221 Compilers

Exercise 5: Points-to analysis

In this exercise we explore a dataflow analysis for pointers. We begin with a Haskell data type for instructions for a simple machine with registers:

```
data Instr = Define String   -- "label:"
           | Mov Reg Reg     -- "mov.l xxx yyy" (yyy:=xxx)
           | Cmp Reg Reg     -- "cmp.l xxx yyy" (test yyy-xxx)
           | Bgt String      -- "bgt label" (branch if greater than zero)
           | New Int Reg     -- "yyy = malloc(xxx)" (storage allocation)
data Register = D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7
```

For the purposes of this question, an extra instruction, called `New`, has been added. For example, this instruction, with identifier *id*,

```
id:     New 8 D0
```

allocates 8 bytes of memory and sets register `D0` to point to it. Our goal is to define a data-flow analysis that computes, for each `Register`, the set of identifiers of the allocations that the register might point to. For example, given:

```
1:      New 8 D0
2:      New 8 D3
3:      Cmp D1 D2
4:      Bgt L
5:      Mov D0 D3
6:  L:
```

then, after line 5, we can say that `D0` points to the allocation at line 1, and `D3` may point to either the allocation at line 1 or the allocation at line 2. We represent this as a points-to-set { (D0, 1), (D3, 1), (D3, 2) }. The effect of an instruction on the points-to-set before its execution depends on the instruction; we define a function `effect`:

```
effect ::  PointsToSet → Instruction → PointsToSet
effect pts (Node id (Cmp r1 r2)) = pts
effect pts (Node id (Bgt label)) = pts
effect pts (Node id (New n r)) = pts ∪ {(r, id)}
effect pts (Node id (Mov r1 r2)) =
```

(i) Complete the missing definition of `effect` above, for `Mov`.

(ii) Write down the defining equation for *pointsIn(n)*, the points-to-set just before node n of the control-flow graph, and the defining equation for *pointsOut(n)*, the points-to-set just after node n.

(iii) Show how `effect` can be improved by enhancing the rule for `New`.

What do you think points-to information might be used for in an optimising compiler? Could you also use it for static detection of software defects?

Paul Kelly   *Imperial College*   *November 2010*

221 Compilers

Exercise 5: Points-to analysis - solution

(i): We need to add points-to relations saying that `r2` might now point to anything `r1` might point to:

```
effect pts (Node id (Mov r1 r2)) = pts ∪ [(r2, id) | (r1,id) ← pts]
```

(ii):

$$pointsIn(n) = \cup_{p \in pred(n)} pointsOut(p)$$

$$pointsOut(n) = effect(pointsIn(n))(instruction_n)$$

(iii): The rule for `New` does not account for the points-to elements that are killed by the assignment. To do better we need to remove them:

```
effect pts (Node id (New n r))
   = pts' ∪ {(r, id)}
   where pts' = [(reg,t) | (reg,t) ← pts, reg ≠ r]
```

Paul Kelly *Imperial College* *November 2010*