Divide-and-query and subterm dependency tracking in the Mercury declarative debugger

Ian MacLarty, Zoltan Somogyi and Mark Brown
University of Melbourne

Declarative debugging

- Program execution represented by tree.
- Each node in the tree corresponds to a procedure/function call in the program.
- The children of each node are the child calls to procedures in the body of the parent procedure.
- Bug = correct children + erroneous parent.
- Eliminate subtrees based on knowledge gained from the user.

Advantages

- Upper bound on effort of finding a bug.
- Much less to remember.
- Debugger directs bugs search.

Not widely used

- Only works well for purely declarative programs.
- Did not previously scale to large search spaces.
- Questions may be difficult to answer.

Mercury procedural debugger

- Declarative debugger built on top of procedural debugger.

- *Interface* events at entry and exit from calls.
  Each node in tree corresponds to an exit event.

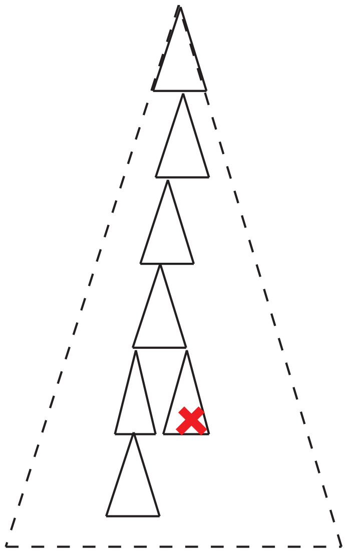
- *Internal* events at decision points which affect control flow (e.g. if-then-elses etc).

- Each event is assigned an event number.

Building the tree

- Generate the tree piece by piece on demand.

- We rerun a call if we need to generate nodes below that call.



Search strategies

Divide and query

- We pick a node for each question which divides the tree into two roughly equal portions.

- Results in O(log n) questions on average.

- Query optimal in the absence of other information.

Divide and query example

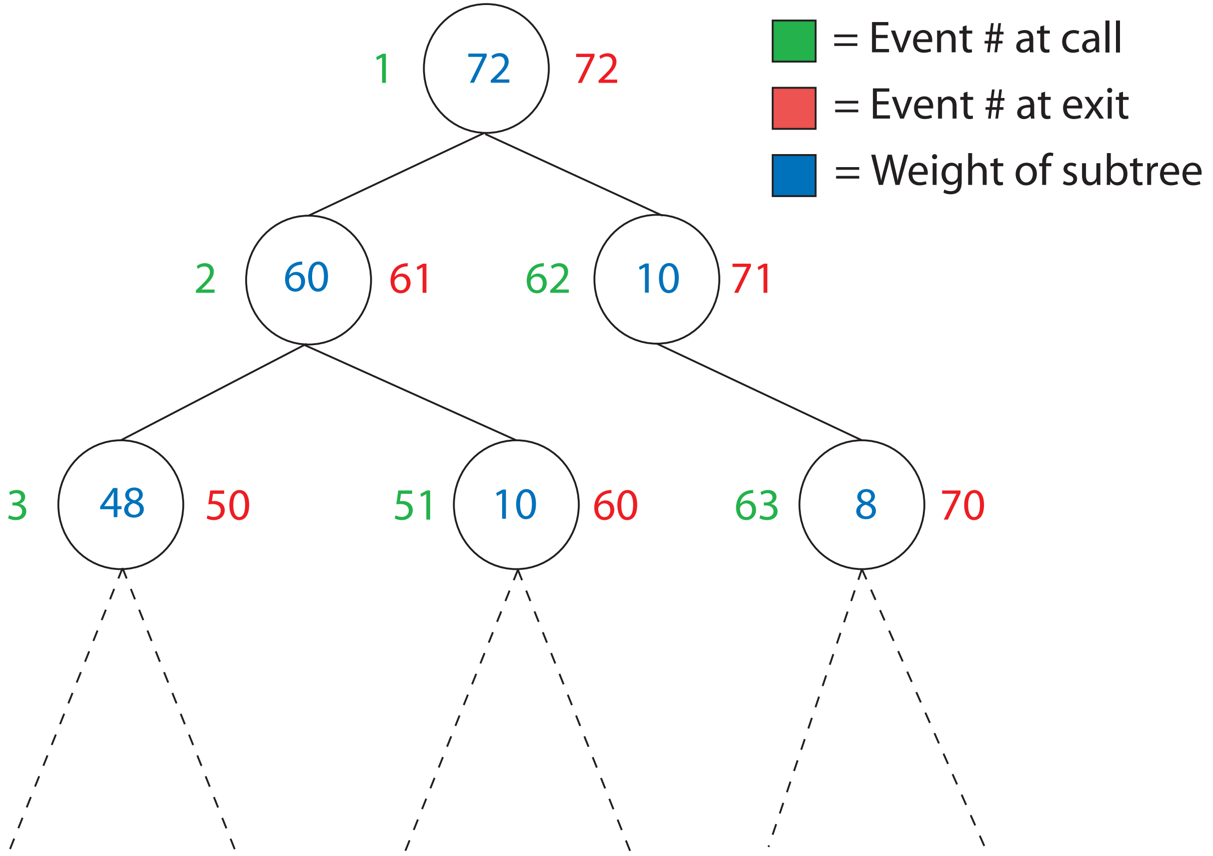
```
area(circle(Radius)) = Radius * pi.  % should be sqr(Radius) * pi
area(box(Width, Height)) = Width * Height.

areas([]) = [].
areas([Shape | Shapes]) = [area(Shape) | areas(Shapes)].

areas([box(2, 3), box(4, 5), circle(2), box(3, 4)]) = [6, 20, 6.28, 12].
Valid?  no
areas([circle(2), box(3, 4)]) = [6.28, 12].
Valid?  no
areas([box(3, 4)]) = [12].
Valid?  yes
area(circle(2)) = 6.28.
Valid?  no
Found bug:
area(circle(2)) = 6.28
```

