Introduction à la configuration automatique

Christian Rinderknecht Yann Régis-Gianas

4 février 2015

Table des matières

1 Introduction

2 Toto

Il existe des recommandations et un ensemble d'utilitaires GNU dont le but est d'aider au développement, à la distribution et à la maintenance de logiciels portables en mode source. En particulier, l'accent est mis sur la configuration automatique avant la compilation. Ce sont les Autotools, qui correspondent aux applications suivantes.

- autoconf produit, à partir d'un fichier de configuration `configure.ac`, contenant des appels de macros M4 et des portions de scripts *shell*, un script *shell* nommé configure et des fichiers de support supplémentaires.
- configure est produit par autoconf et son exécution produit un fichier Makefile à partir d'une description `Makefile.in` et des fichiers de support produits par autoconf.
- aclocal est un générateur de macros M4 en éventuel support du fichier de configuration `configure.ac`, pour des cas spécifiques comme l'emploi conjoint de automake.
- automake est un outil pour produire automatiquement un fichier `Makefile.in` à partir d'une description `Makefile.am` composée de macros M4 et de règles pour make, dans le but que `Makefile.in` inclue automatiquement des règles génériques et conformes aux standards GNU.
- autoheader crée, à partir de `configure.ac`, un fichier d'en-tête C contenant des directives de pré-compilation `#define` pour configure (si `configure.ac` appelle la macro `AC_CONFIG_HEADERS(`*file*`.h)` alors autoheader crée *file*`.h.in`).
- libtoolize fournit un moyen standard d'ajouter le support de libtool (un script pour le support de bibliothèques partagées) aux applications C.
- autopoint installe l'infrastructure de fichiers support à l'internationalisation (système GNU Gettext).
- autoreconf installe et exécute si besoin les différents outils GNU pour la construction d'applications (autoconf, autoheader, aclocal, automake,

libtoolize et autopoint) à chaque niveau de l'arborescence des sources. Cela peut être rendu nécessaire si Automake a été mis à jour sur le système ou si `configure.ac` a changé, par exemple, ou simplement pour créer une nouvelle arborescence.

- autoscan facilite la création et la maintenance du fichier `configure.ac`. Il examine les sources C pour déterminer d'éventuels problèmes de portabilité et analyse aussi l'éventuel `configure.ac` pour en vérifier la complétude et produit un fichier `configure.scan` qui sert de canevas pour un (éventuellement nouveau) `configure.ac`.
- autoupdate met à jour un fichier `configure.ac` qui fait appel à des macros M4 pour Autoconf dont le nom est obsolète.

Nous allons présenter quelques cas simples d'utilisation de quelques-uns de ces utilitaires.

3 Autoconf

Autoconf est un outil pour produire des scripts *shell* qui configurent automatiquement des distributions logicielles en mode source dans le but de les adapter aux nombreux types de systèmes Unix. Les scripts de configuration produits par Autoconf sont indépendants de ce dernier lors de leur exécution, donc les usagers n'ont pas besoin d'avoir Autoconf.

Les scripts de configuration produits par Autoconf ne nécessitent aucune intervention manuelle de la part de l'usager pour être exécutés ; ils n'ont normalement même pas besoin d'un argument spécifiant le type du système. Au lieu de cela, ils testent eux-mêmes la présence des caractéristiques et outils dont la distribution a besoin. (Avant chaque vérification, ils affichent un message sur une ligne disant ce qu'ils testent.) Ainsi ils gèrent bien les systèmes hybridés à partir des variantes communes de Unix. En particulier il n'est pas besoin de maintenir des listes de caractéristiques de chaque distribution de chaque variante de Unix.

Pour chaque distribution pour laquelle Autoconf est employé, ce dernier outil crée un script de configuration à partir d'un script qui liste les caractéristiques et outils système dont la distribution a besoin ou peut utiliser. Après que le script *shell* qui reconnaît et réagit à une caractéristique système est écrit, Autoconf lui permet d'être partagé par de nombreuses distributions qui peuvent (ou doivent) utiliser cette caractéristique. Ainsi, si le script *shell* nécessite ultérieurement des ajustements, il n'est modifié qu'en un seul lieu et tous les scripts de configuration peuvent être régénérés automatiquement pour prendre en compte la mise à jour.

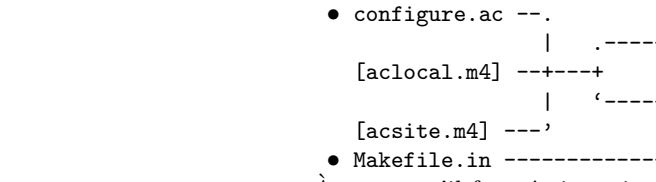
Le script *shell* de configuration produit par Autoconf est nommé par convention configure. Quand il est exécuté, celui-ci crée plusieurs fichiers dans lesquels les paramètres génériques de configuration sont remplacés par les valeurs appropriées. Ces fichiers sont :

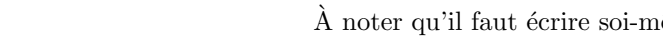
- un ou plusieurs makefiles, typiquement dans chaque sous-répertoire de la distribution ;
- un ensemble de fichiers définis par l'utilisateur dans lesquels contenant des variables à substituer ;

- optionnellement, un fichier d'en-tête C dont le nom est configurable, contenant des directives de précompilation `#define` ;
- un script *shell* nommé `config.status` qui, quand il sera exécuté, recréera les fichiers précédents ;
- un script *shell* optionnel nommé `config.cache` (crée par la commande `configure --config-cache`) qui sauvegarde le résultat des tests effectués par configure pour être partagés avec d'autres scripts configure ou entre différentes exécutions du même ;
- un fichier `config.log` contenant les messages émis par d'éventuels compilateurs dans le but d'aider au déverminage si configure commet une erreur.

