" T | T .
T = F R .
R = | "*" F R .
F = "(" E ")" | "a" .
```

Let us generate a random sentence (corresponding to seed number 28) of 15 symbols on default file `s.aut`, and display its derivation steps. You might not get the same sentence as below, but yours should be the same each time you run your program with the same command line below.

```
Invoked:  aut s15 +n28 -dd

<E>:15

<E>:11 + <T>:3

<E>:7 + <T>:3 + <F>:2 <R>:1

<T>:7 + <F>:2 <R>:3 + a

<F>:4 <R>:3 + a * <F>:2 <R>:1 + a

( <E>:3 ) * <F>:2 <R>:1 + a * a + a

( <E>:2 + <T>:2 ) * a + a * a + a

( <T>:2 + <F>:1 <R>:1 ) * a + a * a + a

( <F>:2 <R>:2 + a ) * a + a * a + a

( a * <F>:0 <R>:0 + a ) * a + a * a + a

( a * a + a ) * a + a * a + a

( a * a + a ) * a + a * a + a
```

Let us produce a discriminating-reverse parser for the grammar above, displaying the grammar itself (notice the rule numbers assigned), the automaton transition tables (`LA` and `ST` stand for lookahead and stack symbol,

respectively) and associated sets of situations, and some statistics (about the grammar and the generated parser). Some output has been deleted to save space.

```
Invoked:  aut gr -dgais

Input grammar statistics:
 Number of rules = 7
 Average length of rightparts = 1.86
 Average number of terminal symbols per rightpart = 0.71
 Vocabulary:
  Number of terminal symbols = 5
  Number of nonterminal symbols = 4

1: S' -> ^ E $
2: E -> E + T
3: E -> T
4: T -> F R
5: R ->
6: R -> * F R
7: F -> ( E )
8: F -> a

Discriminating reverse LR(1) automaton

State 1  [ 0 1 2 3 4 5 6 7 8 ]
 1 . : S' -> ^ E $ . , . $
 2 . : E -> E + T . , . $
 2 . : E -> E + T . , . +
 2 . : E -> E + T . , . )
 3 . : E -> T . , . $
 3 . : E -> T . , . +
 3 . : E -> T . , . )
 4 . : T -> F R . , . $
 4 . : T -> F R . , . +
 4 . : T -> F R . , . )
 5 . : R -> . , . $
 5 . : R -> . , . +
 5 . : R -> . , . )
 6 . : R -> * F R . , . $
 6 . : R -> * F R . , . +
 6 . : R -> * F R . , . )
 7 . : F -> ( E ) . , . $
 7 . : F -> ( E ) . , . +
 7 . : F -> ( E ) . , . *
 7 . : F -> ( E ) . , . )
 8 . : F -> a . , . $
 8 . : F -> a . , . +
 8 . : F -> a . , . *
 8 . : F -> a . , . )
 . 0 : S' -> ^ E . $ , $ .
 . 0 : S' -> . ^ E $ , ^ .
 . 0 : E -> E . + T , + .
 . 0 : R -> . * F R , * .
```

```
 . 0 : F -> ( E . ) , ) .
 . 0 : F -> . ( E ) , ( .
 . 0 : F -> . a , a .
 . 5 : T -> F . R , . $
 . 5 : R -> * F . R , . $
 . 5 : T -> F . R , . +
 . 5 : R -> * F . R , . +
 . 5 : T -> F . R , . )
 . 5 : R -> * F . R , . )
 . 0 : T -> F . R , * .
 . 0 : R -> * F . R , * .
 . 0 : T -> . F R , ( .
 . 0 : R -> * . F R , ( .
 . 0 : T -> . F R , a .
 . 0 : R -> * . F R , a .
 . 0 : E -> E + . T , ( .
 . 0 : E -> . T , ( .
 . 0 : E -> E + . T , a .
 . 0 : E -> . T , a .
 . 0 : S' -> ^ . E $ , ( .
 . 0 : F -> ( . E ) , ( .
 . 0 : S' -> ^ . E $ , a .
 . 0 : F -> ( . E ) , a .

LA(() = Shift
LA(a) = Shift
ST(a) = Reduce 8
ST(E) = Shift
ST($) = Go To 2
ST(T) = Go To 3
ST(R) = Go To 4
ST()) = Go To 5
ST(F) = Go To 6

        .
        .
        .

State 2  [ 1 ]
 1 . : S' -> ^ E . $ , . $

ST(E) = Go To 10

State 10  [ 1 ]
 1 . : S' -> ^ . E $ , . $

ST(^) = Reduce 1

End of generation.

Automaton statistics:
 Number of states = 10
 Number of transitions & actions = 26
```

Now, let us try the parser with the previously generated sentence and display the parsing stack just before each reduction. The pairs of numbers at the left of each line count the number of shifts and reduces. The terminals between parenthesis before each nonterminal correspond to the first terminal of its corresponding phrase in the sentence. Some parsing statistics are also displayed.

