Models for Probabilistic Programs with an Adversary (Extended Abstract)

Robert Rand and Steve Zdancewic
Computer and Information Sciences
University of Pennsylvania

The twin concepts of nondeterminism and probability are central to computing in a variety of domains. Nondeterminism is used to model scenarios in which program behavior is uncertain, such as user input, an action by an opponent or another node in a network, or possible hardware, software or synchronization failure. The unifying theme here is that nondeterminism represents a lack of knowledge. By contrast, probability is precise, modeling the frequency with which we can expect a specific outcome. Las Vegas algorithms like Randomized Quicksort terminate efficiently with high probability while Monte Carlo algorithms like the Miller-Rabin primality test produce a correct result some (generally high) percentage of the time. Probability also plays a central role in Game Theory, Cryptography and Differential Privacy. Tellingly, all three applications involve an adversary, which we can model using nondeterminism. How nondeterminism interacts with the probabilities in the program will be the focus of this study.

Nondeterminism and probability can interact in a variety of ways, depending on the model of nondeterminism being used. Consider the following simple program:

$$(x := 0 \oplus x := 1);\ (y := 0 \sqcup y := 1);\ z := XOR(x, y)$$

where $\oplus$ indicates probabilistic choice and $\sqcup$ indicates nondeterminism. If the nondeterministic choice cannot depend on the outcome of the coin flip, this is equivalent to either the program $(x := 0 \oplus x := 1); y := 0; z := XOR(x, y)$ or the program $(x := 0 \oplus x := 1); y := 1; z := XOR(x, y)$. In either case, z's behavior is determined by the initial coin flip. On the other hand, if the nondeterminism reacts to the random events in the program, this program may well set z to 1 with probability 1 or 0. It's helpful to think of this nondeterminism as governed by an *adversary*, even if it represents a possible hardware or synchronization failure. Our question then becomes "What does the adversary *know*, and what can we guarantee in light of that knowledge?"

We consider the following four models of adversary knowledge:

1. The adversary has knowledge of the program structure, but not the current program state (the mapping of identifiers to values).

2. The adversary knows the values stored in each identifier but not the program history.

3. The adversary knows the complete program history up to the present time.

4. The adversary knows the full program execution, including future actions.

The fourth model can be broken down into two sub-models, depending on whether there is a single source of random bits or distinct sources of bits for each command. This influences whether $(c \sqcup c)$ presents the adversary with a real choice, as the coin flips in each c may land differently in the second sub-model.

$$\frac{\{P\}\ c_1\ \{Q_1\} \qquad \{P\}\ c_2\ \{Q_2\}}{\{P\}\ (c_1 \sqcup c_2)\ \{Q_1 \vee Q_2\}}\ (1)$$

$$\frac{\text{non-probabilistic } P \qquad \{P\}\ c_1\ \{Q\} \qquad \{P\}\ c_2\ \{Q\} \qquad \text{non-disjunctive } Q}{\{P\}\ (c_1 \sqcup c_2)\ \{Q\}}\ (2{,}3)$$

$$\frac{\{P\}\ c_1\ \{Q\} \qquad \text{deterministic } Q \qquad \{P\}\ c_2\ \{Q\}}{\{P\}\ (c_1 \sqcup c_2)\ \{Q\}}\ (4)$$

Figure 1: The Hoare logic choice rules corresponding to our four models. *Probabilistic* means asserting probabilities in the open interval $(0, 1)$, *disjunctive* means containing a disjunction of probabilistic predicates and *deterministic* means neither probabilistic nor disjunctive.

In the first model, the adversary has no knowledge of the program state and can be therefore be thought of as making all of its choices before program execution. This corresponds to the private coin protocol used in interactive proof systems, as well as the assumptions of common cryptographic protocols. We model this level of knowledge by assuming that each non-deterministic choice has an associated infinite stream of zeros and ones which dictate the choice taken at each execution of the given choice command. This type of program is particularly easy to verify using *Hoare Logic*: The first equation in figure 1 says that if program c_1 can guarantee an outcome of Q_1 when P is initially true, and c_2 can similarly guarantee Q_2 then a non-deterministic choice between c_1 and c_2 guarantees Q_1 or Q_2.

The second and third models give the adversary knowledge of the program state and program history, respectively. This complicates the Hoare logic rule, since the adversary may make different choices based on the outcome of an coin flip (as in the Probabilistic Guarded Command Language of [3] and [1]), dramatically broadening the space of possible outcomes. In the second rule of figure 1, we present an elegant Hoare rule involving a non-probabilistic precondition and a non-disjunctive postcondition, based on theorems regarding the interaction of disjunctive and probabilistic assertions in our expanded VPHL paper [4]. Unfortunately this rule is still highly restrictive due to our inability to express the absence of randomly assigned variables in our context. We look at using auxiliary variables, as found in Kleymann [2] to overcome this limitation and distinguish between knowledge carried by the state and the history.

Finally, in the strong model(s), the adversary has access to the program's full array of random bits, corresponding to a compromised system. This further broadens the space of possible outcomes, and limits our possible postconditions. In response, we introduce the simplest possible Hoare logic rule, which requires us to weaken probabilistic postconditions (eg. $Pr(x = 1) = \frac{1}{3} \vee Pr(x = 5) = \frac{1}{3}$) to deterministic ones (eg. $Pr(x = 1 \vee x = 5) = 1$) before combining them.

We also analyze our four models through the lens of program equivalence. In general, we expect $(x \texttt{ := } 0 \ \oplus \ x \texttt{ := } 1); (y = 1 \ \sqcup \ y = 2)$ to behave identically to $(y = 1 \ \sqcup \ y = 2); (x \texttt{ := } 0 \ \oplus \ x \texttt{ := } 1)$, since the two commands modify non-overlapping variables. This is true of models one and four, however in the intermediate models, the adversary can assign y based on the outcome of the coin toss. On the other hand, in the first model the statement `if` b `then` c `else` c is not equivalent to c, since b may be probabilistic and the adversary may proceed differently in each branch. In our other models, any information imparted by b is already known to the adversary, and the commands are equivalent.

Finally, we explore mixed models of adversarial knowledge, in order to analyze programs in which the adversary may know some, but not all, of the random bits in a program. This form of analysis proves relevant for problems in game theory, cryptography and differential privacy, and other closely related fields.

References

[1] Gretz, Friedrich, et al. ”Conditioning in Probabilistic Programming.” Mathematical Foundations of Programming Semantics, 2015.

[2] Kleymann, Thomas. ”Hoare logic and auxiliary variables.” Formal Aspects of Computing 11.5 (1999): 541-566.

[3] Morgan, Carroll, and Annabelle McIver. ”pGCL: formal reasoning for random algorithms.” South African Computer Journal (1999): 14-27.

[4] Rand, R. and S. Zdancewic, “VPHL: A verified partial-correctness logic for probabilistic programs (expanded version).”, Technical Report MS-CIS-15-06, University of Pennsylvania (2015).