Status of the Mercury system

Zoltan Somogyi
Fergus Henderson
Thomas Conway
Andrew Bromage
Tyson Dowd
David Jeffery
Peter Ross
Peter Schachte
Simon Taylor

http://www.cs.mu.oz.au/mercury

Department of Computer Science
University of Melbourne

Mercury

Mercury is a new general purpose programming language intended to support the creation and maintenance of large, reliable, efficient applications by teams of programmers.

- purely declarative
- strong type system
- strong mode system
- strong determinism system
- explicit uniqueness information
- parametric polymorphism
- higher order

Mercury is related to Prolog the way Ada is related to Basic. They are both based on logic, but programming in one does not really resemble programming in the other.

Type and mode inference

```
:- module nrev.
:- interface.
:- pred nrev(list(T), list(T)) is det.
:- mode nrev(in, out) is det.

:- implementation.

nrev([], []).
nrev([H | T], L) :-
    nrev(T, L0),
    append(L0, [H], L).

append([], L, L).
append([H | T], L, [H | NT]) :-
    append(T, L, NT).
```

Inferred:

```
:- pred append(list(T), list(T), list(T)).
:- mode append(in, in, out).
```

Predicate specialization

original:

```
test(A, B) :-
    map(add1, A, B).

add1(N, N + 1).

map(_Pred, [], []).
map(Pred, [X | Xs], [Y | Ys]) :-
    call(Pred, X, Y),
    map(Pred, Xs, Ys).
```

transformed:

```
test(A, B) :-
    map'(A, B).

map'([], []).
map'([X | Xs], [Y | Ys]) :-
    Y = X + 1,
    map'(Xs, Ys).
```

Lookup tables

```
:- type suit --->
	club ; diamond ; heart ; spade ; notrump.

:- pred value(suit::in, int::out) is det.

value(notrump, 40).
value(club, 20).
value(diamond, 20).
value(heart, 30).
value(spade, 30).

Define_entry(value_2_0);
	{
	static const Word mercury_const_1[] = {
		20, 20, 30, 30, 40 };
	r1 = field(mktag(0),
		mkword(mktag(0), mercury_const_1), r1);
	}
	proceed();
```

C interface

```
:- pragma(c_header_code, "
typedef struct mercury_file {
    FILE    *file;
    int     line_number;
} MercuryFile;
").

:- pragma(c_code,
    write_int(Stream::in, Val::in, IO0::di, IO::uo),
"{
    MercuryFile *stream;
    stream = (MercuryFile *) Stream;
    if (fprintf(stream->file, ""%d"", Val) < 0) {
        mercury_output_error(stream);
    }
    IO = IO0;
}").
```

Experience from our project

The Mercury compiler is written in Mercury itself. It is now over 100,000 lines of code in 150 modules.

Most of this was written by three people (1 f/t, 2 p/t) over two years. This implies productivity around 300 lines per week per person.

By now eleven people have worked on the compiler; mostly students, some only second years. They have found the language easy to learn and to use.

Most modules written by other people are easy to modify. Concurrent changes rarely interfere. Code accepted by the compiler works first time surprisingly often.

Performance

We have compared the performance of Mercury against the performance of other logic programming systems on both a set of ten small benchmarks, and a large benchmark (the Mercury compiler, 3.6 Mb of source).

system	speed small	speed large	size large
NU-Prolog	0.9	N/A	2.8 Mb
SICStus compact	1.0	1.0	6.0 Mb
SICStus fast	3.3	1.5	10.3 Mb
Mercury Boehm	4.2	5.6	2.3 Mb
Mercury no gc	16.5	N/A	N/A

The small benchmarks also show Mercury to be about ten times the speed of Quintus and wamcc, and almost double the speed of Aquarius.

Summary

- fast code
- compact code
- high reliability
- high productivity
- supports team programming
- supports maintenance
- interface to and from C
- portable
- free

http://www.cs.mu.oz.au/mercury