autoconf
Jonathan Lennox
Department of Computer Science
Columbia University

Software portability
- Many software products need to run on lots of platforms
  - Unix, Windows, (old) Macintosh, VMS, ...
  - Varieties of Unix: Linux, Solaris, SunOS 4.x, Free/Net/OpenBSD, MacOS X (Darwin), Tru64, AIX, HP/UX, SVR4, SCO, Minix, ...
  - Open-source software especially needs to be portable
  - Create a developer community

Historical Practice (1)
- Ignore the problem
  - 1982: "All the world's a VAX" (running BSD 4.2)
  - 1992: "All the world's a Sun" (running SunOS 4.x)
  - 2002: "All the world's Linux" (on an x86)
  - This is great, for as long as it's true...

Historical Practice (2)
- Sea of platform-specific #ifdef's
  ```c
  #ifdef __linux__
    #ifdef defined(__sun__) && defined(__svr4__) && Solaris */
    #ifdef __sun_cc
      else
      #endif
    #endif
  #endif
  #error "What system is this?"
  #ifndef
  #endif
  
  This only works for platforms you've already ported your code to
  Can quickly become unmanageable

Historical Practice (3)
- Makefile documents --D flags, -L flags, etc., to pass to compiler for specific systems or compilation options
- User modifies the program's Makefile by hand, in a text editor
- Works okay for very small projects; runs into problems very quickly
- Error-prone; users often forget to specify flags, mis-type them, or give the wrong ones
- Porting to a new platform is very difficult

Historical Practice (4)
- Variant of (3): interactive scripts to set all the options
  - Run a shell script. It asks you lots of questions like "does this system support the angle(3) function with extended froundz? (y/n):"
  - Shell script automatically creates your make file
  - Very bad for inexperienced software builders
  - Not (easily) possible to build software non-interactively
  - With good per-system defaults, this can work, however. (Perl's build system works like this)
Solution: autoconf

- 'configure' script
  - Automatically checks the characteristics of the build system
  - Programs, libraries, header files, system calls
  - Generates a Makefile from a programmer-supplied template
  - Generates a config.h file defining compiler and system characteristics

Autoconf file flows:

Developer

configure.in  autoconf configure
aclocal.m4
acsite.m4  autoheader config.h

Builder

./configure; make

Typical configure.in

del Foxes this file with autoconf to produce a configure script.
AC_MAINTAINER([maintainer]) [1.0]
AC_CONFIG_PROGRA M(dir) [1.0]
AC_CONFIG_HEADERS([config.h]) [1.0]
del Checks for programs.
AC_PROG_CC [1.0]
del Checks for libraries.
AC_LIBDIR([libdir]) [1.0]
AC_CONFIG_AR([ar]) [1.0]
del Thanks for header files.
AC_PROG_AR([ar]) [1.0]
AC_PROG_RANLIB([ranlib]) [1.0]
AC_PROG_INSTALL([install]) [1.0]
del Thanks for library functions.
AC_PROG_LIB [1.0]
AC_LIB [1.0]
del Checks for library functions.
AC_PROG_LIB [1.0]
AC_LIB [1.0]

Typical configure run

$ configure
rm -rf .config
$ configure
configure

Autoconf file flows:

Builder

Makefile.in

configure config.status Makefile

config.h.in

config.h

config.cache

config.log

10-Apr-02 Advanced Programming Spring 2002 7

10-Apr-02 Advanced Programming Spring 2002 8

10-Apr-02 Advanced Programming Spring 2002 9

10-Apr-02 Advanced Programming Spring 2002 10

10-Apr-02 Advanced Programming Spring 2002 11

10-Apr-02 Advanced Programming Spring 2002 12
**configure.in structure**
- AC_INIT (package, version, [bug-report-address])
  - start configure.in
- AC_CONFIG_SRCDIR (unique-file-in-source-dir)
  - uniquely identify source directory
- AC_CONFIG_FILES (file... [cmds], [init-cmds])
  - create files from templates (e.g. Makefile)
- AC_CONFIG_HEADERS (header... [cmds], [init-cmds])
  - create header files (e.g. config.h)
- AC_OUTPUT
  - output all the generated files

**configure.in: program checks**
- Autoconf can check if the build system has certain programs
  - AC_PROG_AWK
    - Sets output variable AWK to awk, gawk, nawk, or awk
  - AC_PROG_LEX / AC_PROG_YACC
    - Find lex (flex) or yacc (byacc, bison)

**configure.in: compiler checks**
- Autoconf can find the compiler and check its characteristics
  - AC_PROG_CC, AC_PROG_CXX
    - Find the C or C++ compiler
  - AC_PROG_C_STDCC
    - Check if the C compiler is ANSI C, after trying various compiler options to make it do so
  - AC_C_CONST
    - Check if the C compiler supports 'const'
      - If not, define const to the empty string, so you can use it anyway
  - AC_C_BIGENDIAN
    - Check if the system is "big-endian"; i.e., stores integers most-significant byte first (i.e., big-endian; 108 & 100 in little-endian)

**configure.in: library checks**
- Autoconf can determine whether certain libraries are available
