Using Make to Manage Large Proofs

Phillip J. Windley

Laboratory for Applied Logic
Department of Computer Science
Brigham Young University
Provo, Utah 84602-6576

windley@cs.byu.edu

Abstract

Large proofs involve many theories which are usually built from separate files. This note describes how the standard UNIX make utility can be used to help manage large proofs. We provide a general `Makefile` that can be used with few changes and a program for automatically generating dependencies between HOL theories.

1 Introduction

`make` is a command generator that is distributed with UNIX and is also available for a large number of other operating systems. Using a description file, `make` generates a sequence of commands that are executed by the shell. What makes `make` most useful, however, is its ability to use dependency relations between separate modules to order the building of those modules [OT91].

Large proofs in the HOL theorem proving system are performed using multiple files which create individual theories. Because of the hierarchical nature of HOL theories (theories can have parents), these theories form a dependency graph. Maintaining the proof, in these cases, involves creating multiple theories in the right order—something `make` was made for.

2 A Brief Introduction to `make`

As stated in the last section, `make` generates a series of commands which are then executed by the shell. `make` generates these commands according to the instructions in a program of sorts called the `Makefile`.

The `Makefile` contains instructions about how to build a *target* from a *source*. These instructions can be very general and usually consist of instruction for building a particular kind of object file from a particular kind of source file. The file types being denoted by the file extension.

`make` uses these instructions to rebuild out of date targets (i.e. those whose date of creation or modification is earlier than that of the source file). `make` can also be told about dependencies between files using a dependency graph (expressed textually).

3 Dependencies

Dependencies between theories in HOL are created using the `new_parent` command. Because of these theory dependencies, ML files must be loaded into HOL according to the partial order introduced by the dependency graph. `make` is a natural fit to this problem since it executes commands in a partial order base on the dependency graph.

Creating dependency graphs for `make` can be a chore when there are a number of files. We have developed a program, `depgen` that creates dependency graphs for `make` from ML programs that create and use HOL theories.

For any set of ML files, `depgen` will determine the dependencies between files created using `new_theory` and theories that use them via `new_parent`. `depgen` knows about several popular aliases for declaring theories as parents as well such as `map new_parent`, `load_parent`, and `map load_parent`. Commands line switches allow users to extend this set with any personal favorites.

The UNIX manual page for `depgen` is in Appendix A.

4 Makefile

In addition to the dependency generator described in the last section, we have developed a `Makefile` for HOL that can be used, with very few changes, to manage HOL proofs.

The key to this is a rule in the `Makefile` for generating HOL theory files (with a `.th` extension) from the corresponding ML files (with a `.ml` extension): [1]

```
# ==================================================================
# Add new suffixes for ML and theory files
# ==================================================================
.SUFFIXES: .ml .th
```

[1]Appendix B contains the complete `Makefile.base` file.

```
# ========================================================================
# Default action for creating a .th file from a .ml file.
# ========================================================================
.ml.th:
@ echo Making \${Home}/\$*.th...;\\
  \${RM} \$*.th;\\
  echo 'set_flag(`abort_when_fail`,true);;'\\
             'set_flag(`prompt`,false);;'\\
             'set_flag(`sticky`,true);;'\\
             'set_flag(`timing`,true);;'\\
             'set_search_path (search_path() @ '\\
             '   [`\${Root}/ml/`;'\\
             '   ]);;'\\
             'loadf `\${Root}/ml/aux_defs.ml`;;'\\
     'loadt `\${Home}/\$*.ml`;;'\\
     'quit();;' | \${Hol} || \${RM} \$*.th
```

This rule tells `make` that a `.th` file can be created from the `.ml` file on which it depends by executing the listed ML commands inside HOL. This is accomplished by echoing the ML commands into HOL through a pipe. The variables (denoted by a prepended $) in the rule are `Makefile` variables defined elsewhere in the `Makefile` with the exception of `$*` which is replaced with the name of the source file.

5 Use

Using the HOL `Makefile` is easy:

1. The file `Makefile.base` is placed in the directory where the proof is being built.
2. Ensure that all of the files have the same name up to the extension as the theory files they create.
3. Edit `Makefile.base` to reflect its location, the location of the HOL executable, and the location of any personal library directories.
4. At the command prompt, type "`make -f Makefile.base rebuild`"

This will create a `Makefile` with the correct dependency list. You can build any of the theories that depend on the `.ml` files in the directory by typing "`make file.th`". `make` will automatically ensure that any theories that `file.th` depends on are up to date.

6 Conclusion

Using `make` to maintain large proofs makes the job of rebuilding theories much easier. Changes to theories can be conveniently incorporated into the proof from the command line.

The author would like to thank Steve Bancroft who wrote the original `depgen` program for HOL in 1989.

References

[OT91] Andrew Oram and Steve Talbott. *Managing Projects with Make*. O'Reilly & Associates, Inc., 1991.