```
Invoked:  aut pr -dps
2,0: ^ ( a           * a + a ) * a + a *
4,1: ^ ( (a)F * a           + a ) * a + a * a +
4,2: ^ ( (a)F * (a)F           + a ) * a + a * a +
4,3: ^ ( (a)F * (a)F (+)R           + a ) * a + a * a +
4,4: ^ ( (a)F (*)R           + a ) * a + a * a +
4,5: ^ ( (a)T           + a ) * a + a * a +
6,6: ^ ( (a)E + a           ) * a + a * a + a $
6,7: ^ ( (a)E + (a)F           ) * a + a * a + a $
6,8: ^ ( (a)E + (a)F ())R           ) * a + a * a + a $
6,9: ^ ( (a)E + (a)T           ) * a + a * a + a $
7,10: ^ ( (a)E )           * a + a * a + a $ $
9,11: ^ (()F * a           + a * a + a $ $ $ $
9,12: ^ (()F * (a)F           + a * a + a $ $ $ $
9,13: ^ (()F * (a)F (+)R           + a * a + a $ $ $ $
9,14: ^ (()F (*)R           + a * a + a $ $ $ $
9,15: ^ (()T           + a * a + a $ $ $ $
11,16: ^ (()E + a           * a + a $ $ $ $ $ $
13,17: ^ (()E + (a)F * a           + a $ $ $ $ $ $ $ $
13,18: ^ (()E + (a)F * (a)F           + a $ $ $ $ $ $ $ $
13,19: ^ (()E + (a)F * (a)F (+)R           + a $ $ $ $ $ $ $ $
13,20: ^ (()E + (a)F (*)R           + a $ $ $ $ $ $ $ $
13,21: ^ (()E + (a)T           + a $ $ $ $ $ $ $ $
15,22: ^ (()E + a           $ $ $ $ $ $ $ $ $ $
15,23: ^ (()E + (a)F           $ $ $ $ $ $ $ $ $ $
15,24: ^ (()E + (a)F ($)R           $ $ $ $ $ $ $ $ $ $
15,25: ^ (()E + (a)T           $ $ $ $ $ $ $ $ $ $
16,26: ^ (()E $           $ $ $ $ $ $ $ $ $ $

Successful end of parsing.

Statistics:
 Number of shift actions = 16
 Number of reduce actions = 26
 Average length of reductions = 1.54
 Maximum stack length = 7
 Average stack length = 3.95
 Average depth of stack exploration per shift action = 0.50
 Average depth of stack exploration per reduce action = 2.00
 Average number of state transitions per action = 0.86
```

Let us now compare the above results with a direct canonical LR(1) parser. First, let us generate the parser, displaying only the automaton tables (`NS` stands for next symbol, that is, current lookahead) and some statistics (the output has been partially deleted).

```
Invoked:  aut gd -das

Input grammar statistics:
 Number of rules = 7
 Average length of rightparts = 1.86
 Average number of terminal symbols per rightpart = 0.71
 Vocabulary:
  Number of terminal symbols = 5
  Number of nonterminal symbols = 4

Direct (classical) LR(1) automaton

State 1

NS(E) = Go To 2
NS(T) = Go To 3
NS(F) = Go To 4
NS(() = Shift 5
NS(a) = Shift 6

State 2

NS($) = Shift 7
NS(+) = Shift 8

        .
        .
        .

State 26

NS(*) = Reduce 7
NS()) = Reduce 7
NS(+) = Reduce 7

State 27

NS()) = Reduce 6
NS(+) = Reduce 6

End of generation.

Automaton statistics:
 Number of states = 27
 Number of transitions & actions = 80
```

Finally, let us try the generated direct parser with the same sentence. In this case, before each parsing stack state its corresponding symbol is shown. As before, the stack is only displayed before each reduction.

```
Invoked:  aut pd -dp
(:5 a:15           * a + a ) * a + a *
(:5 F:13 *:21 a:15           + a ) * a + a * a +
```

```
(:5 F:13 *:21 F:25           + a ) * a + a * a +
(:5 F:13 *:21 F:25 R:27           + a ) * a + a * a +
(:5 F:13 R:20           + a ) * a + a * a +
(:5 T:12           + a ) * a + a * a +
(:5 E:11 +:19 a:15           ) * a + a * a + a $
(:5 E:11 +:19 F:13           ) * a + a * a + a $
(:5 E:11 +:19 F:13 R:20           ) * a + a * a + a $
(:5 E:11 +:19 T:24           ) * a + a * a + a $
(:5 E:11 ):18           * a + a * a + a $ $
F:4 *:10 a:6           + a * a + a $ $ $ $
F:4 *:10 F:17           + a * a + a $ $ $ $
F:4 *:10 F:17 R:23           + a * a + a $ $ $ $
F:4 R:9           + a * a + a $ $ $ $
T:3           + a * a + a $ $ $ $
E:2 +:8 a:6           * a + a $ $ $ $ $ $
E:2 +:8 F:4 *:10 a:6           + a $ $ $ $ $ $ $ $
E:2 +:8 F:4 *:10 F:17           + a $ $ $ $ $ $ $ $
E:2 +:8 F:4 *:10 F:17 R:23           + a $ $ $ $ $ $ $ $ $
E:2 +:8 F:4 R:9           + a $ $ $ $ $ $ $ $ $
E:2 +:8 T:16           + a $ $ $ $ $ $ $ $ $
E:2 +:8 a:6           $ $ $ $ $ $ $ $ $ $
E:2 +:8 F:4           $ $ $ $ $ $ $ $ $ $
E:2 +:8 F:4 R:9           $ $ $ $ $ $ $ $ $ $
E:2 +:8 T:16           $ $ $ $ $ $ $ $ $ $
E:2 $:7           $ $ $ $ $ $ $ $ $ $

Successful end of parsing.
```