Mercury: an efficient, purely declarative logic programming language

Zoltan Somogyi
Fergus Henderson
Thomas Conway (presenter)

University of Melbourne, Australia

What is Mercury?

- a new general purpose programming language based on the paradigm of *purely* declarative logic programming

- intended to support the creation of large, reliable, efficient, real-world applications by teams of programmers

- currently being designed and implemented at the University of Melbourne, alpha release expected soon

Features of Mercury

- module system
- high level
- purely declarative
- strong polymorphic type system
- with higher-order predicates and closures
- strong mode system
- strong determinism system
- imperative update
- compile-time checking for all of the above
- very efficient implementation

```
:- pred valid(set(set(T)), bool, io, io).
:- mode valid(in, out, di, uo) is det.

valid(Colours, Result, S0, S) :-
  (
    some [Colour1,Colour2,Element] (
      set__member(Colour1, Colours),
      set__member(Colour2, Colours),
      Colour1 \= Colour2,
      set__member(Element, Colour1),
      set__member(Element, Colour2)
    )
  ->
    write_string("Invalid colour.\n", S0, S),
    Result = no
  ;
    S = S0,
    Result = yes
  ).
```

The Mercury compiler

We are writing the Mercury compiler in Mercury itself, using NU-Prolog for bootstrapping.

The front end puts several annotations on the internal form of the program. The code generator is relatively simple.

The representations on either side of the code generator are optimized independently.

The compiler generates portable C code, optionally using GNU C extensions. The resulting object code is quite dense and very fast.

Using Mercury

The compiler's 56,000 lines contain about 31K of code and 8K lines of declarations; the rest are comments and whitespace. This is good productivity for about two person-years of work.

The compiler strives to give precise and meaningful error messages. We have found these to be very effective during development. Modules that compile without errors often work first time.

We find it easy to modify each other's code. The declarations help us understand the code, and the module system limits the scope of changes.

Benchmarks

System	deriv	poly	crypt	queen	avg
SWI	1.0	1.0	1.0	1.0	1.0
NU	1.6	1.8	1.4	33.8	1.7
SICStus	6.8	7.9	6.0	30.8	6.8
Aquarius	15.8	18.9	7.6	26.1	19.0
Mercury	24.1	39.3	27.2	34.6	36.3

The bigger the program, the bigger Mercury's performance advantage over systems based on program analysis (e.g. Aquarius).

For many predicates we generate almost optimal code.

Advantages of Mercury

- Purely declarative
- High level
- Catches errors at compile time
- Supports programming-in-the-large
- Very efficient
- Free portable implementation (soon...)

Currently not yet implemented:

- Mode checking for higher-order preds and closures.
- The types `float` and `univ`.
- Mode analysis for some tricky cases with partially instantiated data structures.
- Imperative update.
- A user's guide and a language reference manual.
- ...

```
:- module hello.
:- interface.
:- import_module io.

:- pred main(io_state, io_state).
:- mode main(destr_in, unique_out) is det.

:- implementation.
:- import_module string.

main(S0, S) :-
        write_string("Hello, ", S0, S1),
        write_string("world\n", S1, S).
```