Adding New Diagnostic Syntax Error Messages to Unicon

Clinton Jeffery

Unicon Technical Report: 19

2017-05-28

Abstract

Unicon uses a parser generated by iyacc, a modified version of Berkeley YACC. Syntax error messages are produced using the same two keys that YACC uses to determine what to do at each step: the parse state and the current token. Whenever the language grammar is modified, these parse states change in unobvious ways. To solve the problem of mapping changing parse states and tokens onto diagnostic messages, a tool named merr is used on a set of example error fragments to associate messages with corresponding parse states and tokens. This technical report describes the process of adding new diagnostic messages.

Unicon Project
http://unicon.org

University of Idaho
Department of Computer Science
Moscow, ID, 83844, USA

1 Introduction

This technical report describes how to add new syntax error messages to the Unicon translator. When it encounters a syntax error with which no message is associated, Unicon emits a generic message along the lines of the following.

```
synerrfiletest.icn:32: # "end" syntax error (232;272)
```

The two integers in parentheses, separated by a semi-colon, are the parse state and the lexical token at the point of error. It is these two integers that YACC uses to determine that there is a syntax error, and these two integers that Unicon uses to decide what message to write.

2 A Brief History

Unicon's grammar descends from Icon, and Icon's syntax error messages are keyed off YACC parse states, which are integers internal and specific to whichever YACC implementation is used. For example, Icon traditionally used AT&T YACC. If its grammar were run through GNU Bison, the integers would be different and the syntax errors emitted would become nonsensical.

In addition to this portability problem, any change to the Icon grammar meant a painful and mysterious task of updating a manually-maintained table that mapped parse states to error messages. The difficulty of this task was severe enough to reinforce the project owner's reluctance to change the syntax, constraining research directions albeit adding "language stability through unmaintainability".

A custom tool called *merr* was developed for Unicon in order to solve the problem of updating the syntax error message table[1]. Along the way, *merr* added the ability to disambiguate error messages for a given parse state, depending on the current input token. Compared with Icon, Merr also frees developers to use whichever YACC implementation they wish (AT&T YACC, Berkeley YACC, or Bison, for example). In the case of Unicon, there was no option of sticking with Icon's error message table even if it could have been updated to handle Unicon's additions, as AT&T YACC is proprietary, and iyacc is based on Berkeley YACC. It was essential that a new error message table be created, and *merr* provided the missing piece.

3 Running Merr During the Unicon Translator Build Process

Merr is included in Unicon source distributions. It is simply named merr.icn in the unicon translator directory, unicon/uni/unicon. A Merr executable is built by saying "unicon merr" in that directory.

The makefile rules for running Merr during the Unicon translator build are included in the unicon/uni/unicon makefile, but are commented out. In order to run merr, the lines

```
#yyerror.icn: unigram.icn
#           merr -u unicon
```

must be uncommented. On some platforms you may need to say `./merr -u ./unicon` or similar, to avoid path problems.

Adding these makefile rules will cause merr to run, and generate a new yyerror.icn whenever unigram.icn is changed, generally due to iyacc being run on unigram.y. The Unicon translator uses some slight modifications to the generic yyerror.icn that *merr* generates by default. The -u option asks merr to update the existing yyerror.icn body with the current state tables calculated using the error fragments. If merr is run without -u, the yyerror.icn should be hand-modified after merr is run, to fix things up as follows.

- write() should be changed to iwrite()
- the parse errors should be bundled onto global list parsingErrors instead of being written out directly.

These changes facilitate the use of the Unicon parser in the IDE, both directly (as part of syntax checking) and indirectly, so that error messages show up in the IDE error box and not just on standard error output, which is easily missed in a GUI application.

4 Adding New Error Messages

Error fragments, and their associated error messages, are stored in a file named meta.err in the unicon/uni/unicon directory. The existing file contains about 75 error fragments, and was created by starting with all the Icon error message fragments, and adding messages pertaining to Unicon-specific syntax such as class declarations.

