

The Mercury Frequently Asked Questions List

Version 14.01.1

Fergus Henderson
Thomas Conway
Zoltan Somogyi

Copyright © 1995–1997,1999,2001–2014 The University of Melbourne.

Permission is granted to make and distribute verbatim copies of this FAQ list provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this FAQ list under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this FAQ list into another language, under the above conditions for modified versions.

Table of Contents

1	Problems during compilation and installation.	1
2	Common programming errors	1
3	Problems caused by unimplemented Mercury features.....	3
4	What to do when all else fails	3

1 Problems during compilation and installation.

1. *The compiler crashes during the “make install” phase*

Mercury’s use of GNU C extensions can cause problems with some versions of GCC, notably versions 4.2 and 4.3. Either use a different version of GCC, or configure Mercury so that it does not require any GNU C extensions. The following describes how to do the latter.

For versions of Mercury after 0.13.1, invoke the configure script with the option ‘`--with-llds-base-grade=none`’. This will cause the compiler to be built in the ‘`none.gc`’ grade, which does not use any GNU C extensions.

The configure script for Mercury version 0.13.1 (and before) does not support the above option. Instead create a file in the top-level directory of the unpacked source tree named ‘`Mmake.params`’ containing the line ‘`GRADE=none.gc`’.

After running the configure script, do “make” and “make install” as normal.

2. *The “make install” phase takes ages! What’s going on?*

Mercury supports many grades. A grade is a combination the target language and feature options. These features include garbage collection, debugging and profiling support. See “Grades and grade components” in the User’s Guide.

During the “make install” phase the newly built compiler is used to compile the standard library in each of the selected grades. The runtime system is also rebuilt for each grade.

To install fewer grades and reduce the time “make install” takes, see the configure script’s options, in particular ‘`--disable-most-grades`’. Alternatively, run ‘`make install LIBGRADES=<grades>`’ instead, where “<grades>” is a space-separated list of grades.

Users of Mac OS X should note that there appear to be performance problems with the linker on some versions of that operating system that can lead to excessive linking times with code generated by the Mercury compiler.

2 Common programming errors

1. *What does the error message “undefined symbol ‘.’/2” mean?*

You need to explicitly import the ‘`list`’ module

```
:- import_module list.
```

if your code uses lists. Similarly, if your code uses arithmetic operations, you will need to import the ‘`int`’ and possibly ‘`float`’ modules.

2. *Why doesn’t ‘`X <= 3`’ work?*

In Mercury, less-than-or-equal-to is written as ‘`<=`’ not as ‘`<=`’, which is used for reverse implication.

3. *I defined a type like this: ‘:- type number ---> int ; float.’ Why doesn’t this work?*

You tried to define a type that is an undiscriminated union of two types, which is not allowed by the Mercury type system. The declaration above defines an enumerated type with two constants, “int” and “float”. This is not what you want, but it is legal Mercury, which is why you don’t get an error message on the type declaration itself.

4. *I get a “scope error” in an if-then-else. I checked and both branches bind the same variables. What is the error?*

This error generally happens if you attempt bind non-local variables in the *condition* of the if-then-else. For example, the following code attempts to bind ‘Value’ in the call to ‘map.search’, but ‘Value’ occurs outside of the if-then-else — in particular, it occurs in the head of the clause.

```
:- pred map.search(map(K, V), K, V).
:- mode map.search(in, in, out) is semidet.

:- pred lookup(map(string, int), string, int).
:- mode lookup(in, in, out) is det.

lookup(Map, Key, Value) :-
    (if map.search(Map, Key, Value) then
        true
    else
        Value = -1
    ).
```

Binding non-local variables in the condition of an if-then-else is forbidden since it would be unsound; it would lead to inconsistent results. For example, ‘(X = 1 -> Y = 1 ; Y = 2), X = 2’ would fail, but ‘X = 2, (X = 1 -> Y = 1 ; Y = 2)’ would succeed — breaking one of the fundamental laws of logic, ‘(P, Q) <=> (Q, P)’.

Mode analysis therefore rejects such programs. (In certain rare circumstances, the compiler may report this as a “mode error” rather than a “scope error”.) The way to fix such errors is to avoid binding non-local variables in the *condition*, and instead bind them in the *then* part of the if-then-else. So in the above example, you should introduce a new local variable, which we will call ‘Value1’:

```
lookup(Map, Key, Value) :-
    ( map.search(Map, Key, Value1) ->
        Value = Value1
    ;
        Value = -1
    ).
```

5. *“I keep getting a link error ‘undefined symbol init_gc’. Why?”*

If you are using ‘`mmake`’ to recompile your program, and you are overriding the default grade (e.g. by setting ‘`GRADE=asm_fast`’ in your ‘`Mmake`’ file), you must make sure that you do ‘`mmake clean`’ every time you change grades.

3 Problems caused by unimplemented Mercury features

1. *How can I avoid getting a compile-time error when I try to fill in a partially instantiated data structure?*

At the moment, you can create a partially instantiated data structure, but you can’t fill in the holes. The reason is that the code that does the filling in must temporarily alias two variables together, and the current mode checker does not allow this. This limitation will go away in the future.

2. *I’m getting an error from the C compiler:*

```
foo.c:45: initializer is not computable at load time
```

You’re using an old version of ‘`gcc`’. Check that the version of ‘`gcc`’ in your `PATH` is version 2.6.3 or later. Mercury does not (at the current time) support versions of `gcc` earlier than 2.6.3. (Using the ‘`--no-static-ground-terms`’ option may also solve this problem, but results in less efficient code.)

4 What to do when all else fails

1. *I’m getting an error message that I don’t understand. What can I do?*

Try compiling with the ‘`-E`’ (‘`--verbose-error-messages`’) option. This option causes the compiler to give even more verbose descriptions than usual.

2. *I followed the instructions in the user’s guide, but it still didn’t work. What do I do next?*

Send email to bugs@mercurylang.org, and we’ll try to solve your problem.