Sequential Nondeterminism

CS 536: Science of Programming, Spring 2023

2023-02-22 p.2

A. Why

- Nondeterminism can help us avoid unnecessary determinism.
- Nondeterminism can help us develop programs without worrying about overlapping cases.

B. Objectives

At the end of these practice questions you should

- Be able to evaluate nondeterministic conditionals and loops.

C. Nondeterminism

1. What are the reasons mentioned in the text for why using nondeterminism might be helpful?
2. Let $IF \equiv \textbf{\textit{if}}\ B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2 \ \square \ \dots \ \square\ B_n \rightarrow S_n\ \textbf{\textit{fi}}$ and $BB \equiv B_1 \vee B_2 \vee \dots B_n$.
   a. What property does BB have to have for us to avoid a runtime error when executing IF?
   b. Does it matter if we reorder the guarded commands? (E.g., if we swap $B_1 \rightarrow S_1$ and $B_2 \rightarrow S_2$.)
3. Let $U_1 \equiv \textbf{\textit{if}}\ B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2\ \textbf{\textit{fi}}$ and $U_2 \equiv \textbf{\textit{if}}\ B_1\ \textbf{\textit{then}}\ S_1\ \textbf{\textit{else if}}\ B_2\ \textbf{\textit{then}}\ S_2\ \textbf{\textit{fi fi}}$.
   a. Fill in the table below to describe what happens for each combination of B_1 and B_2 being true or false.

If σ ⊨ …	U_1	U_2
$B_1 \wedge B_2$		
$B_1 \wedge \neg B_2$	Executes S_1	
$\neg B_1 \wedge B_2$		
$\neg B_1 \wedge \neg B_2$.		

   b. For what kinds of states σ can statements U_1 and U_2 behave differently?
4. Let $DO \equiv \textbf{\textit{do}}\ B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2 \ \square \ \dots \ \square\ B_n \rightarrow S_n\ \textbf{\textit{od}}$ and $BB \equiv B_1 \vee B_2 \vee \dots B_n$. What property does BB have to have for us to avoid an infinite loop when executing DO?
5. Consider the loop $i := 0;\ \textbf{\textit{do}}\ i < 1000 \rightarrow S_1;\ i := i+1 \ \square\ i < 1000 \rightarrow S_2;\ i := i+1\ \textbf{\textit{od}}$ (where neither S_1 nor S_2 modifies i). Do we know anything about how many times or in what pattern we will execute S_1 vs S_2?
6. What is $M(S, \{x = 1\})$ where $S \equiv \textbf{\textit{do}}\ x \le 20 \rightarrow x := x*2 \ \square\ x \le 20 \rightarrow x := x*3\ \textbf{\textit{od}}$?

7. Consider the loop $x := 1$; **do** $x \geq 1 \rightarrow x := x+1$ □ $x \geq 2 \rightarrow x := x-2$ **od**. Can running it lead to an infinite loop?

8. What are the possible final states of this program? (Assume $n \geq 0$.)

   x:= 0; y:= 0; k := 0; aa := 0; bb := 0;
   do *k < n → x := x+a; k := k+1; aa := aa+1*
   □ *k < n → y := y+b; k := k+1; bb := bb+1*
   od

Problems 9 - 11 all refer to the Array Value Matching problem in the notes (Example 10).

9. In the notes, we approached the problem by asking "*What do we do if b0[k0] < b1[k1]*?" and so on for the other 5 tests. Another way to approach the problem is to ask "*When do we want to increment k0* ?" and so on for the other 2 indexes. If we take this approach, which of the three programs 10(a), 10(b), or 10(c) do we wind up with?
10. With the matching program, we can have an execution sequence that (say) does many increments of *k0*, interspersed with occasional increments of *k1* and *k2*. Rewrite the program so that we do as many increments of *k0* as possible for moving on to *k1*, and so on.
11. Translate program 10(c) into a deterministic language like C, Java, or whatever.

Solution to Activity 7 (Nondeterministic Sequential Programs)

1. Nondeterminism makes it easier to (1) Find and combine partial solutions. (2) Delay considering overlapping cases.
2. (Basic properties of nondeterministic if)
   a. We need $\sigma \vDash BB$, because if $\sigma \vDash \neg BB$, then $M(IF, \sigma) = \{\perp_e\}$. (In English: At least one guard must be true; if none of them are true, we get a runtime error.)
   b. The order of the guarded commands doesn't matter: If more than one guard is true, we nondeterministically choose one element from the set of corresponding statements, and in a set, the elements aren't ordered.
3. (Deterministic vs nondeterministic conditionals) Recall $U_1 \equiv$ ***if*** $B_1 \rightarrow S_1 \square B_2 \rightarrow S_2$ ***fi*** and $U_2 \equiv$ ***if*** B_1 ***then*** S_1 ***else if*** B_2 ***then*** S_2 ***fi***.
   a. Execution of U_1 and U_2:
   b. U_1 and U_2 behave the same when one of B_1 and B_2 is true and the other is false. When both are true, U_2 always executes S_1 but U_1 will execute S_1 or S_2. When both of B_1 and B_2 are false, U_1 yields a runtime error but U_2 does nothing.
4. The nondeterministic ***do-od*** loop halts if *BB* is false at the top of the loop; an infinite loop occurs when *BB* is always true at the top of the loop.
5. Say S_1 is run m times and S_2 is run n times. We know $0 \le m, n \le 1000$ and $m+n = 1000$, but that's all. At each iteration, the choice is nondeterministic (i.e., unpredictable). The choice does not have to be random (like with a coin flip), and the sequence of choices don't have to follow an pattern or distribution or be fair, etc. We can't even assign a probability to any particular sequence of choices (like "always choose S_1").
6. [2023-02-22] {{x = 24}, {x = 27}, {x = 32}, {x = 36}, {x = 48}, {x = 54}}.
   - Since x always has the form 2^n * 3^m, in the last iteration we must have had (for example) x = 8 and multiplied by 3 to get 24. (Having x = 8 and multiplying by 2 to get 16 wouldn't have stopped the loop.)
   - The full possibilities for the last iteration are 8*3 = 24, 16*2 = 32, 16*3 = 48, 12*2 = 24, 12*3 = 36, 18*2 = 36, 18*3 = 54, 9*3 = 27.
   - So altogether we get that x can be 24, 27, 32, 36, 48, or 54.
   - The given answers of x = 12, 16, 18, 24, or 27 were the right ones when the test was x ≤ 10, but I changed the tests to x ≤ 20 and forgot to change the answers, sigh.)
7. It's possible that the loop could run forever. There's no guaranteed fairness in nondeterministic choice, so we could increment *x* by 1 many more times than we decrement it by 2.
8. The states are the ones with k = n; 0 ≤ aa ≤ n, 0 ≤ bb ≤ n, aa+bb=n, x = aa*a + bb*b:

   { {k = n, aa = α, bb = n−α, x = α*a + (n-α)*b} | 0 ≤ α ≤ n} }.
9. Program 10(b).
10. (Do sequences of same increment)

do *b0[k0] < b1[k1] → k0 := k0+1* ***od***;
do *b1[k1] < b2[k2] → k1 := k1+1* ***od***;
do *b2[k2] < b0[k0] → k2 := k2+1* ***od***

11. (Omitted)