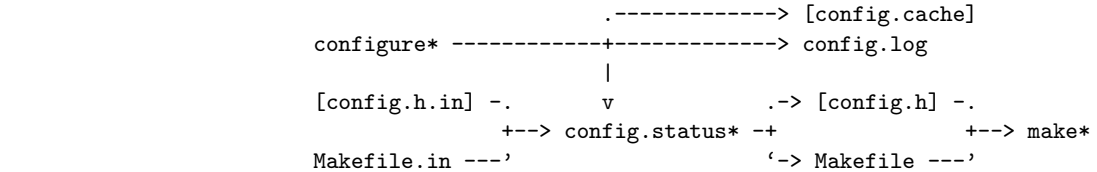
Donc, pour créer un script configure avec Autoconf il faut écrire un fichier d'entrée `configure.ac` et lancer autoconf. Si on veut écrire ses propres tests (sous forme de macros M4) pour compléter ceux proposés par défaut par Autoconf, il faut les mettre dans deux fichiers `aclocal.m4` et `acsite.m4`. Si on a besoin d'un fichier d'en-tête C (souvent nommé `config.h`) on peut alors employer Autoheader et l'on distribue alors sa spécification (souvent `config.h.in`).

Voici un diagramme tiré *verbatim* du manuel de Autoconf, qui montre comment les fichiers qui peuvent être utiles à la configuration sont produits. Les programmes qui sont exécutés sont suffixés par `*`. Les fichiers optionnels apparaissent entre crochets. À noter aussi que autoconf et autoheader lisent aussi les macros prédéfinies de Autoconf dans `autoconf.m4` (pré-installé).

- Fichiers utilisés pour préparer la distribution du logiciel (côté distributeur) :
  - Si l'on ne réutilise pas un `configure.ac`, on peut se faire aider pour en créer un neuf à partir des sources :
    sources $\longrightarrow$ autoscan* $\longrightarrow$ `configure.scan` $\longrightarrow$ `configure.ac`
  - 

  - 

  À noter qu'il faut écrire soi-même la description `Makefile.in` du makefile pour Autoconf. Nous verrons plus loin la différence avec l'emploi de Automake.
- Fichiers utilisés pour la configuration de la distribution (côté usager) :



3.1 Pour les projets en Objective Caml

3.1.1 Le fichier `configure.ac`

Voici un fichier configure.ac pour les petits projets en Objective Caml.

```
# -*-autoconf-*-

# Autoconfiguration for Objective Caml projects
# (c) 2003 Christian Rinderknecht

# Process this file with 'autoconf' to produce a configure script.
[...]
```

La première ligne est une méta-information pour Emacs et les suivantes informent sur la nature du script, son auteur et comment s'en servir.

```
[...]
# ------------------------------------------------------------------
# General settings
#
AC_INIT(Fun Calc,1.0,Christian.Rinderknecht@devinci.fr)
AC_PREREQ(2.53)
AC_CONFIG_SRCDIR(main.ml)

ac_version=$(autoconf -V | grep GNU)

echo "Autoconfiguration of $PACKAGE_STRING"
echo "(c) 2003 Christian Rinderknecht"
echo
echo "This is $ac_version"
[...]
```

Certaines macros [1] font l'hypothèse que d'autres macros ont été appelées précédemment. De plus, toute spécification `configure.ac` doit contenir un appel à la macro `AC_INIT` avant de procéder aux tests et doit appeler `AC_OUTPUT` à la fin. Le premier argument de `AC_INIT` est le nom du paquetage (*package*), ici Fun Calc, et le second son numéro de version, ici 1.0. Cette macro peut prendre deux arguments supplémentaires : l'adresse électronique du mainteneur du paquetage (à contacter en cas de problèmes) et le nom de l'archive `tar`. Par défaut, le nom de l'archive sera celui du paquetage sans le préfixe `GNU`, mis en minuscule et dont tous les caractères autres qu'alphanumériques et tirets-bas sont changés en trait d'union (donc ici le nom de base de l'archive sera `fun-calc`). Une fois qu'on aura exécuté autoconf, on pourra récupérer une partie de ces informations en exécutant

```
~/TP-Autotools/Projet/OCaml$ ./configure --version
Fun Calc configure 1.0
generated by GNU Autoconf 2.57
[...]
```

1. Les macros M4 de Autoconf sont toutes préfixées par « `AC_` ».

```
~/TP-Autotools/Projet/OCaml$ ./configure --help
'configure' configures Fun Calc 1.0 to adapt to many kinds of
systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...
[...]
Report bugs to <Christian.Rinderknecht@devinci.fr>.
```

On remarque en passant qu'il est de bon aloi de toujours appeler configure en forçant le chemin d'accès (`./configure`) pour éviter de fâcheux malentendus dont on se passe volontiers durant la phase de configuration.

On peut référencer ces arguments dans la spécification `configure.ac` par le biais des variables *shell* prédéfinies, en suivant l'ordre d'écriture, `PACKAGE_NAME`, `PACKAGE_VERSION`, `PACKAGE_BUGREPORT` et `PACKAGE_TARNAME`. On lit aussi dans notre `configure.ac` (première commande `echo`) l'emploi de la variable `PACKAGE_STRING` qui vaut la concaténation de `PACKAGE_NAME` et de `PACKAGE_VERSION`.

