Mercury (programming language)

Mercury is a functional logic programming language made for real-world uses. The first version was developed at the University of Melbourne, Computer Science department, by Fergus Henderson, Thomas Conway, and Zoltan Somogyi, under Somogyi's supervision, and released on April 8, 1995.

Mercury is a purely declarative logic programming language. It is related to both Prolog and Haskell.[1] It features a strong, static, polymorphic type system, and a strong mode and determinism system.

The official implementation, the Melbourne Mercury Compiler, is available for most Unix and Unix-like platforms, including Linux, macOS, and for Windows.

Contents

Overview

Mercury is based on the logic programming language Prolog. It has the same syntax and the same basic concepts such as the selective linear definite clause resolution (SLD) algorithm. It can be viewed as a pure subset of Prolog with strong types and modes. As such, it is often compared to its predecessor in features and run-time efficiency.

The language is designed using software engineering principles. Unlike the original implementations of Prolog, it has a separate compilation phase, rather than being directly interpreted. This

Mercury

Paradigm	Logic, functional, object-oriented
Designed by	Zoltan Somogyi
Developer	University of Melbourne
First appeared	April 8, 1995
Stable release	20.06.1 / November 3, 2020
Typing discipline	Strong, static, polymorphic
Implementation language	Mercury
Platform	IA-32, x86-64, ARM, Sparc64, Java, CLI, Erlang/OTP
OS	Cross-platform: Unix, Linux, macOS, Solaris, FreeBSD, OpenBSD, Windows, Android
License	GPL compiler, LGPL standard library
Filename extensions	.m
Website	www.mercurylang.org (http://www.mercurylang.org)
Major implementations	

allows a much wider range of errors to be detected before running a program. It features a strict static type and mode system[1] and a module system.

Melbourne Mercury Compiler
Influenced by
Prolog, Hope, Haskell

By using information obtained at compile time (such as type and mode), programs written in Mercury typically perform significantly faster than equivalent programs written in Prolog.[2][3] Its authors claim that Mercury is the fastest logic language in the world, by a wide margin.[1]

Mercury is a purely declarative language, unlike Prolog, since it lacks *extra-logical* Prolog statements such as ! (cut) and imperative input/output (I/O). This enables advanced static program analysis and program optimization, including compile-time garbage collection,[4] but it can make certain programming constructs (such as a switch over a number of options, with a default) harder to express. (While Mercury does allow impure functionality, this serves mainly as a way to call foreign language code. All impure code must be marked explicitly.) Operations which would typically be impure (such as input/output) are expressed using pure constructs in Mercury using linear types, by threading a dummy *world* value through all relevant code.

Notable programs written in Mercury include the Mercury compiler and the Prince XML formatter. Software company Mission Critical IT has also been using Mercury since 2000 to develop enterprise applications and its Ontology-Driven software development platform, ODASE.[5]

Back-ends

Mercury has several back-ends, which enable compiling Mercury code into several languages, including:

Production level

- Low-level C for GNU Compiler Collection (GCC), the original Mercury back-end
- High-level C
- Java
- C#
- Erlang

Past

- Assembly language via the GCC back-end
- Aditi, a deductive database system also developed at the University of Melbourne. Mercury-0.12.2 is the last version to support Aditi.
- Common Intermediate Language (CIL) for the .NET Framework

Mercury also features a foreign language interface, allowing code in other languages (depending on the chosen back-end) to be linked with Mercury code. The following foreign languages are possible:

Back-end	Foreign language(s)
C (both levels)	C
Java	Java
Erlang	Erlang
IL	Common Intermediate Language (CIL) or C#

Other languages can then be interfaced to by calling them from these languages. However, this means that foreign language code may need to be written several times for the different backends, otherwise portability between backends will be lost.

The most commonly used back-end is the original low-level C back-end.

Examples

Hello World:

```
:- module hello.
:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.

:- implementation.
main(!IO) :-
   io.write_string("Hello, World!\n", !IO).
```

Calculating the 10th Fibonacci number (in the most obvious way):[6]

```
:- module fib.
:- interface.
:- import_module io.
:- pred main(io::di, io::uo) is det.

:- implementation.
:- import_module int.

:- func fib(int) = int.
fib(N) = (if N =< 2 then 1 else fib(N - 1) + fib(N - 2)).

main(!IO) :-
       io.write_string("fib(10) = ", !IO),
       io.write_int(fib(10), !IO),
       io.nl(!IO).
       % Could instead use io.format("fib(10) = %d\n", [i(fib(10))], !IO).
```

`!IO` is a "state variable", which is syntactic sugar for a pair of variables which are assigned concrete names at compilation; for example, the above is desugared to something like:

```
main(IO0, IO) :-
       io.write_string("fib(10) = ", IO0, IO1),
       io.write_int(fib(10), IO1, IO2),
       io.nl(IO2, IO).
```

Release schedule

Releases are named according to the year and month of release. The current stable release is 20.06 (June 30, 2020). Prior releases were numbered 0.12, 0.13, etc., and the time between stable releases can be as long as 3 years.

There is often also a snapshot *release of the day* (ROTD) consisting of the latest features and bug fixes added to the last stable release.

IDE and editor support

- Developers provide support for Vim
- Flycheck library for Emacs
- A plugin is available for the Eclipse IDE
- A plugin is available for the NetBeans IDE

See also

- Curry, another functional logic language
- Alice, a dialect language of Standard ML
- Logtalk, language, an object-oriented extension of Prolog which compiles down to Prolog
- Oz/Mozart, a multiparadigm language
- Visual Prolog, language, a strongly typed object-oriented extension of Prolog, with a new syntax

References

1. The Mercury Project - Motivation (http://www.mercurylang.org/about/motivation.html)
2. The Mercury Project - Benchmarks (http://www.mercurylang.org/about/benchmarks.html)
3. Somogyi, Zoltan; Henderson, Fergus; Conway, Thomas (October–December 1996). "The execution algorithm of Mercury: an efficient purely declarative logic programming language" (http://www.mercurylang.org/documentation/papers.html#jlp). *Journal of Logic Programming*. Mercurylang.org. **29** (1–3): 17–64. CiteSeerX 10.1.1.46.9861 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.9861). doi:10.1016/S0743-1066(96)00068-4 (https://doi.org/10.1016%2FS0743-1066%2896%2900068-4). Retrieved 2008-08-30.
4. Mazur, Nancy (May 2004). *Compile-time garbage collection for the declarative language Mercury* (https://mercurylang.org/documentation/papers/CW2004_03_mazur.pdf) (PDF) (Thesis). Katholieke Universiteit Leuven.
5. Mission Critical IT (http://www.missioncriticalit.com/)
6. Adapted from Ralph Becket's Mercury tutorial (http://www.mercurylang.org/documentation/papers/book.pdf)

External links

- Official website (http://www.mercurylang.org)