Termination analysis for Mercury

Chris Speirs
Zoltan Somogyi
Harald Sondergaard

crs@students.cs.mu.oz.au
{zs,harald}@cs.mu.oz.au
Department of Computer Science
University of Melbourne

Why termination analysis?

Some optimizations reorder goals. However, before transforming *p, q* to *q, p* the compiler ought to check that *q* is guaranteed to terminate.

If analysis cannot prove the termination of a predicate, the system should inform the programmer.

Even if the pinpointed predicate actually terminates, its termination will probably not be apparent to other programmers.

It is also likely to have a performance problem.

Relationships among argument sizes

```
qsort(In, Out) :-
    In => [Head | Tail],
    split(Head, Tail, Lo, Hi),
    qsort(Lo, SLo),
    qsort(Hi, SHi0),
    SHi <= [Head | SHi0],
    append(SLo, SHi, Out).
```

To prove that qsort cannot recurse forever, we must prove that both Lo and Hi are smaller than In.

We thus need to know about relationships among the sizes of the arguments of split.

Size can be measured in number of heap cells, or words in heap cells, or ...

Size inequalities

One per procedure, of the form

$|\text{output supplier inputs}| + \gamma_p \geq |\text{outputs}|$

We propagate these backwards through the clauses, updating the right side. We iterate until reaching a fixpoint.

```
split(Pivot, List, Lo, Hi) :-
        % list + γsplit
    List => [Head | Tail],
        % head + γsplit + 1
    Pivot > Head,
        % head + γsplit + 1
    split(Pivot, Tail, Lo0, Hi),
        % head + lo0 + hi + 1
    Lo <= [Head | Lo0].
        % lo + hi
```

Impact of output suppliers

```
qsort(In, Out) :-
        % head + in + 2 γqsort
    In => [Head | Tail],
        % head + tail + 2 γqsort + head + 1
    split(Head, Tail, Lo, Hi),
        % lo + hi + 2 γqsort + head + 1
    qsort(Lo, SLo),
        % slo + hi + γqsort + head + 1
    qsort(Hi, SHi0),
        % slo + shi0 + head + 1
    SHi <= [Head | SHi0],
        % slo + shi
    append(SLo, SHi, Out).
        % out
```

$\text{in} + \gamma_{\text{qsort}} \geq \text{in} + 2\,\gamma_{\text{qsort}} \quad \Rightarrow 0$

$\text{in} + \gamma_{\text{qsort}} \geq \text{head} + \text{in} + 2\,\gamma_{\text{qsort}} \quad \Rightarrow \infty$

Tracing inputs to recursive calls

To prove the absence of infinite recursion, we must prove

|head inputs| > |recursive inputs|

for *any* set of input argument positions. We use a fixpoint iteration, starting with all input arguments, to find a good set.

```
qsort(In, Out) :-
        % in > in - 1
    In => [Head | Tail],
        % in > tail
    split(Head, Tail, Lo, Hi),
        % in > lo
    qsort(Lo, SLo),
    ...
```

Conditional termination

Input predicates reduce the size of the state argument if they return an item, whereas output predicates leave the state size unchanged.

The answer *yes* from the analyzer means *the program terminates provided its inputs are finite*.

```
main(S0, S) :-
    io__read(Result, S0, S1),
    (
        Result = ok(Term),
        process(Term, S1, S2),
        main(S2, S)
    ;
        Result = eof,
        S = S1
    ).
```

Performance

On a large set of small tests, our system proved the termination of as many predicates as a recent algorithm by Lindenstrauss and Sagiv (47 out of 53, 89%).

Our system proved the termination of 72% of the procedures in the Mercury standard library and 55% of the procedures in the Mercury compiler.

The termination analysis stage accounted for an average of 1.8% of total compilation time. On an UltraSPARC 1/170, it usually took about one quarter of a second. It never took more than one second, even for files with 3000 lines.

Conclusion

Termination analysis is part of the Mercury compiler. It is fast enough to turn on all the time.

The stage 1 and stage 2 algorithms both include fixpoint computations for increased accuracy.

We handle higher-order calls and intermodule calls conservatively. We have a design for handling intermodule calls more accurately.

Termination analysis has pointed out a performance bug in tree234. Fixing the bug sped up the entire compiler by 35%.