Ensuite (toujours après `AC_INIT`) nous appelons la macro `AC_PREREQ` avec le numéro de version minimal de Autoconf dont le script a besoin, ici 2.53. Si nous avions exigé `AC_PREREQ(2.63)` alors que nous utilisons la version 2.57 de Autoconf, voici ce qui se passe :

```
~/TP-Autotools/Projet/OCaml$ autoconf
configure.ac:10: error: Autoconf version 2.63 or higher is required
configure.ac:10: the top level
autom4te: /usr/bin/m4 failed with exit status: 1
```

L'appel de macro suivant est `AC_CONFIG_SRCDIR(main.ml)`. L'unique argument doit être un fichier présent dans le répertoire des sources. Le script configure vérifiera alors l'existence de ce fichier. Puisqu'il faudra donner en argument à configure le répertoire des sources avec l'option `--srcdir`, cet argument servira donc à détecter une éventuelle erreur avec cette option. Par exemple, si nous saisissons

```
~/TP-Autotools/Projet/OCaml$ autoconf
~/TP-Autotools/Projet/OCaml$ ./configure --srcdir=toto
configure: error: cannot find sources (ast.ml) in toto
```

Au passage, la variable `srcdir` est automatiquement initialisée par configure avec la valeur de l'option `--srcdir` (s'il n'a en a pas, la valeur est par défaut `.`). Ensuite nous déterminons la version courante de Autoconf pour l'afficher. Pour cela nous lançons autoconf avec l'option `-V`, ce qui donne par exemple :

```
~/TP-Autotools/Projet/OCaml$ autoconf -V
autoconf (GNU Autoconf) 2.53
Written by David J. MacKenzie and Akim Demaille.

Copyright 2002 Free Software Foundation, Inc.
```

```
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

Nous récupérons la ligne contenant le numéro de version à l'aide de grep et nous l'affichons. L'exécution de la portion de configure correspondant à cela donne :

```
~/TP-Autotools/Projet/OCaml$ ./configure
Generic autoconfiguration for Objective Caml projects
(c) 2003 Christian Rinderknecht

This is autoconf (GNU Autoconf) 2.53
[...]
```

Voici la suite de la spécification dans `configure.ac` :

```
[...]
# ------------------------------------------------------------------
# Checking for files named core, core.* and *.core.
#
# These files are deleted by the configure script (feature of
# autoconf), but core.* may be not a system core file. Hence this
# section backs up such files and prompts the user to check them.
#
core_files=$(ls core *.core core.* 2> /dev/null)
maybenot_core=$(ls core.* 2> /dev/null)

if test -n "$core_files";
then
  echo ------------------------------------------------------------
  if test $(echo $core_files | wc --words) = 1;
  then
    echo "File $core_files is going to be deleted!"
  else
    echo "Files $core_files are going to be deleted!"
  fi
  if test -n "$maybenot_core";
  then
    echo n "  => Backup of $maybenot_core in \
            BACKUP-$maybenot_core..."
    cp $maybenot_core BACKUP-$maybenot_core
    echo " done."
    echo "  => Rename or delete it and rerun autoconf \
          and configure."
  fi
fi
[...]
```

Autoconf a pour particularité de supprimer d'office tout fichier dont le nom vérifie les motifs *shell* `core *.core` et `core.*`. L'hypothèse est que ces fichiers

ont été produit par un plantage de l'application construite et donc qu'ils sont inutiles (et encombrants). Mais il se pourrait que ce ne soit pas le cas, en particulier si le nom de fichier a la forme `core.*`, par exemple `core.ml`. Le but de cette portion de script est d'éviter qu'un tel fichier soit perdu en le renommant et en informant l'usager.

Voyons maintenant comment spécifier la recherche des compilateurs O'Caml.

```
[...]
echo ------------------------------------------------------------
echo Compilers

# Checking for ocaml bytecode compiler
#
AC_PATH_PROG(OCAMLC,ocamlc,none)
AC_PATH_PROG(OCAMLC_OPT,ocamlc.opt,none)
[...]
```

La macro M4 nommée `AC_PATH_PROG` prend deux arguments obligatoires et deux optionnels. Son but, lorsqu'elle sera expansée dans configure, est de rechercher parmi un ensemble de chemins prédéfinis (et optionnellement passés en argument) un programme. Si ce dernier est trouvé, une variable *shell* donnée est initialisée avec le nom du programme préfixé par son chemin d'accès complet. Sinon la variable est vide ou alors prend une valeur passée en argument de la macro. Ainsi, dans notre premier exemple, la macro initialisera une variable `OCAMLC` en partant à la recherche du compilateur ocamlc. S'il n'est pas trouvé, la variable prendra la valeur `none`, sinon la valeur sera le chemin d'accès complet au compilateur. Les chemins de recherche prédéfinis sont ceux indiqués par la variable d'environnement `PATH`. L'exécution de la portion correspondante dans configure donne ici :

```
~/TP-Autotools/Projet/OCaml$ ./configure
[...]
------------------------------------------------------------------
Compilers
checking for ocamlc... /home/der_gi/rinderkn/bin/ocamlc
checking for ocamlc.opt... /home/der_gi/rinderkn/bin/ocamlc.opt
[...]
```

La suite de `configure.ac` est :

```
[...]
if test "$OCAMLC_OPT" = "none";
then
  if test "$OCAMLC" != "none";
  then
    OCAMLC_VER=‘$OCAMLC -version‘
    echo "  => ocamlc" v$OCAMLC_VER \
         "is the default compiler to byte-code"
```

```
  fi
else
  OCAMLC=$OCAMLC_OPT
  OCAMLC_VER=‘$OCAMLC -version‘
  echo "  => ocamlc.opt" v$OCAMLC_VER \
       "is the default compiler to byte-code"
fi
[...]
```

Si les variantes du compilateur (vers le byte-code) en byte-code et en code natif ont été trouvés [2], nous retenons celle en code natif car elle est plus rapide et, si besoin, nous initialisons avec la variable `OCAMLC`. Puis la version du compilateur retenu est déterminée, stockée dans `OCAMLC_VER` et affichée pour information. Cela donne par exemple :