Efficient divide and query

- Only feasible if the debugger can know the size of subtrees without having to materialize them first.

- Use difference in event numbers at call and exit events to estimate the weight of a subtree.

Subtree weights for divide and query



Tracking the origin of a subterm

- The user often has more information to give than simply “erroneous” or “correct”.
- If a call is erroneous there is often a small part of one of the arguments which is incorrect.
- If the user indicates that a subterm of an argument is incorrect the declarative debugger will ask about the call which created the subterm.

Subterm dependency tracking example

```
areas([box(2, 3), box(4, 5), circle(2), box(3, 4)]) = [6, 20, 6.28, 12].
Valid?  browse return
browser> cd 2/2/1
browser> print
6.28
browser> mark
area(circle(2)) = 6.28
Valid?  no
Found bug:
area(circle(2)) = 6.28
```

Decreases question sizes and number of questions.

Subterm dependency tracking algorithm

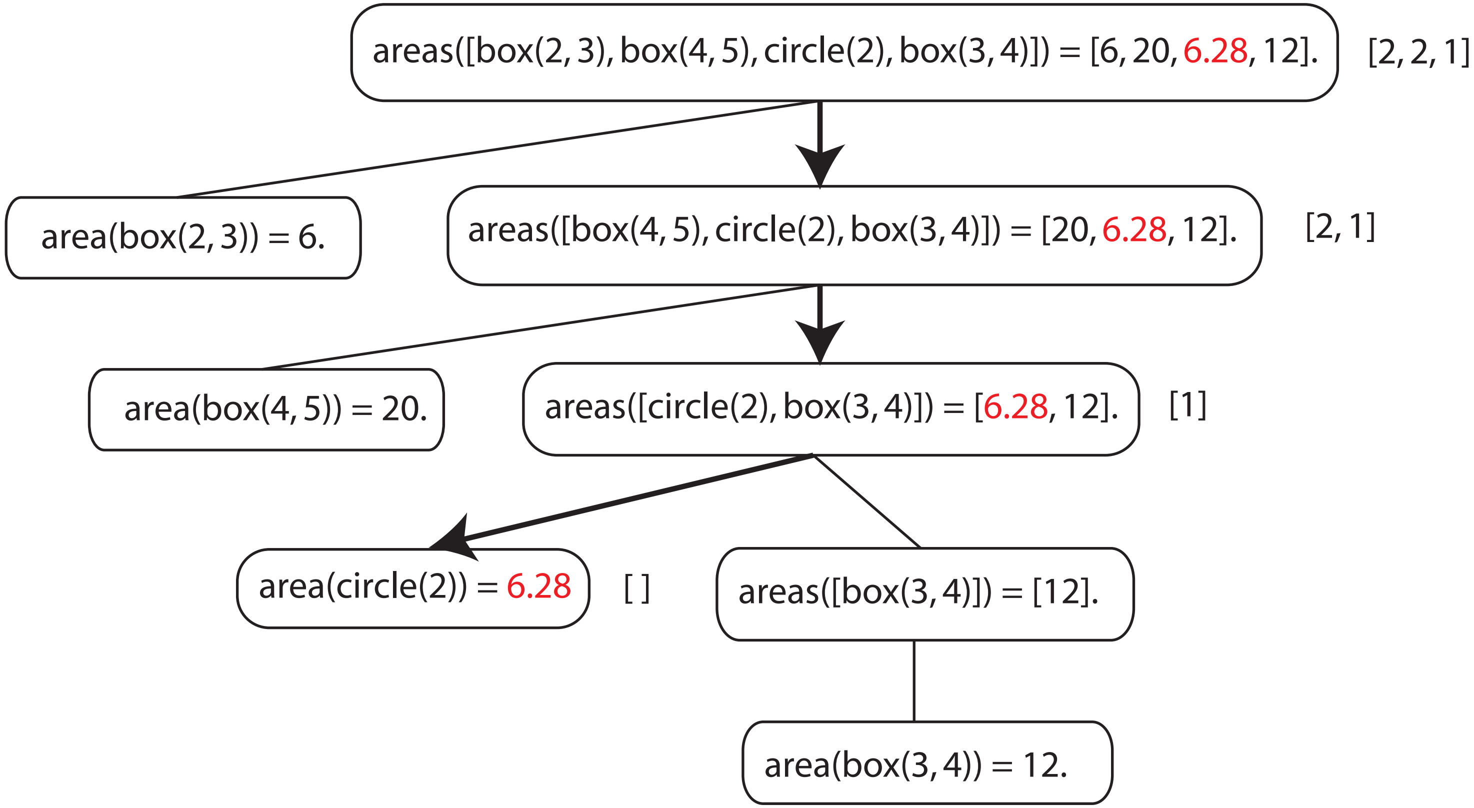
We store a representation of the procedure body in the generated executable.

Algorithm has two parts:

- *Tracking subterm within procedure body*
  Use representation of procedure body and internal events.

- *Tracking subterm between procedure calls*
  Keep track of path to subterm and argument in which subterm appears.

Subterm tracking example



areas([box(2, 3), box(4, 5), circle(2), box(3, 4)]) = [6, 20, 6.28, 12].

[2, 2, 1]

area(box(2, 3)) = 6.

areas([box(4, 5), circle(2), box(3, 4)]) = [20, 6.28, 12].

[2, 1]

area(box(4, 5)) = 20.

areas([circle(2), box(3, 4)]) = [6.28, 12].

[1]

area(circle(2)) = 6.28

[ ]

areas([box(3, 4)]) = [12].

area(box(3, 4)) = 12.

Differences from other approaches

- Require no additional information besides procedure bodies. No time overhead when not used. Space cost is acceptable.
- Allow *sub*-values to be marked.
- Proceed directly to the call which created the marked subterm instead of eliminating calls not on the slice.

Experiences

We have used the declarative debugger to find several real bugs in the Mercury compiler and the declarative debugger itself.

Future work

- Using information from passing and failing test cases to guide search.

- Using CVS/RCS/Subversion diffs.

Latest version available at www.cs.mu.oz.au/mercury.

Questions?