A depgen

```
DEPGEN(1)                                                         DEPGEN(1)

 NAME
      depgen - HOL Dependency List Generator

 SYNOPSIS
      depgen [ -m ] [ -t theory-search-string ]
                    [ -p parent-search-string ] source-files

 DESCRIPTION

      depgen is a dependency list generator for HOL source code files. In a given
      list of source files (usually specified by *.ml), for every "new theory" (and
      its corresponding parent) declaration, a list of dependencies will be
      generated.

 FLAGS
      -m      Display as dependencies only the theories declared within the source
              files searched (to be compatible with the Unix 'make' facility.)

      -t      Add a customized search argument for "new theory" declarations. Adds to
              the current list of built-in "new theory" search strings. May be used
              an arbitrary number of times. Search argument "new_theory" is built-in,
              and does not need to be declared on the command line.

      -p      Add a customized search argument for a parent declaration.  Adds to the
              current list of built-in parent search strings.  May be used an
              arbitrary number of times. Search arguments "load_parent", and "map
              load_parent" are built-in.
```

```
EXAMPLE
     % depgen -m -t my_theory -p get_parent *.ml

     This will generate a dependency listing depending upon the source files
     specified on the command line. The listing will be generated in 'makefile'
     format, with 'my_theory' as an additional theory declaration search string, and
     'get_parent' as an additional parent declaration search string.

     Sample Output:

     alu.th: alu.ml aux.th gen.th
     aux.th: aux.ml
     gen.th: mkgen.ml muxdef.th
     muxdef.th: defmux.ml

     In this example, four source files were specified on the command line, alu.ml,
     aux.ml, mkgen.ml, and defmux.ml.  As shown on the first line, the theory alu.th
     was declared inside alu.ml, with aux.th and gen.th as declared parent
     dependencies.  In aux.ml, the theory aux.th was declared, with no parent
     dependencies. . etc..

BUGS
     If, for some strange reason, a source file contains the text `new_theory` (the
     text new_theory delimited by two graves) an incorrect dependency list will be
     generated for that file.  If it is encountered when -m is specified, the whole
     dependency listing may be affected.

AUTHOR
     Steven M. Bancroft / University of California, Davis

SEE ALSO
     make(1)
```

B Makefile.base

```
# ======================================================================
#
#   MAKEFILE FOR HOL THEORIES
#
# ======================================================================

# ======================================================================
#
# MAIN ENTRIES:
#
# make all     : create theories and compile code
#
```

```
# make clean     : remove theories and compiled code
#
# ======================================================================

# ======================================================================
# MACROS:
#
# Name      : name of library
#
# Hol     : the pathname of the version of hol used
#
# Root     : root directory for the directory holding this file
#
# Home      : directory (relative to Root) that contains this file
#
# Library   : root directory for library
#
# ML        : list of ML sources
#
# ======================================================================

Name=UINTA

RM=/bin/rm

Hol=/usr/local/bin/hol

Root=/users/windley/hol

Home=${Root}/uinta

Lib=${Root}/Library

Backup=laforge

RootTheory=

default:
@echo "===> ${Name}. Type \"make all\" or \"make clean\""

clean:
-/bin/rm -f *_ml.o
@echo "===> ${Name}. All object code deleted"

clobber:
-/bin/rm -f *_ml.o *.th print
@echo "===> ${Name}. All object code and theory files deleted"

# ======================================================================
# Add new suffixes for ML and theory files
# ======================================================================
```

```
.SUFFIXES: .ml .th

# ======================================================================
# Default action for creating a .th file from a .ml file.
# ======================================================================
.ml.th:
@ echo Making ${Home}/$*.th...;\
  ${RM} $*.th;\
  echo 'set_flag(`abort_when_fail`,true);;'\
             'set_flag(`prompt`,false);;'\
             'set_flag(`sticky`,true);;'\
             'set_flag(`timing`,true);;'\
             'set_search_path (search_path() @ '\
             '   [`${Root}/ml/`;'\
             '   ]);;'\
             'loadf `${Root}/ml/aux_defs.ml`;;'\
     'loadt `${Home}/$*.ml`;;'\
     'quit();;' | ${Hol} || ${RM} $*.th


# ======================================================================
# List of ML source texts
# ======================================================================
ML=  ####SOURCES####


all: ${RootTheory}
@echo "===> ${Name} rebuilt on `date`"

# ======================================================================
# Print recently changed files
# ======================================================================
print: ${ML}
-@for f in $?; \
  do a2ps $$f; \
     echo "Printing $$f" ;\
  done
 @touch print

# ======================================================================
# Transfer changed files by rcp to backup machine
# ======================================================================
backup: ${ML} Makefile Makefile.base
@-for f in $?; \
  do rcp $$f ${Backup}:${Home}; \
     echo "Transfering $$f" ;\
  done
 @touch backup

rebuild:
${RM} -f Makefile
```

```
make -f Makefile.base Makefile

# ======================================================================
Makefile:
${RM} -f $@
sed -e "s/####SOURCES####/`echo *.ml`/" \
    -e '/^Makefile/,$$ d' <Makefile.base >> $@
echo '# automatically generated dependency list: ' >> $@
depgen -m *.ml >> $@
chmod -w $@
```