```
~/TP-Autotools/Projet/OCaml$ ./configure
[...]
  => ocamlc.opt v3.07+2 is the default compiler to byte-code
[...]
```

La suite de `configure.ac` porte sur la recherche des compilateurs O'Caml vers du code natif et possède la même structure :

```
[...]
# Checking for ocaml native-code compiler
#
AC_PATH_PROG(OCAMLOPT,ocamlopt,none)
AC_PATH_PROG(OCAMLOPT_OPT,ocamlopt.opt,none)

if test "$OCAMLOPT_OPT" = "none";
then
  if test "$OCAMLOPT" != "none";
  then
    OCAMLOPT_VER=‘$OCAMLOPT -version‘
    echo "  => ocamlopt" v$OCAMLOPT_VER \
         "is the default compiler to native-code"
  fi
else
  OCAMLOPT=$OCAMLOPT_OPT
  OCAMLOPT_VER=‘$OCAMLOPT -version‘
  echo "  => ocamlopt.opt" v$OCAMLOPT_VER \
       "is the default compiler to native-code"
fi

# Checking consistency between compilers version numbers
#
```

2. Il existe en effet deux variantes du compilateur O'Caml vers du byte-code pour le programmeur : une composée de byte-code et une de code natif. Cela vient du fait que le compilateur O'Caml est autogénéré (ou *bootstrapped* en anglais), c'est-à-dire qu'il est écrit en O'Caml lui-même.

```
if test "$OCAMLC_VER" != "$OCAMLOPT";
then
  echo "  ** Warning: Version numbers of native" \
       " and byte-code compilers differ."
fi
[...]
```

On notera la vérification d'identité entre les versions des compilateurs natif et byte-code. Si les numéros sont différents un avertissement est alors émis. L'exécution de cette portion de script donne par exemple :

```
~/TP-Autotools/Projet/OCaml$ ./configure
[...]
checking for ocamlopt... /home/der_gi/rinderkn/bin/ocamlopt
checking for ocamlopt.opt... /home/der_gi/rinderkn/bin/ocamlopt.opt
  => ocamlopt.opt v3.07+2 is the default compiler to native-code
[...]
```

La portion qui suit dans `configure.ac` est :

```
[...]
echo ------------------------------------------------------------
echo Preprocessors

AC_PATH_PROG(CAMLP4,camlp4,none)

if test "$CAMLP4" != "none";
then
  CAMLP4_VER=‘$CAMLP4 -v 2>&1 \
              | sed -n 's|.*version* *\(.*\)$|\1|p' ‘
  echo "  => camlp4" v$CAMLP4_VER "is the default preprocessor"
  if test "$OCAMLC_VER" != "$CAMLP4_VER";
  then
   echo "  ** Warning: Version numbers of ocamlc and camlp4 differ."
  fi
fi

AC_PATH_PROG(CAMLP4O,camlp4o,none)
[...]
```

Son but est de localiser deux variantes du préprocesseur pour O'Caml (semblable à `cpp` pour les projets en langage `C`) et d'afficher son numéro de version. si celui-ci est différent de celui du compilateur O'Caml, on affiche un message d'avertissement. La suite est :

```
[...]
echo ------------------------------------------------------------
echo Dependences finders

AC_PATH_PROG(OCAMLDEP,ocamldep,none)
```

```
AC_PATH_PROG(OCAMLDEP_OPT,ocamldep.opt,none)

if test "$OCAMLDEP_OPT" = "none";
then
  if test "$OCAMLDEP" != "none";
  then
    echo "  => ocamldep is the default dependences finder"
  fi
else
  OCAMLDEP=$OCAMLDEP_OPT
  echo "  => ocamldep.opt is the default dependences finder"
fi
[...]
```

Son objectif est de localiser deux variantes de ocamldep, l'outil de recherche de dépendances entre modules fichiers O'Caml (semblable à `gcc -MM`). La suite de `configure.ac` est :

```
[...]
echo ------------------------------------------------------------
echo Lexer generators

AC_PATH_PROG(OCAMLLEX,ocamllex,none)
AC_PATH_PROG(OCAMLLEX_OPT,ocamllex.opt,none)

if test "$OCAMLLEX_OPT" = "none";
then
  if test "$OCAMLLEX" != "none";
  then
    echo "  => ocamllex is the default lexer generator"
  fi
else
  OCAMLLEX=$OCAMLLEX_OPT
  echo "  => ocamllex.opt is the default lexer generator"
fi
[...]
```

Cette partie est dédiée à la recherche des variantes du générateur d'analyseurs lexicaux ocamllex. La suite de `configure.ac` est :

```
[...]
echo ------------------------------------------------------------
echo Parser generator

AC_PATH_PROG(OCAMLYACC,ocamlyacc,none)
[...]
```

Cette partie recherche le générateur d'analyseurs syntaxiques ocamlyacc. Elle est plus simple car il n'existe qu'en code natif donc il n'y a pas de choix éventuel à faire entre deux variantes. La suite de `configure.ac` est :

```
[...]
echo ------------------------------------------------------------
echo Directories and paths

if test "$OCAMLC" != "none";
then
  OCAMLLIB=`$OCAMLC -where`
  echo "  => The standard library path is" $OCAMLLIB
fi
echo "  => The source directory is" \'$srcdir\'
[...]
```

Dans cette section de script, si un compilateur O'Caml a été localisé nous l'interrogeons sur le lieu de la bibliothèque standard et nous en informons l'usager. Au passage nous initialisons la variable `OCAMLIB` avec ce répertoire. Nous rappelons aussi le répertoire où se trouvent les fichiers sources en affichant le contenu de la variable `srcdir` (qui a été initialisée par l'option `--srcdir` ou vaut par défaut `.`). Ensuite nous trouvons dans `configure.ac` :

