C99 Parser User Guide
Matt Wette
June 2019
With NYACC Version 1.04.0

Introduction

This is a manual for the C99 parser provided with NYACC. It is a LALR(1) based parser written in Scheme as implemented in Guile. The grammar can be examined by looking at the source code in the file `nyacc/lang/c99/mach.scm`. The parser has been used to generate object code (see `https://www.gnu.org/software/mes/`) but it has options to make it useful for processing C code in other ways. For example, the FFI Helper included with NYACC uses the parser to generate FFI (foreign function interface) code for many C libraries. It does this by parsing the associated C header files.

`parse-c99` [*options*] [Procedure]

where options are

`#:cpp-defs` *defs-list*
: *defs-list* is a list of strings where each string is of the form *NAME* or *NAME=VALUE*.

`#:inc-dirs` *dir-list*
: *dir-list* is a list of strings of paths to look for directories.

`#:inc-help` *helpers*
: *helpers* is an a-list where keys are include files (e.g., `"stdint.h"`) and the value is a list of type aliases or CPP define (e.g., `"foo_t" "FOO_MAX=3"`). The default helper is `c99-def-help` (see below).

`#:mode` *mode*
: *mode* is one literal `'code`, `'file`, or `'decl`. The default mode is `'code`.

`#:debug` *bool*
: If *bool* evaluates to true, print productions as they are reduced.

This will parse the content taken from the current input port until end if input is reached. A parse tree in the form of an SXML expression is returned. See below for the syntax. This needs to be explained in some detail. tdd = typedef dict: (("<time>" time_t) ... ("<unistd.h>" ...)) Default mode is `'code`.

```
(with-input-from-file "abc.c"
  (parse-c #:cpp-defs '("ABC=123"))
           #:inc-dirs '("." "./incs" "/usr/include"))
           #:inc-help (append '("myinc.h" "foo_t" "bar_t") c99-std-help)
           #:mode 'file))
```

Note: for `file` mode user still needs to make sure CPP conditional expressions can be fully evaluated, which may mean adding compiler generated defines (e.g., using `gen-cpp-defs`).

Include Helpers

The C99 parsers can use "include helpers". This allows files to be parsed without reading full include files. The user provides typenames (types defined using `typedef`) and defines. The syntax for the include-helper optional argument to the parsers is

```
(define my-inc-helper
 '(("foo.h" "foo_t" "ABC=123" "SUM(X,Y)=((X)+(Y))")
   ("bar.h" "bar_t" "DEF=456" "MAX(X,Y)=((X)>(Y)?(X):(Y))"))
```

The special helper `"__builtin"` will be "included" automatically at the start of parsing. This allows one to generate definitions for compiler builtins like `__builtin_va_list`.

```
(define inc-helper
 '(("__builtin" "__builtin_va_list=void*")))
```

If no inc-helper is provided, the default is `c99-def-help`, which is defined (in the module `(nyacc lang c99 util)`) as

```
(define c99-def-help
  '(("__builtin"
     "__builtin_va_list=void*"
     "__inline__=inline" "__inline=__inline__"
     "__restrict__=restrict" "__restrict=__restrict__"
     "__signed__=signed" "__signed=__signed__"
     "asm(X)=__asm__(X)" "__asm(X)=__asm__(X)"
     "__attribute(X)=__attribute__(X)"
     "__volatile__=volatile" "__volatile=__volatile__"
     "__extension__=" "__extension=__extension__"
     "asm=__asm__" "__asm=__asm__"
     "__attribute(X)=__attribute__(X)")))
```

The module `(nyacc lang c99 util)` also defines `c99-std-help`, which includes the above and typedefs and CPP defines for many standard includes (e.g., `alloca.h`, `limits.h`). See the source `nyacc/lang/c99/util.scm` for more detail.

Misc Items

The special symbol `C99_ANY` can be used for symbols which you don't want to define. In the parser will handle this as XXX

Stuff

Note on CPP replacement text: IIRC, C99 will remove comments from CPP statements before processing. I preserve this and remove inside the CPP parser.

The Unit Parser

TALK ABOUT `fixed-width-int-names`

`parse-c99` [*#:cpp-defs def-a-list*] [*#:inc-dirs dir-list*] [*#:mode* (*'code*|*'file*)] [*#:debug bool*] [Procedure]

This needs to be explained in some detail. Default mode is `'code`.

```
(with-input-from-file "abc.c"
```

```
(parse-c #:cpp-defs '("ABC=123"))
         #:inc-dirs (append '("./incs") c99-std-dict)
         #:inc-help '(("myinc.h" "foo_t" "bar_t"))
         #:mode 'file))
```

Modes

There are several modes for parsing which affect the way the C preprocessor statements are handled, and how the parse tree is generated. The following list explains the intent behind these parsing modes. Later we mention some fine points.

- *code* mode (the default) In this mode, the proprocess works like a normal C compiler. The preprocessor statements are evaluated as they are read and macros in the code are expanded as they are read.
- *decl* mode This mode is intended to be used for tools which want to extract the declarations and definitions which are explicit in a file, but allow access to declarations and definitions in included files.
- *file* mode is intended to be used for tools which want to transform C files somehow. For example, one could parse a file and remove all comments. This will keep the CPP structure at the top level. Preprocessor statements at the top level are not evaluted. Note: There is a change in versions starting with 0.77.0. In these all defines required for evaluating CPP expressions in if-then have to be resolved.

Options are as follows

`#:cpp-defs`
: This is a list of define strings (e.g., `'("ABC=123")`.

`#:inc-dirs`
: This is an ordered list of directories to search for include files.

`#:inc-help`
: This is an a-list of include helpers, where keys are the include file or path (e.g., `sys/types.h`).

`#:mode` This is the mode: `'code`, `'decl` or `'file`. The default is `'code`.

`#:xdef?` This is a predicate function to determine whether to expand a definition (used in file mode). See below.

Note: The user needs to define `"__has_include(X)=__has_include__(X)"` to enable has-include; `"__has_include=__has_include__"` will not work. (Should I worry that it does not?)

`name` *mode* => *#t*|*#f* [xdef?]
: Given string *name* and *mode* indicate whether the parser should expand using CPP defines. The default is `(lambda(name mode) (eqv? mode 'code))`.

Expression Parser

To be documented.

Copying