Project Demo

Michael P. Fourman

February 2, 2010

You will find it easiest to work within emacs as this will allow you to scroll back and forth through the demo output. To run the student code do the following:

- run the command
  `ml projdemo`
  (start a new session for each student).
- to the ml prompt type
  `doStudent` *login_name*
- when the student code has successfully compiled type
  `projtest completed`
- You can now test each stage by typing (for example)
  `projdemo Stage1`

Stage1

Testing using code from file `t1`

Code from file is

```
x :=  x  +  y
y :=  x  +  y
```

Code after "optimisation" is

```
_0 :=  x
_1 :=  y
_2 :=  _0  +  _1
_3 :=  _2  +  _1
y :=  _3
x :=  _2
```

Testing using code from file `t2`

Code from file is

```
v :=  v  +  w
x :=  x  +  y
x :=  x  +  z
y :=  x  +  y
y :=  y  +  w
v :=  x  +  y
```

Code after "optimisation" is

```
_0 :=  x  +  y
_0 :=  _0  +  z
_1 :=  _0  +  y
_1 :=  _1  +  w
_2 :=  _0  +  _1
v :=  _2
y :=  _1
x :=  _0
```

Note that the repeated subexpression `x + y` is only computed once, and that the redundant computation of `v + w` is not performed.

Stage2

Testing using code from file `t1`

Code from file is

```
x :=  x  +  y
y :=  x
```

Code after "optimisation" is

```
_1 :=  x  +  y
y :=  _1
x :=  _1
```

Anything similar, that gives the correct result is OK (temporary variables may be used differently, or not at all).

Testing using code from file `t3`

Code from file is

```
_0 :=  x
x :=  y
y :=  _0
```

Code after "optimisation" is

```
_0 :=  y
_1 :=  x
y :=  _1
x :=  _0
```

The point here is to make sure that the values in `x` and `y` are actually swapped.

Stage3 Testing using code from file `t4`

Code from file is

```
_0 :=  x
x :=  x  +  x
x :=  x  +  x
y :=  x
x :=  _0
```

Code after "optimisation" is

```
_1 :=  x
_0 :=  _1  +  _1
_0 :=  _0  +  _0
x :=  _1
y :=  _0
```

This is a code fragment without jumps, so it should be simple. The next example tests both forward and backward jumps.

Testing using code from file `t9`

Code from file is

```
x :=  x  +  y
if ( x  >  0 ) goto 4
y :=  y  +  x
if ((t)) goto 5
z :=  z  +  w
w :=  w  +  x
if ( x  <  y ) goto 0
```

Code after "optimisation" is

```
_0 :=  x  +  y
x :=  _0
if ( x  >  0 ) goto 9
_0 :=  y  +  x
y :=  _0
_0 :=  w  +  x
w :=  _0
if ( x  <  y ) goto 0
if ((t)) goto 12
_0 :=  z  +  w
z :=  _0
if ((t)) goto 5
```

Notice that `flowToCode` performs some rearrangements, so that the correspondence with the source code is not always clear.

Stage4 Testing using code from file `t6`

Code from file is

```
c :=  10  +  a
y :=  y  +  c
if ( y  >  c ) goto 0
```

Code after "optimisation" is

```
_0 :=  10  +  a
!0 :=  _0
_1 :=  !0
_0 :=  y  +  _1
c :=  _1
y :=  _0
if ( y  >  c ) goto 2
```

Note that the computation of `10 + a` is taken outside the loop. This example might be simplified by passing the value in `c`. However, any attempt to do this should take account of the following example, whose source code is only slightly different.

Testing using code from file `t7`

Code from file is

```
y :=  y  +  c
c :=  10  +  a
if ( y  >  c ) goto 0
```

Code after ”optimisation” is

```
_0 :=  10  +  a
!0 :=  _0
_0 :=  !0
_1 :=  y  +  c
y :=  _1
c :=  _0
if ( y  >  c ) goto 2
```

Here, the value of c is used within the loop before it is changed.

In the next example, the invariant expression is not the final value of any variable in the loop.

Testing using code from file `t8`

Code from file is

```
z :=  10  +  a
y :=  y  +  z
z :=  10  +  y
if ( y  >  c ) goto 0
```

Code after ”optimisation” is

```
_0 :=  10  +  a
!0 :=  _0
_1 :=  y  +  !0
_0 :=  10  +  _1
y :=  _1
z :=  _0
if ( y  >  c ) goto 2
```

Stage5 Testing using code from file `t10`

Code from file is

```
z :=  z  +  1
x :=  x  +  1
y :=  y  +  1
w :=  w  +  1
if ( x  >  y ) goto 9

z :=  z  +  2
x :=  x  +  2
y :=  y  +  2
w :=  w  +  2

z :=  a  +  b
x :=  z  +  y
y :=  x  +  z
```

Code after "optimisation" is

```
_2 :=  w  +  1
_1 :=  y  +  1
_0 :=  x  +  1
w :=  _2
y :=  _1
x :=  _0
if ( x  >  y ) goto 11

_1 :=  w  +  2
_0 :=  y  +  2
w :=  _1
y :=  _0

_0 :=  a  +  b
_1 :=  _0  +  y
_2 :=  _1  +  _0
y :=  _2
x :=  _1
z :=  _0
```

The code here contains three blocks, indicated by adding blank lines to the layout above. Neither `x` nor `z` is needed as an input to the final block. However, `x` is needed as an output of the first block, as it is used in the condition.

The next test looks at the analysis of data flow through an inner loop. Few people will have attempted this stage. Testing using code from file `t11`

Code from file is

```
z :=  a  +  b
y :=  y  +  z
x :=  x  +  y
w :=  w  +  x
v :=  v  +  w
z :=  z  +  z
if ( x  <  y ) goto 0

z :=  a  +  b
w :=  c  +  d
x :=  b  +  c
y :=  a  +  d
```

Code after "optimisation" is

```
_0 :=  a  +  b
_0 :=  y  +  _0
_1 :=  x  +  _0
_2 :=  w  +  _1
_3 :=  v  +  _2
v :=  _3
w :=  _2
x :=  _1
y :=  _0
if ( x  <  y ) goto 0

_3 :=  a  +  d
_2 :=  b  +  c
_1 :=  c  +  d
_0 :=  a  +  b
y :=  _3
x :=  _2
w :=  _1
z :=  _0
```

The variables `w,x,y,z` are not needed as an inputs of the final block; however, `v` is needed. Thus `v, x, y` are needed as outputs of the loop (because `x` and `y` are needed in the condition of the loop). Furthermore, `w` is needed, since `v` depends on `w`. Note that `y` does not depend on `z`; it depends on `a` and `b`.

Stage6a Testing using code from file `t5`

Code from file is

```
x :=  x  +  5
x :=  x  +  10
x :=  x  +  x
y :=  a  +  b
y :=  y  +  c
z :=  b  +  c
z :=  a  +  z
```

Code after "optimisation" is

```
_2 :=  a  +  b
_2 :=  _2  +  c
_0 :=  x  +  30
_0 :=  _0  +  x
z :=  _2
y :=  _2
x :=  _0
```

Note that `a + b` is computed only once, and that the final value of x is computed as `30 + x + x` (in some order).

Stage6b This may be tested as for Stage2. I hope that some students will have made a better job than I have of eliminating unneccessary temporary varaibles.