MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Fall Semester, 2007

Assignment 2

Issued: Tuesday, Sept. 11

This handout contains:

- Software Lab for Tuesday, September 11
- Pre-Lab exercises to do before Thursday, September 13 at 10AM; you can come and do them in lab on Wednesday, September 12. Don't forget that Part 2.2 of the on-line Tutor problems are also due before Thursday lab.
- Robot Lab for Thursday, September 13 (read the lab before coming to 34-501)
- Post-lab writeup and exercises due Tuesday, September 18 at 10AM. Don't forget that Part 2.3 and 2.4 of the on-line Tutor problems are also due before Tuesday lecture.

Higher-order procedures; Behaviors and utility functions

This week's work gives you practice with *higher-order procedures*, i.e., procedures that manipulate other procedures. Tuesday's lecture, the post-class software lab, and the on-line problems, cover Python's support for *functional programming*. Thursday's lab applies higher-order procedures to the tasks of specifying robot *behaviors* with the aid of *utility functions*.

Make sure you do:
`athrun 6.01 update`
so that you can get the `Desktop/6.01/lab2` directory which has the files mentioned in this handout.

Tuesday Software Lab: Higher-order procedures

At the end of this lab, go to the on-line Tutor at `http://sicp.csail.mit.edu/6.01/fall07`, choose PS2, and paste all your code, including your test cases, into the box provided in the "Software Lab" problem.

This session gives you practice with Python's basic tools for functional programming: higher-order procedures and list comprehension.

A higher-order procedure is a procedure that takes procedures as inputs and/or returns procedures as results. We will be working with a simple model of probability spaces, one that we will use later in the course.

Probability

Probability theory is a calculus that allows us to assign numerical assessments of uncertainty to possible events, and then do calculations with the numerical assessments in a way that preserves their meaning. (A similar system that you might be more familiar with is algebra: you start with some facts that you know, and the axioms of algebra allow certain manipulations that will preserve truth.)

We'll just consider worlds that can be represented by the values of a small number of discrete variables. We'll let U be the *universe* (or sample space), which is a set of *atomic events*. An atomic event is just an outcome or a way the world could be. The universe associated with rolling a normal die once can be written as

$$U = \{1, 2, 3, 4, 5, 6\} \ .$$

The universe associated with rolling two dice, or one die twice, can be written as

$$U = \{(1,1), (1,2), ..., (1,6), ..., (6,5), (6,6)\} \ .$$

Atomic events could indicate which room a robot is in and whether the battery is charged, or any description of the world under consideration, but we are assuming that there are a finite number of possible combinations of values, so that the universe is finite.

An *event* is a subset of U. For example the set of atomic events in which the single die roll result is greater than 3 is an event. A probability *measure* P is a mapping from events to numbers that satisfy the following axioms:

$$\begin{aligned} P(U) &= 1 \\ P(\{\}) &= 0 \\ P(A \cup B) &= P(A) + P(B) - P(A \cap B) \end{aligned}$$

Or, in English:

- The probability that something will happen is 1.
- The probability that nothing will happen is 0.
- The probability that an atomic event in the set A or an atomic event in the set B will happen is the probability that an atomic event of A will happen plus the probability that an atomic event of B will happen, minus the probability that an atomic event that is in both A and B will happen (because those events effectively got counted twice in the sum of $P(A)$ and $P(B)$).

Armed with these axioms, we are prepared to do anything that can be done with discrete probability!

For example, consider the universe associated with rolling two fair dice (shown above); each of the atomic events is equiprobable. So, each event has probability 1/36.

- The probability of rolling a pair of twos is: $P(Die1 = 2 \cap Die2 = 2) = 1/36$
- The probability that the first die is a 2 is: $P(Die1 = 2) = 1/6$, the sum of 6 atomic events.
- The probability of one of the dice coming up 2 is

  $$P(Die1 = 2 \cup Die2 = 2) = P(Die1 = 2) + P(Die2 = 2) - P(Die1 \cap Die2) = 1/6 + 1/6 - 1/36 \ ,$$

  note that the $(2, 2)$ atomic event is included in both events "Die1 = 2" and "Die2 = 2".

One more important idea is *conditional probability*, where we ask the probability of some event E_1, assuming that some other event E_2 is true; we do this by restricting our attention to the part of the sample space (universe) in E_2. The conditional probability is the amount of the sample space that is both in E_1 and E_2, divided by the amount in E_2:

$$P(E_1|E_2) = \frac{P(E_1 \cap E_2)}{P(E_2)} \ .$$

That is, the fraction of the E_2 probability that is also in E_1.

The expression $P(E_1|E_2)$ is read as: the conditional probability of E_1 **given** E_2 or, more loosely, the probability of E_1 given E_2. The expression $P(E_1 \cap E_2)$ is read as: the probability of E_1 **and** E_2.

Manipulating Probability Spaces

We're going to build Python models of discrete *probability spaces*, that is, sample spaces with corresponding probability measures. Concretely, we'll represent a probability space as a list of atomic events, each with its assigned probability. Each element of the probability space, which we will call a *sample* (sometimes also known as an outcome) will be a list with two elements: a probability and the value that defines an atomic event. For example, to represent the probability space for a standard, six-sided die we can have:

```
dieSpace = [ [1.0/6, 1], [1.0/6, 2], [1.0/6, 3],
             [1.0/6, 4], [1.0/6, 5], [1.0/6, 6] ]
```

The atomic events don't have to be equiprobable. We can also model a loaded die:

```
loadedDieSpace = [ [1.0/10, 1], [1.0/10, 2], [1.0/10, 3],
                   [1.0/10, 4], [1.0/10, 5], [1.0/2, 6] ]
```

It is essential that the probabilities of all of the samples add up to 1.0, as required by the axioms.

The value used to define an atomic sample could be more complicated, e.g. another list. For example, if we wanted to represent the probability space for the roll of two fair dice, we could have:

```
diceSpace = [ [1.0/36, [1, 1]], [1.0/36, [1, 2]], [1.0/36, [1, 3]],
              ..., [1.0/36, [6, 6]] ]
```