```
[...]
# ------------------------------------------------------------------
# Explicit variable substitutions
#
AC_SUBST(OCAMLC_VER)
AC_SUBST(OCAMLC)
AC_SUBST(OCAMLOPT_VER)
AC_SUBST(OCAMLOPT)
AC_SUBST(OCAMLLIB)
AC_SUBST(CAMLP4)
AC_SUBST(OCAMLDEP)
AC_SUBST(OCAMLLEX)
AC_SUBST(srcdir)
AC_SUBST(ac_version)
[...]
```

Cette section est différente des précédentes et très importante. Elle emploie la macro M4 nommée `AC_SUBST` qui prend en argument une variable du script (idéalement initialisée par celui-ci en fonction des résultats des tests effectués) et qui déclare la substitution textuelle future par sa valeur dans le fichier *fichier*`.in`, où *fichier* est l'argument de la macro `AC_CONFIG_FILES` (ci-après). Typiquement, ce fichier dans lequel sont substituées les variables est `Makefile.in` et configure produit à partir de lui le fichier `Makefile`. La fin de `configure.ac` dit :

```
[...]
echo ------------------------------------------------------------
AC_CONFIG_FILES(Makefile)
AC_OUTPUT
chmod a-w Makefile
```

C'est ici qu'on appelle donc `AC_CONFIG_FILES`. La macro `AC_OUTPUT` exécute les substitutions et la production de `Makefile` (ici). La dernière commande assure que le makefile produit n'est pas modifiable, car, en théorie, on ne doit pas modifier soi-même un fichier produit automatiquement. À l'écran on verrait :

```
~/TP-Autotools/Projet/OCaml$ ./configure
[...]
------------------------------------------------------------------
configure: creating ./config.status
config.status: creating Makefile
```

En réalité, comme le suggère l'affichage, configure produit un autre script nommé config.status qui, lui, fabrique *in fine* `Makefile` à partir du fichier `Makefile.in` en réalisant les substitutions de variables spécifiées dans le fichier `configure.ac`.

3.1.2 Le fichier `Makefile.in`

Dans le plus simple des cas, la différence entre un fichier `Makefile.in` et un `Makefile` tient au fait que l'on n'écrit pas directement dans `Makefile` les chemins vers les programmes qui sont utilisés. Au lieu de cela, on écrit un `Makefile.in` qui correspond à un `Makefile` habituel mais où les variables définissant ces chemins sont destinées à être textuellement substituées par une valeur correcte pour la plateforme par le truchement de configure (plus précisément config.status). Ces variables sont alors notées entre arrobes [3] dans `Makefile.in`.

Le fichier `Makefile.in` dans l'archive, associé au `configure.ac` présenté précédemment, commence ainsi :

```
# -*-makefile-*-

# Makefile for an Objective Caml project

# ------------------------------------------------------------------
# General settings
#
VPATH := src
.PHONY: clean

SRCDIR := @srcdir@
[...]
```

La nouveauté est donc la variable `SRCDIR` dont la valeur est textuellement définie par config.status, *via* configure, en substituant `@srcdir@` par la valeur de la variable `srcdir` définie dans `configure.ac`. En conclusion, le fichier `Makefile.in` ne présente aucune particularité si ce n'est la définition de variables Make à partir de « variables Autoconf ».

3. Il serait plus correct d'un point de vue étymologique de dire *à rond bas-de-casse* comme le font les typographes pour désigner le symbole @ (prononcé *ad* en latin).

3.2 Pour les projets en C/C++

Les langages C et C++ posent beaucoup de problèmes de portabilité car de nombreux compilateurs ont été développés depuis les années 70, chacun suivant sa route en fonction des ses objectifs et de son système d'exploitation. Malgré les normes ANSI pour le C, ISO pour le C++et POSIX pour les appels systèmes, certaines fonctionnalités sont donc soit absentes, soit mal implémentées. Pour gagner en portabilité, il convient donc de :

- S'informer sur l'environnement de compilation ;
- Détecter les problèmes ;
- Adapter la compilation.

3.2.1 Configuration standard pour un programme C/C++

Les informations à collecter pour compiler un programme C/C++sont au minimum le compilateur et les outils de création de bibliothèque. Considérons donc le fichier de configuration minimal suivant :

```
# Initialisation d'Autoconf.
AC_INIT(configure.ac)

# Quel est le compilateur C par défaut ?
AC_PROG_CC

# Pas de fichier de configuration ici.
AC_CONFIG_FILES([])
AC_OUTPUT
```

Après exécution d'Autoconf, puis du configure généré, on obtient :

```
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
configure: creating ./config.status
```

On remarque que le premier compilateur recherché est gcc, qu'il est bien détecté sur le système. Ensuite, configure effectue quelques tests standards de manière à définir ses variables de configuration.

Pour tester l'existence d'un compilateur C++, on utilise la macro `AC_PROG_CXX` de façon similaire. Pour tester l'existence de l'utilitaire ranlib, on utilise `AC_PROG_RANLIB`.

Voici donc un fichier de configuration standard pour tester un environnement C/C++ :

```
# Initialisation d'Autoconf.
AC_INIT(configure.ac)
AC_PROG_CC
AC_PROG_CXX
AC_PROG_RANLIB
AC_CONFIG_FILES([])
AC_OUTPUT
```

Et l'exécution de configure donne :

