PAT - Assignment 3

Mikkel Willén
bmq419

May 17, 2024

Task 1

We follow the Online PE algorithm, with the initail store:

```
    (Q, Right, Left, Qtail)

(Q, Right)

((init, [Q, Right]))
  (init, [Q, Right, _, _]) : Qtail := Q; Left := (); goto (loop,
                                          [Q, Right, Left, Q]);

  (loop, [Q, Right, Left, Q]) : goto (cont, [Q, Right, Left, Q]);
  (cont, [Q, Right, Left, Q]) : goto (cont1, [Q, Right, Left, (
                                  1: right 2: goto 0 3: write 1)]);
  (cont1, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                             goto (cont2, [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)]);
  (cont2, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                             goto (cont3, [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)]);
  (cont3, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                             goto (cont4, [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)]);
  (cont4, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                             goto (do-if, [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)]);
  (do-if, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                             if 0 = firstsym(Right) goto (jump,
                             [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)])
                             else (loop, [Q, Right, Left,
                             (1: right 2: goto 0 3: write 1)]);


  (jump, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
                     goto (loop, [Q, Right, Left, (3: write 1)]);
```

```
(loop, [Q, Right, Left, (3: write 1)]) : goto (cont,
                                [Q, Right, Left, (3: write 1)]);
(cont, [Q, Right, Left, (3: write 1)]) : goto (cont1,
                                          [Q, Right, Left, ()]);
(cont1, [Q, Right, Left, ()]) : goto (cont2, [Q, Right, Left, ()]);
(cont2, [Q, Right, Left, ()]) : goto (do-write, [Q, Right, Left, ()]);
(do-write, [Q, Right, Left, ()]) : Right := cons(1, tl(Right));
                                   goto (loop, [Q, Right, Left, ()]);
(loop, [Q, Right, Left, ()]) : goto (stop, [Q, Right, Left, ()]);
(stop, [Q, Right, Left, ()]) : return Right;


(loop, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
          (cont, [Q, Right, Left, (1: right 2: goto 0 3: write 1));
(cont, [Q, Right, Left, (1: right 2: goto 0 3: write 1)]) :
          goto (do-right, [Q, Right, Left, (2: goto 0 3: write 1)]);
(do-right, [Q, Right, Left, (2: goto 0 3: write 1)]) :
                       Left := cons(firstsym(Right), Left);
                       Right := tl(Right); goto (loop,
                       [Q,  Right, Left, (2: goto 0 3: write 1)]);
(loop, [Q,  Right, Left, (2: goto 0 3: write 1)]) : goto (cont,
                       [Q,  Right, Left, (2: goto 0 3: write 1)]);
(cont, [Q,  Right, Left, (2: goto 0 3: write 1)]) : goto (cont1,
                                   [Q,  Right, Left, (3: write 1)]);
(cont1, [Q,  Right, Left, (3: write 1)]) : goto (cont2,
                                   [Q,  Right, Left, (3: write 1)]);
(cont2, [Q,  Right, Left, (3: write 1)]) : goto (cont3,
                                   [Q,  Right, Left, (3: write 1)]);
(cont3, [Q,  Right, Left, (3: write 1)]) : goto (do-goto,
                                   [Q,  Right, Left, (3: write 1)]);
(do-goto, [Q,  Right, Left, (3: write 1)]) : goto (loop,
                                              [Q,  Right, Left, Q]);
```

We note that the last loop laben and store, called from the loop branch, is the same as the store right after init. Thus we can eliminate some of the trivial transitions by transition compression, where we get the following:

```
read(Right)

init: Left := '();
      if 0 = firstsym(Right) goto jump else loop;

jump: Right := cons(1, tl(Right));
      return Right;

loop: Left := cons(firstsym(Right), Left);
      Right := tl(Right);
      if 0 = firstsym(Right) goto jump else loop;
```

Task 2

The transformation performs the target Futamura projection(which is the first Futamura projection), where we specialize the interpreter to a specific source program, which generates a target program.

Task 3

```
read (Right);

lab0:
    Left := '()';
    if B = firstsym(Right) goto lab2 else goto lab1;

lab1:
    Left := cons(firstsym(Right), Left);
    Right := tl(Right);
    if B = firstsym(Right) goto lab2 else goto lab1;

lab2:
    Right := cons('1', tl(Right));
    goto lab3;

lab3:
    if B = firstsym(Right) goto lab5 else goto lab4;

lab4:
    Right := cons(firstsym(Left), Right);
    Left := tl(Left);
    if B = firstsym(Right) goto lab5 else goto lab4;

lab5:
    Right := cons('1', tl(Right));
    return(Right);
```

Task 4

Since the number of static values at specialization time is finite, there is a finite number of different PE states and thus the partial evaluator terminates no matter what. There are only a static number of values at specialization time, since the static values are just fragments of the static Turing machine program.