

Unit A3 Supplement, 1/29/05

Here is my response to your summary questions (rephrased/combined in some case).

Q1: Possible to express the “actual” solution of a problem in set notation, esp. when the number of possible solution is very large? How general the predicate notation on a set could be.

A1: The list notation is good only for a finite set. But as long as the set is finite, you can still contain a large number of members, at least theoretically. Many problems have infinitely many solutions. The “problem” (or solution) set for such problems must be in the predicate notation. If the specification part (at the right of the bar ‘|’) of a representation is precise enough for the “processor” to interpret, the predicate notation is good enough. For example, $\{(c, e) \mid c \text{ causes } e\}$ would be good only for informal discussion, not for a computational process, while $\{(x, y) \mid x < y, \text{ where } x, y \in \text{NaturalNumbers}\}$ can be processed mechanically.

Q2: Representation of sets besides the basics?

A2: When you deal with a real-world problem, the data can be complex. As an example, let us consider Group Exercise 3 (Slide 24), which we did not have time to discuss. Here is the first attempt: $\{(course, instructor) \mid instructor \text{ teaches } course\}$. Would this be right? No, because it characterizes a single instance of course-instructor mapping. The problem must address the entire map that we get the complete information about who is teaching what in within realistic constraints. How about this: $\{(courses, instructors, assignment) \mid assignment \text{ is an injective function from } courses \text{ to } instructors, \text{ where } courses \text{ and } instructors \text{ are non-empty sets of available courses and instructors}\}$? For example, imagine $courses = \{410, 460, 470\}$, $instructors = \{NN, NK, PD, MM\}$, and $assignment = \{(410, PD), (460, MM), (470, NN)\}$. $assignment$ is a function (why?) and injective (why?). Now, you should be able to see how the materials discussed in Discrete Structures begin to play a role. [Review questions: What’s wrong if $assignment$ is not a function or injective? Why doesn’t $assignment$ need to be surjective?]

Q3: Problems in CS for which it is impossible to prove whether they are “computable” or not? How to finally decide if something is “computable” or not?

A3: To answer these questions, we need to know a little more about the notion of “computability,” which we will discuss in the rest of Module A and much of Module B. Keep your question. As a side note, I want to mention a logical problem that cannot be proven true or false. Consider the sentence: “I’m telling a lie.” Am I really telling a lie? This “liar paradox” has a close connection to “computability.” We will probably touch on this point later.

Q4: The distinction between what is necessarily included in the discussion about complexity vs. what is excluded?

A4: The complexity area of the Theory of Computation focuses on the time or space performance of algorithms as a function of their input size. In this area, we only discuss algorithms, which must terminate for all inputs; thus, computability is not an issue. In addition, we only deal with mechanisms that are capable of solving the problem in hand; thus, choosing the simplest mechanism is not an issue.

Q5: How to apply/figure out the properties in the three subareas of the Theory in “my” problems or other problems like the floor tiling?

A5: The entire course is supposed to be the opportunity for you to gain this ability. So, at this point, you should not worry about not being able to do this as much as you wish. However, I hope you got some sense of the three subareas of the Theory and be able to guess which areas are more relevant to your own problems. To get some additional ideas, you may want to use the sample problems from Spring 2003 (<http://www.tcnj.edu/~komagata/csc460/05s/SampleProblems.pdf>). Problems 1 through 10 primarily involve issues in computability; Problems 11 through 19 primarily involve issues in complexity; Problems 20 through 23 primarily involve issues in languages/automata. As I mentioned in class, most of your problems have some relevance to some of these areas. In addition, if your problem calls for a program that should not terminate or involves high degree of interactivity, it might require analysis beyond the traditional Theory. For example, the robotic control of automobile must not terminate and my Birds program is most appropriately designed as interaction of autonomous birds.

Q6: How can formal models of interactivity be mathematically expressed?

A6: I am planning to include a very basic discussion related to this topic near the end of the semester. Some references are listed in the section, “Beyond Turing computability” of the references file (<http://www.tcnj.edu/~komagata/csc460/05s/References.pdf>). Here is a preview of the discussion. To address interactivity (or computation beyond algorithms), people have proposed different approaches:

- Instead of discrete data/processing, use analog computation (e.g., analog version of artificial neural network)
- Instead of limiting to a single input-processing-output session, allow the mechanism to repeat the process with some memory between sessions
- Instead of prohibiting interaction within the process, allow the process to interact with (or consult) another process.

With any of these approaches, we can show that it would become possible to do what was not possible with a single session of algorithmic computation.

If you have more questions, let me know.

// End