```
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking for ranlib... ranlib
configure: creating ./config.status
```

Entre autres choses, ce configure définit automatiquement les variables `CC`, `CXX`, `CFLAGS`, `CPPFLAGS`, `CXXFLAGS`, `RANLIB`. Il est toujours possible de forcer la valeur de ces variables en suffixant la commande './configure' par une affectation de la forme 'VARIABLE=VALEUR'. Par exemple, si vous voulez utiliser la version 2.95 de gcc, qui n'est plus la version par défaut sur les systèmes GNU/Linux, voici la commande :

```
# ./configure CC=gcc-2.95 CFLAGS='-O3 -DNDEBUG'
```

3.2.2 Détection de problèmes et leurs conséquences

Autoconf nous donne plusieurs niveaux de granularité pour tester d'éventuels problèmes. Le principe général de fonctionnement, c'est de générer et d'essayer de compiler un petit bout de code qui teste précisément le point de portabilité en question. Tout d'abord, il faut définir le langage utilisé pour ces tests, c'est le rôle de la macro `AC_LANG (LANGUAGE)`. Généralement, les macros de tests suivent la forme : `NOM_MACRO(A_TESTER, [ACTION_A_FAIRE_SI_OK], [ACTION_SINON])`.

Les actions sont de deux types :

- **Appel de macros Autoconf** comme `AC_MSG_RESULT`, `AC_MSG_NOTICE`, `AC_MSG_ERROR` pour informer l'utilisateur, mais aussi toutes les macros de tests (on peut ainsi chaîner les tests).
- **Commande shell quelconque**

Pour récolter les informations et les injecter dans le programme, un header peut-être créé à la demande par la commande `AC_CONFIG_HEADER(FICHIER)`. Usuellement, on appelle ce fichier 'config.h'.

Le premier niveau de granularité consiste à tester l'existence de headers par la macro `AC_CHECK_HEADER`. Voici sa signature :

```
 - Macro: AC_CHECK_HEADER (HEADER-FILE, [ACTION-IF-FOUND],
          [ACTION-IF-NOT-FOUND], [INCLUDES = ‘default-includes’])
```

Automatiquement, cette macro définit une macro C `HAVE_NOM_DU_HEADER_H` dans le config.h si le header existe. Ainsi, on peut compiler conditionnellement certaines parties du programme C/C++en utilisant le préprocesseur :

```
#ifdef HAVE_MYHEADER_H
# include "my_header.h"
#else
  // The function my_foo is not present, it does not matter:
  //we just do as if it was here.
  void my_foo() { }
#endif // HAVE_MYHEADER_H
```

Cependant, on veut parfois un comportement plus strict, on peut alors utiliser une action d'autoconf :

```
AC_CHECK_HEADER(my_header.h, [], [AC_MSG_ERROR([my_header.h is not
                present, I cannot compile.])])
```

De manière analogue, on peut tester l'existence d'une bibliothèque grâce à la macro :

```
 - Macro: AC_CHECK_LIB (LIBRARY, FUNCTION [, ACTION-IF-FOUND [,
          ACTION-IF-NOT-FOUND [, OTHER-LIBRARIES]]])
```

La bibliothèque est spécifiée par le nom utilisé lors de la liaison par l'option '-l' du compilateur. Par exemple, si la bibliothèque est la libncurses (utilisée pour contrôler les terminaux UNIX), la commande de liaison est '-lncurses'. Donc, pour tester l'existence de la fonction `initscr` dans cette bibliothèque :

```
AC_CHECK_LIB(ncurses, initscr)
```

Si la bibliothèque est bien trouvée, une macro C `HAVE_LIBNCURSES` est définie et la commande -lncurses est rajoutée automatiquement à la commande d'édition des liens.

Enfin, on peut aussi générer soit-même ses propres tests pour vérifier la bonne compilation ou/et le bon comportement de certaines fonctions du système. Pour cela, Autoconf nous fourni les macros suivantes :

```
 - Macro: AC_COMPILE_IFELSE (INPUT, [ACTION-IF-FOUND],
          [ACTION-IF-NOT-FOUND])
 - Macro: AC_LINK_IFELSE (INPUT, [ACTION-IF-FOUND],
          [ACTION-IF-NOT-FOUND])
 - Macro: AC_RUN_IFELSE (INPUT, [ACTION-IF-FOUND],
          [ACTION-IF-NOT-FOUND], [ACTION-IF-CROSS-COMPILING])
```

La première macro permet de tester la bonne compilation d'un programme, par exemple, on peut tester si la fonction C 'printf' se compile bien :

```
AC_COMPILE_IFELSE(
[
   #include <stdio.h>
   int main()
   {
      (void)printf("%d%d%f", 10, 10, 10);
   }
],
[echo Ok, printf works.],
[echo Ko, printf is out of order.])
```

La seconde permet de tester si l'édition de lien se passe correctement :

```
AC_LINK_IFELSE(
[
  int getopt(int);

  int main()
  {
     (void)(getopt(1));
  }
],
[echo Ok, getopt is already present in the system C standard library.],
[echo Ko, we should use our getopt library.])
```

Cela ne veut pas dire que 'getopt' est bien utilisée ou qu'elle fonctionne correctement, la dernière macro permet de tester l'exécution du programme :

```
AC_RUN_IFELSE(
[
  int getopt(int);

  int main()
  {
     (void)(getopt(1));
  }
],
[echo GetOpt does support bad use !],
[echo Ok, getopt is strict.])
```

Pour conclure, sachez qu'il existe encore de nombreuses macros d'Autoconf pour effectuer ce type de tests, vous les trouverez dans la section `Existing Tests` et `Writing Tests` de la documentation d'Autoconf (`info autoconf`).

4 Automake pour les projets en C/C ++

Automake est un outil de génération de Makefile. Les Makefiles générés fournissent des facilités pour le développement, la maintenance et la distribution

de programmes. Comme expliqué précédemment, Automake fournit des fichiers `Makefile.in` qui sont des squelettes de Makefile qui attendent les valeurs de configuration calculé par configure.

Pour générer ses `Makefile.in`, Automake attend de l'utilisateur qu'il lui fournisse un ensemble de `Makefile.am`. Ces fichiers définissent de manière minimale les informations relatives à la compilation et l'installation du logiciel.

4.1 Exemple minimaliste

Créons un fichier foo.c contenant :

