Introduction to Theoretical Computer Science

Exercise Sheet: Week 4. Answers

(1) Write down the code for an RM macro 'if $R_1 > R_2$ then goto I_j'. The macro must leave all registers unchanged after its execution. Assume a predefined GOTO macro.

Using R_{-3} (say) as a scratch register:

```
        loop : 0  DECJZ (1, preend)       R1 = 0, so ≯ R2, so don't jump
               1  DECJZ (2, prejump)      R2 = 0 and R1 was > 0, so jump
               2  INC   (−3)              count num of DECs of R1 and R2
               3  GOTO  (loop)            R1, R2 both DEC'ed, go round again
     prejump : 4  INC   (1)               restore R1 to its state before DEC'ed by line 0
prejumploop : 5  DECJZ (−3, j)           if done, goto target, else
               6  INC   (1)               add one back to R1
               7  INC   (2)               and R2
               8  GOTO  (prejumploop)     until finished
      preend : 9  DECJZ (−3, end)         if done, continue at end, else
              10  INC   (1)               add one back to R1
              11  INC   (2)               and R2
              12  GOTO  (preend)
         end : 13
```

(2) Give a simple recursive definition of a sequence coding function $\mathbb{N}^* \to \mathbb{N}$, based on the pairing function in the slides.

For example, $\langle\rangle = 0$ and $\langle h, t\rangle = 2^h 3^{\langle t\rangle}$.

(3) Our register machines have a finite number of registers, each holding an unbounded number. Turing machines have an unbounded number of cells, each holding one of a finite set of symbols.

Suppose we allow register machine to have an unbounded number of registers, but each register is finite (e.g. 32 bits) – like current computer memory. With no changes to the instruction set, are these machines still Turing powerful? Why or why not?

No. A program can only refer to finitely many registers; therefore any such machine is just a finite automaton.

Suppose now that we add a form of indirect addressing. For example, we might say that the register operand of an instruction can now be either i, as before, meaning R_i, or (i), meaning R_{R_i}. Does that help?

No. We're still limited to the registers given directly, plus the registers $R_0, \dots, R_{2^{32}-1}$. Incidentally, this was also a bit of an issue in reality: the amount of memory one could attach to a CPU went above 2GiB *(or 4GiB) before architectures switched to 64-bit addresses, so various hacks were added.*

Why aren't Turing machines affected by this issue? Can you adapt ideas from TMs to solve this issue for RM?

Because TMs can access any cell, even though they have to get there one at a time. A similar idea might be to add a 'current register', whose index can be increased by one without limit.

Another feature to add to RM would be to allow unbounded indirection in a tree-like way, or allow some form of indirect specification where the index is given by a sequence of registers. (This needs some work to spell out formally.)

(4) The proof of the Halting Problem relies on the lethal combination of *self-reference* (when the machine is run on itself) and *negation* (when we flip the result of the halting analyser). Here are some other famous contradictions/paradoxes. Discuss what they show or how they might be resolved.

(a) 'The barber shaves all and only the men who do not shave themselves.'

There is no contradiction if one allows the case that the barber is a woman. However, if one adds the requirement that the barber is a man then there really is a contradiction. (The above is a story problem formulation of Russel's Paradox from set theory.)

(b) 'The set of sets that are not members of themselves.'

In standard set theory, the paradox is broken in two ways: (i) firstly by requiring that no set can be a member of itself, and (ii) secondly by restricting the ways in which one can form sets. (i) is not a complete answer, because then the defined set is the set of all sets, which must be a member of itself, which is not allowed. (ii) resolves the remaining contradiction by saying that the definition is simply not a legitimate definition: basically, we ban any definitions of sets that require quantifying over all sets. (More precisely, we allow quantification only over the members of sets that have already been defined.)

(c) 'The smallest natural number not definable in under eleven words.'

This is a similar issue to the previous. What does 'definable' mean here? We haven't said, and the paradox relies on being able to write circular and woolly definitions. If we made things precise enough, e.g. 'the smallest number not definable by a first-order logic predicate of length ten', the problem goes away, because 'definable by FOL predicate of length ten' is not itself expressible as an FOL predicate of length ten. Even if we do that, it's uncomputable to work out the shortest definition of a number.