  - AC_CHECK_LIB(library, function)
    - Check whether the library can be linked (as -library), and then whether the function can be found inside it
    - Once a library is found, it's linked in by default for future calls to AC_CHECK_LIB, so you can have one library depend on another.

**configure.in: header checks**
- Autoconf can check whether certain header files are available
  - AC_CHECK_HEADER
    - Check whether a header file is available
  - AC_HEADER_STDCC
    - Check whether the system header files conform with ANSI C. (Not the same as AC_PROG_C_STDCC or _STDCC_)!
  - AC_HEADER_TIME
    - Check whether `<time.h> and <sys/time.h>` can be included

**configure.in: type checks**
- Autoconf can check characteristics of structures and types in the compilation environment
  - Type checking code #includes all detected header files checked so far
    - AC_CHECK_MEMBER(aggregate, member)
      - Check whether the given aggregate (struct or union) is defined, and if so, whether it contains the given member
    - AC_CHECK_TYPE(type)
      - Check whether the compiler knows about a specific type
    - AC_TYPE_SIZE_T
      - Check whether the compiler knows about the type `size_t` if not, typedef to `unsigned`.
configure.in: function checks

- Autoconf can check whether system and library functions are available
  - AC_CHECK_FUNCS(functions...)
    - Check whether the given functions are available
  - AC_REPLACE_FUNCS(functions...)
    - Check whether the given functions are available, and if not, link in replacement code re-implementing them

Autoconf output: Makefile

- Some autoconf output is needed by the Makefile, so is defined as template substitutions
  - libraries, programs, search paths
- Developer must write Makefile: template Makefile
  - Other than template variables, looks exactly like a normal Makefile
- Patterns in Makefile.in are substituted with results of autoconf tests
  - @CC@ → C compiler
  - @AWK@ → Awk executable
  - @CFLAGS@ → compiler flags
  - @LIBS@ → linked libraries

Autoconf output: config.h

- Other autoconf output is needed by the source code, so symbols are defined in config.h.
- Source code includes "config.h", then makes decisions based on the symbols defined.

System-dependent tests

- Some things can't be checked automatically
  - Things that only work as root
  - Details of system object formats
- For these, autoconf provides system-dependent checks
  - Check the system type of either the build or the host system
    - Standard GNU naming system
      - idtable-unknown-linux-gnu
      - sparse-sun-solaris
  - Use shell pattern matching on these names

Custom tests

- Sometimes you need to check for things that autoconf doesn't already have tests for
- You can write custom tests:
  - AC_TRY_CROSS, AC_TRY_COMPILE,
    AC_TRY_LINK, AC_TRY_RUN
    - Try to preprocess / compile / link / run a specific fragment of C code.
    - Specify actions to take if test succeeds or fails.

Results of custom tests

- Custom tests need to be able to output their results
  - AC_DEFINE
    - Define a C preprocessor symbol in config.h
  - AC_SUBST
    - Substitute a variable pattern into Makefile
  - AC_CACHE_CHECK
    - Cache a variable in config.cache for future configure runs
  - AC_MSG_CHECKING / AC_MSG_RESULT
    - Output messages telling the user that something's being checked
Subtleties of custom tests

- Autoconf actually works by using the `m4` macro processor to create a shell script.
- So you can embed your own shell (/bin/sh) code in your custom tests.
- HOWEVER:
  - You can't just write bash code and expect everything to work!
  - Since the point of the `./configure` script is to run anywhere, you need to write shell code that can run on any Unix system's shell.
  - Lowest-common-denominator scripting.

Custom libraries of tests

- If you need to execute your own tests, you can write autoconf functions.
  - `AC_DEFUN` defines new functions.
- Custom functions can be embedded into a custom file.
  - `aclocal.m4`: project-specific custom functions.
  - `aclocal.m4`: system-wide custom functions.

Other parts of the GNU build environment

- `automake`:
  - Automates creation of `Makefile.in`.
  - Automatically supports `make clean`, `make install`, building outside the source directory, creating tar.gz distributions, etc.
  - Good for simple projects, not very flexible for complex projects.
- `libtool`:
  - Creating shared libraries, and dynamically-loadable libraries, is wildly different on all the platforms that support them.
  - Libtool is a shell script that encapsulates the knowledge of how to do this, how to set path paths automatically, and so forth.