```
#include <stdio.h>

int main()
{
  (void)printf("Hello World !\n");
}
```

On commence par créer un fichier minimaliste de configuration, configure.ac :

```
AC_INIT(configure.ac)
AM_INIT_AUTOMAKE(foo, 0.1, [yourname@devinci.fr])
AM_CONFIG_HEADER(config.h)
AC_PROG_CC
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Ce programme est le plus simple qu'il soit : on veut juste le compiler et créer un exécutable du même nom. Un `Makefile.am` possible pour définir sa compilation est le suivant :

```
bin_PROGRAMS=foo
```

On définit un programme qui se nomme 'foo' et qui est de la famille des programmes binaires. Par défaut, Automake essaye de trouver un fichier foo.c. Si on voulait spécifier précisément l'ensemble des sources du programme, il suffirait d'utiliser la variable `SOURCES` (on pourrait alors spécifier plusieurs fichiers sources) :

```
bin_PROGRAMS=foo
foo_SOURCES=foo.cc
```

Il créé un fichier `Makefile.in` si on lance la commande :

```
aclocal
autoheader
automake -i -a -c --foreign
```

Vous pouvez regarder le fichier `Makefile.in`, en particulier, on remarque l'ensemble des commandes qui seront disponibles :

```
.PHONY: CTAGS GTAGS all all-am check check-am clean clean-binPROGRAMS \
   clean-generic ctags dist dist-all dist-gzip distcheck distclean \
   distclean-compile distclean-depend distclean-generic \
   distclean-hdr distclean-tags distcleancheck distdir \
   distuninstallcheck dvi dvi-am info info-am install install-am \
   install-binPROGRAMS install-data install-data-am install-exec \
   install-exec-am install-info install-info-am install-man \
   install-strip installcheck installcheck-am installdirs \
   maintainer-clean maintainer-clean-generic mostlyclean \
   mostlyclean-compile mostlyclean-generic pdf pdf-am ps ps-am \
   tags uninstall uninstall-am uninstall-binPROGRAMS \
   uninstall-info-am
```

Maintenant, nous pouvons lancer Autoconf pour obtenir notre configure et lancer une compilation :

```
./configure && make
```

Pour finir cette introduction, vous pouvez lancer la commande 'make distcheck' qui termine son exécution par :

```
========================================
foo-0.1.tar.gz is ready for distribution
========================================
```

Cette commande compile le programme, créé une tarball, simule son installation et lance sa batterie de tests si elle est présente.

4.2 Compilation et maintenance

4.2.1 Compilation d'une bibliothèque

Créons un répertoire `src` dans lequel nous définissons les fichiers suivants :

```
// display.h
void display_message();

// display.c

#include <stdlib.h>
#include <errno.h>
#include <curses.h>
#include <error.h>
#include <unistd.h>

void display_message()
{
  int i, x, y;
  char* message = "Hello World";
  WINDOW* win = initscr();
  if (win == NULL)
```

```
    error(EXIT_FAILURE, errno, "Initialization problem.");
  y=LINES/2;
  x=(COLS-strlen(message))/2;
  mvprintw(y,x,message);
  refresh();
  sleep(2);
  (void)clear();
  (void)refresh();
  (void)endwin();
}
```

Ces fichiers vont constituer une bibliothèque fournissant la fonction 'display_message' appelée par notre programme foo. Automake va nous créer cette bibliothèque :

```
noinst_LIBRARIES=libdisplay.a
libdisplay_a_SOURCES= display.c
noinst_HEADERS=display.h
```

On remplace le PROGRAMS par LIBRARIES et déclare cette bibliothèque comme appartenant à la famille des bibliothèques qui ne s'installent pas : elles sont juste utilisées pour compiler le programme foo. Si on voulait installer cette bibliothèque sur le système, il suffirait de changer `noinst` par `lib`. Ensuite, les sources sont définies comme pour un programme normal. On spécifie aussi l'existence de headers – qu'on ne désire pas installer non plus – grâce à la variable `HEADERS`.

Il reste à connecter ce répertoire et son `Makefile.am` au projet. Pour cela, deux choses à faire. D'abord, définir une variables `SUBDIRS` dans le `Makefile.am` de la racine pour lui permettre de rentrer dans le répertoire `src`. On rajoute donc dans le `Makefile.am` de la racine du projet :

```
SUBDIRS=src
```

Ensuite, il faut demander à configure de générer le Makefile du sous-répertoire. On modifie donc la ligne suivante dans `configure.ac` :

```
AC_CONFIG_FILES([Makefile] [src/Makefile])
```

On peut relancer les Autotools grâce à la commande `autoreconf`. Ensuite, lancer `make` à la racine permet de compiler la bibliothèque et le programme (qui ne l'utilise toujours pas).

4.2.2 Compilation d'un programme

Pour utiliser une bibliothèque définie dans le projet, il suffit d'augmenter la variable `foo_LDADD` par src/libdisplay.a :

```
foo_LDADD= src/libdisplay.a
```

Notre bibliothèque utilise la libncurses, on doit donc spécifier une option à l'éditeur de lien. Or, la libncurses est externe à notre projet donc son intégration dépend de la configuration, c'est à dire de configure. Il suffit donc de tester son existence dans le configure.ac par la macro `AC_CHECK_LIB` pour que l'option soit automatiquement définie par Autoconf.

