Name: _____

Exercise B4, 2/22/05

Part 1: Review "Reduction"

Once a problem is solved, we don't want to repeat solving similar problems from scratch. As educated people, we will definitely try to apply what we know. The notion of "reduction" is useful in this respect.

Task: Come up with at least one unique example of "reduction" between problems (in the way discussed in class, *not* the everyday use such as "price reduction"). Then, analyze how "positive"/"negative" properties can be transferred from one problem to the other.

Part 2: TM Variants

The TM industry has a wide variety of models. As in the real world, different models suit different needs. However, it has been known that most of these variants are equivalent in the sense that they can recognize exactly the same problems, i.e., TM-recognizable problems. In other words, we can show that equivalent variants can simulate each other. Let us now consider the following two variants:

- **Multiple-tape TM**: A Turing machine with multiple infinite tapes and a finite control which independently accesses positions on different tapes (but still via step-by-step left/right movements). The input is placed on the first tape.
- **Nondeterministic TM**: A Turing machine whose transition function can actually be a relation. That is, for a given state and input symbol, there may be more than one transitions. Naturally, the behavior of such a machine would branch whenever it encounters a multiple-transition point. Note: Recall the distinction between NFA and DFA.

To show the equivalence, we will need to discuss the both directions of simulation. First, a standard TM is reducible to a multiple-tape TM or a nondeterministic TM, i.e., we can simulate a standard TM with a multiple-tape TM or a nondeterministic TM. That is, these new variants are at least as powerful as the standard one. We will discuss the other direction in class.

Task: Choose one of the following options and concisely discuss the essence of how the variant of your choice can be simulated by a standard TM. Be prepared to discuss in class.

- Option A: Multiple-tape TM
- Option B: Nondeterministic TM

Part 3: Alternative Abstract Models of Computation

Although the Turing machine is the most widely used abstract model of computation, there are many others. In this part, we discuss Unlimited Register Machines (URM) and "recursive functions." Note that "recursive function" here refers to a full specification of a computational model, not just instances of recursive calls. Brief descriptions for these models are available in Appendices below. Again, this part is about the question of equivalence, this time between TMs and these new models. If all of these models are equivalent, there must be something universal about the notion of "computation."

For this part, let us limit to one direction, i.e., simulation of these new models by a TM (or reduction of the models to TMs). Then, we know that the TM is at least as powerful as these models.

Task: Choose one of the following options and concisely discuss the essence of how the model of your choice could be simulated by a TM. Be prepared to discuss in class.

- Option A: Unlimited Register Machines (URM) [Probably, this option is more intuitive.]
- Option B: Recursive Functions

Survey: Time spent between classes: _____

Appendix A: Unlimited Register Machine (URM)

URM is an abstract model of computation, which is closer to a CPU than a TM, with the following properties:

1. There are an infinite number of *registers* called R_1, R_2, R_3, ..., each of which contains a natural number. The number in register R_i is referred to as r_i.
2. Register contents may be changed by the following *instructions*:
   - Zero: Z(i) will reset r_i to 0.
   - Successor: S(i) will add 1 to r_i.
   - Transfer: T(i, j) will copy r_i to R_j, i.e., $r_j = r_i$.
   - Jump: J(i, j, n) will first evaluate if $r_i = r_j$. If true, the URM will proceed to the nth instruction. Otherwise, it will proceed to the next instruction.
3. A *program* consists of a finite sequence of instructions. Execution of a program begins with the first instruction and continues to the next ones until there is no instruction (except for possible jumps).
4. One may assume arbitrary initial values in the registers.

References

- Shepherdson, J. C. and Sturgis, H. E. 1963. Computability of recursive functions. *J. ACM* 10, 217-55. [original]
- Cutland, N. J. 1980. *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press. [contains a description and the use of URM]

Appendix B: Recursive Functions

The class of recursive functions (μ-recursive functions) is the class of total functions that can be built up from the basic functions (below) by a finite number of operations of *substitution*, *recursion*, and *minimalization* (below):

Basic functions

1. Zero: 0 (i.e., returns 0 as constant)
2. Successor: $x + 1$ (i.e., add 1)
3. Projection: $U^n_i (x_1, x_2, ..., x_n) = x_i$ (i.e., picks up the ith element from a list of n elements)

Operations

- Substitution (composition): Given computable functions $f(y_1, ..., y_k)$ and $g_1(x)$, ..., $g_k(x)$, define $h(x) = f(g_1(x), ..., g_k(x))$
- Recursion: Given computable functions $f(x)$ and $g(x, y, z)$, define
  - $h(x, 0) = f(x)$
  - $h(x, y + 1) = g(x, y, h(x, y))$
- Minimalization: Given computable function $f(x, y)$, define $h(x)$ = the least y such that $f(x, y) = 0$

References

- Godel-Kleene. 1936. [original, not easily accessible]
- Cutland, N. J. 1980. *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press. [detailed discussion of recursive functions]

// End