The format of the fragments and messages is

fragment
::: *message*

which is to say: one or more lines of source code that will produce a syntax error, ending with a line consisting of three colons followed by the syntax error message to use when that fragment (or any source code resulting in the same parse state) occurs. If multiple fragments result in the same parse state, the first such fragment gives the default syntax error message, but other fragments with that parse state will give messages for specific input tokens encountered at the point that the syntax error occurs.

5 Example

The following example, derived from bug #168 on the Unicon bug tracker on Source Forge, illustrates the process. Thanks to Charles Evans for reporting and/or fixing many bugs such as this. The error fragment is just a test program, but you should strip it down to the minimum required to produce the error. The test file can be named whatever you like; the filename is not preserved when it is placed into the merr meta.err file as a fragment. I typically use synerr.icn, or synerrN.icn for some integer N when I have multiple errors in a directory. In this particular case the error code looks like

```
procedure action_12()
yyval :=
end
```

The default diagnostic you get, before adding the error fragment and running merr, looks like:

```
unicon synerr
Parsing synerr.icn:
synerr.icn:3: # "end" syntax error (232;272)
(use -M for assistance with this error)
```

What should the compiler have said? Perhaps “Assignment is missing right operand value.” Fine. Add all this to the end of meta.err:

```
procedure action_12()
yyval :=
end
::: Assignment is missing right operand value.
```

Run the commands “unicon merr” followed by “merr -u unicon”; again you may need to put ./ or .\in front of merr and unicon in these examples. If all goes well, a new yyerror.icn is written, adding a table entry for parse state 232 that we need. A diff taken by running merr without the -u option shows what Unicon’s yyerror() customizes compared with the generic yyerror.icn that merr writes out. Customization adds about 20 lines to be copied back into yyerror.icn if merr is run without -u.

```
— yyerror.icn (revision 5236)
+++ yyerror.icn (working copy)
@@ -65,6 +65,7 @@
+ t[232] := table("Assignment is missing right operand value.")
@@ -91,43 +92,16 @@
- if __merr_errors = 0 then iwrite(&errout)
+ if __merr_errors = 0 then write(&errout)
```

```
else if map(s)== "stack underflow. aborting..." then return
__merr_errors +:= 1
- if __merr_errors > 10 then {
- if *\parsingErrors > 0 then {
- every pe := !parsingErrors do {
- iwrite(\&errout, pe.errorMessage)
- }
- }
- istop("too many errors, aborting")
- }
- prefix := (\yyfilename||":") | ""
+ if __merr_errors > 10 then
+ write("too many errors, aborting") & stop()
if s == "syntax error" then
s := t[(\statestk)[1], yychar]
if s == "syntax error" then {
s ||:= " (" || (\statestk)[1] ||";"|| yychar || ")"
-
}
- s := prefix ||yylineno||": # \""|| yytext || "\" " || s
- if \merrflag then {
- if ferr2 := open(\yyfilename) then {
- iwrite(&errout, "Reporting (-M) your error to the Oracle (",
- merraddress, ") for assistance.")
- iwrite(&errout, "Type any question you have on the lines below.",
- " Type a blank line to finish.")
- ferr := open("mail " || merraddress, "pw")
- while iwrite(ferr, ""  == read())
- iwrite(ferr)
- iwrite(ferr, s)
- iwrite(ferr)
- while iwrite(ferr, read(ferr2))
- close(ferr2)
- close(ferr)
- }
- }
- /parsingErrors := []
- errorObject := ParseError( yylineno, s )
- put( parsingErrors, errorObject )
+ write(&errout, (\yyfilename|"lambda.icn"),
":",yylineno, ": # \"", yytext, "\": ", s)
```

We keep the new line added at 65, but restore the rest of the Unicon customizations to yyerror.icn so that the commit is just the one line addition to the table. Now the Unicon diagnostic for this input looks like

```
$ unicon synerr
Parsing synerr.icn:
synerr.icn:3: # "end": Assignment is missing right operand value.
```

6 Conclusions

Using Merr to update the Unicon translator with new messages is easy, but all computational tasks require care. It will be nice if there someday exists a tool that will generate *all* error fragments directly from the grammar, so that no undefined error messages occur unless new grammar rules were added. Until that day, error fragments should be added as they are encountered.

References

[1] Merr: an LR Syntax Error Message Tool. unicon.org/merr