4.2.3 Définir une batterie de tests

Automake permet de mettre en place un système de tests unitaires lancable à partir de make par la commande `make check`. Il suffit simplement de définir dans une variable `TESTS`, l'ensemble des programmes de tests et ensuite de déclarer pour chacun de ces `check_PROGRAMS` les sources utilisés pour les compiler. Le processus de test consiste à compiler ces programmes et à les exécuter successivement : un test réussit ssi son status de sortie est `EXIT_SUCCESS`.

Par exemple, on créé un répertoire 'tests' à la racine du projet. On définit un source 'test1.c' qui se content de faire 'exit(EXIT_SUCCESS)'. On définit alors le `Makefile.am` de la façon suivante :

```
TESTS=test1
check_PROGRAMS=test1
test1_SOURCES=test1.c
```

On met à jour le `SUBDIRS` du Makefile de la racine et on demande à configure de générer le nouveau Makefile. Ensuite, lancer 'make check' à la racine donne :

```
gcc -DHAVE\_CONFIG\_H -I. -I. -I..  -g -O2 -c `test -f 'test1.c' || echo
'./'`test1.c
gcc  -g -O2   -o test1  test1.o  -lncurses
PASS: test1
==================
All 1 tests passed
==================
```

On peut aussi définir des tests qui ne doivent pas réussir (pensez à un compilateur qui doit rejeter les programmes mal formés). Pour cela, on utilise la variable `XFAIL_TESTS`. Enfin, les programmes de tests peuvent aussi être des scripts shell par exemple.

4.3 Installation et distribution

4.3.1 Paramètres d'installation

Grâce à 'configure', on peut spécifier l'emplacement exact où on veut installer un logiciel : ces binaires, ses bibliothèques, ses documentations et ses fichiers annexes (demos ...). On trouve donc des options :

```
--bindir                -- specify: directory
--datadir               -- specify: directory
--includedir            -- specify: directory
```

```
--infodir               -- specify: directory
--libdir                -- specify: directory
--libexecdir            -- specify: directory
--localstatedir         -- specify: directory
--mandir                -- specify: directory
--oldincludedir         -- specify: directory
--prefix                -- specify: prefix directory
--program-prefix        -- specify: prefix directory
--sbindir               -- specify: directory
--sharedstatedir        -- specify: directory
--srcdir                -- specify: directory
--sysconfdir            -- specify: directory
```

Il y a deux types d'options : celles qui permettent de contrôler finement les répertoires et celles qui sont plus globables. `-prefix` est la plus intéressante car elle permet de spécifier une racine à partir d'où on peut installer les différents composants du programme. Par exemple, si le préfixe est '/usr', les bibliothèques seront installées dans '/usr/lib', les binaires dans '/usr/bin' ... etc.

configure définit donc un certain nombre de variables qui correspondent à ces répertoires : ($*bindir*, $*srcdir*,$*sbindir*,$*pkgdatadir*, $*datadir*...).

4.3.2 Définition de l'installation et de la distribution

D'une manière générale, on utilise un format de nommage des variables pour déterminer si l'objet qu'elle réfère doit être distribué et installé (en précisant où l'installer). Une variable se décompose donc en quatre parties :

- `dist/no_dist` : qui précise si l'objet doit être distribué ;
- `le répertoire d'installation/no_inst` : qui définit la cible d'installation ou qui supprime l'installation de l'objet ;
- `la famille de l'objet` : PROGRAMS, SCRIPTS, DATA, LIBRARIES, HEADERS...

Par exemple :

```
# Définit un programme à installer des /sbin
sbin_PROGRAMS=foo
# Définit un programme à ne pas installer
noinst_PROGRAMS=foo
# Définit un fichier source à ne pas distribuer:
nodist_SOURCES=foo.c
# Attention ici, le fichier sera installé dans $pkgdatadir/
# et non $pkgdatadir/html
pkgdata_DATA=html/foo.html
```

Pour voir le répertoire d'installation comme une racine, on utilise un modificateur 'nobase' :

```
# Le fichier sera bien installé dans $pkgdatadir/html
nobase_pkgdata_DATA=html/foo.html
```

On peut aussi créer ses propres répertoires :

```
htmldir=$pkgdatadir/html
html_DATA=html/foo.html
```

Enfin, lorsqu'un de vos fichiers doit juste être distribué dans la tarball et qu'il ne fait partie d'aucune des catégories citées, on utilise la variable `EXTRA_DIST` :

```
EXTRA_DIST=BUGS INSTALL
```

4.3.3 make dist/distcheck/distclean/install

Le but de ces commandes est d'automatiser la création de tarball. Elles déterminent les dépendances du programme, c'est-à-dire l'ensemble des informations nécessaires à sa compilation et les intègrent dans la tarball. Une tarball ne contient donc pas tout ce qui se trouve dans votre répertoire de développement, si vous voulez rajouter un fichier, il faut le spécifier à l'aide de la variable `EXTRA_DIST`.

'make dist' s'occupe seulement de créer la tarball sans la tester. 'make distcheck' lui, s'occupe en plus d'installer la tarball avec un préfixe local au répertoire courant, compile le logiciel et lance 'make check'. Si tout ce passe bien, il vous informe que votre tarball est bonne à distribuer. Enfin, 'make distclean' nettoie l'ensemble de l'arborescence de développement.

La tarball ainsi créé contient un configure qui générera un Makefile. La commande essentielle à fournir à vos utilisateurs est donc la suivante :

```
tar xvfz votre-tarball.tar.gz
./configure
make
make check
make install
```