

1 Model Checking

Consider a system $(\mathcal{V}, \theta, \tau)$, where \mathcal{V} is the set of boolean state variables, θ is a predicate on \mathcal{V} that defines the initial condition, and τ is a predicate on $\mathcal{V} \cup \mathcal{V}'$ that defines the transition relation (where as usual $\mathcal{V}' = \{x' \mid x \in \mathcal{V}\}$ is the set of “primed” variables describing the state after the transition).

The system $(\mathcal{V}, \theta, \tau)$ defines a graph, with states $S = (\mathcal{V} \mapsto \{0, 1\})$ (the set of functions from S to $\{0, 1\}$) as states, and with set of edges $E = \{(s, t) \in S \times S \mid (s, t) \models \tau\}$. A predicate B on \mathcal{V} defines a set of states $\llbracket B \rrbracket = \{s \in S \mid s \models B\}$. Denote by d the *diameter* of the graph, i.e., the maximum length over all $s, t \in S$ such that t is reachable from s of the shortest path between s and t , or:

$$d = \max\{\text{length of shortest path from } s \text{ to } t \text{ in } (S, E) \mid s, t \in S \text{ and } t \text{ is reachable from } s \text{ in } (S, E)\}.$$

- $\text{Post}(B, \tau)$ is a predicate on \mathcal{V} defining the set of states that can be reached from a state in $\llbracket B \rrbracket$ in one step;
- $\text{Pre}(B, \tau)$ is a predicate on \mathcal{V} defining the set of states that can reach a state in $\llbracket B \rrbracket$ in one step.

Let H be a predicate on \mathcal{V} defining the error states. To check whether $\llbracket H \rrbracket$ is reachable from $\llbracket \theta \rrbracket$ in the graph (S, E) , you can use the algorithms for forwards and backwards exploration that have been presented in class. However, if you believe the error is present, perhaps it is a better idea to search forward from $\llbracket \theta \rrbracket$, and backward from $\llbracket H \rrbracket$, simultaneously. The idea is that θ and H as predicates are usually represented by a small BDD, while the predicates for the intermediate stages of the search may be (and in general are) larger.

1. Write such an algorithm.
2. Assume that $\llbracket H \rrbracket$ is reachable from $\llbracket \theta \rrbracket$. What is the worst-case running time, measured in terms of the number of times you have to apply the Pre and Post operators, of the
 - forward search algorithm (the one based on Post);
 - backward search algorithm (the one based on Pre);
 - bidirectional search algorithm (the one you wrote)?
3. Assume that $\llbracket H \rrbracket$ is not reachable from $\llbracket \theta \rrbracket$. What is the worst-case running time, measured in terms of the number of times you have to apply the Pre and Post operators, of the
 - forward search algorithm (the one based on Post);
 - backward search algorithm (the one based on Pre);
 - bidirectional search algorithm (the one you wrote)?
4. Assume that you have a function *size* that given a BDD B , gives you the number *size*(B) of nodes in the BDD. Can you use function *size* to improve your bidirectional algorithm?

2 Statecharts

1. Describe in statecharts the interface of a microwave oven. To cook, you can choose the time and the power. Events include: programming power level, programming time (I leave to you the level of detail with which you model button pushes), quick setting (+ 30 seconds), clear, start, stop, door opens, door closes, motor turns on/off, microwave emission turns on/off, clock tick at 1 Hz, interior light goes on/off.
2. Write a few properties that should always be true, for operator safety and for product usability. Brief reminder:
 - The formula $AG\phi$ means that ϕ should hold at every state. Example: $AG(\text{door_open} \rightarrow \text{light_on})$ requires that the light is on when the door is open, assuming that door_open and light_on have been defined as attributes of the states of the statechart.
 - The formula $EF\phi$ means that, on some future path, ϕ will hold. (Hint: this formula can be used to say that there must be a way to do x , for some reasonable x).

3 Software Verification

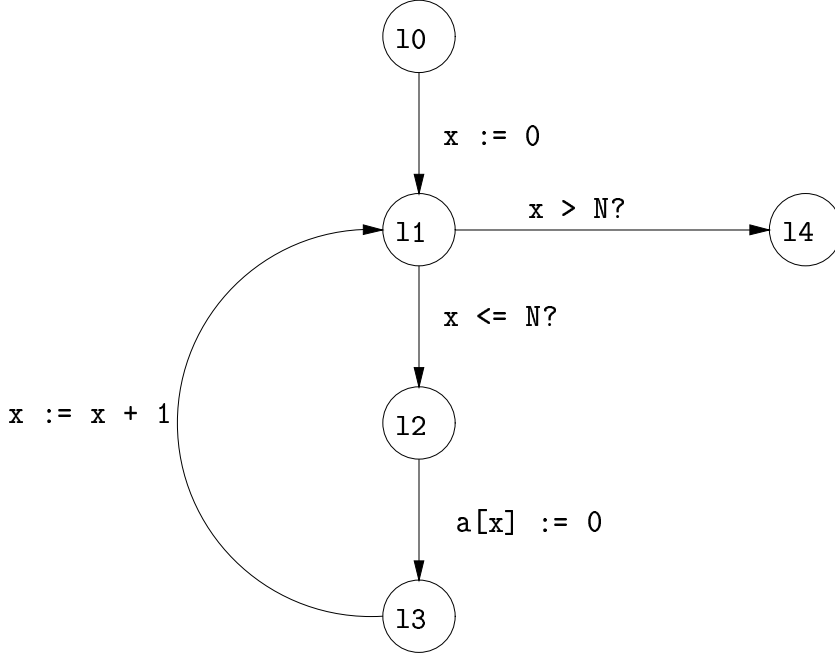
3.1 Warm-up

First, a warm-up example. Consider the following program:

```
a: array [0..N] of integer
x: integer

10: x := 0
11: while x <= N {
12:   a [x] := 0
13:   x := x + 1
   }
14:
```

Where the 10...14 are the program locations. Note that a location *precedes* the corresponding statement: for instance, 13 is just *before* statement $x := x + 1$ is executed. The flow chart of the program is as follows:



A *loop invariant* is a property that is true just before the condition of the while loop is tested, that is, at location 11. In this case, we can guess that $\{0 \leq x \leq N + 1\}$ is a loop invariant.

To verify that $\{0 \leq x \leq N + 1\}$ is a loop invariant, we proceed in stages. The loop invariant is associated with location 11, which is the entrance of the while loop. We proceed in steps:

1. First, we use the rule for assignments to show that the invariant holds the first time we reach 11. Remember that the general form for the rule is $\{\phi[E/x]\}x := E\{\phi\}$. We get:

$$\{0 \leq 0 \leq N + 1\}x := 0\{0 \leq x \leq N + 1\}$$

and since the antecedent always holds at 10, we conclude that $\{0 \leq x \leq N + 1\}$ holds the first time we reach 11.

2. In order to prove that $\{0 \leq x \leq N + 1\}$ holds at 11 after the loop, we can first backtrack to 13 using the rule for assignments again:

$$\{0 \leq x + 1 \leq N + 1\}x := x + 1\{0 \leq x \leq N + 1\}$$

and derive that we must prove that $\{0 \leq x + 1 \leq N + 1\}$ holds at location 13.

3. Since the statement at 12 does not modify x , at location 12 we must again prove that $\{0 \leq x + 1 \leq N + 1\}$.
4. At 11, we assume that the invariant holds, or $\{0 \leq x \leq N + 1\}$. Since we get to 12 only when $x \leq N$, at 12 we know that $\{0 \leq x \leq N + 1 \wedge x \leq N\}$. Then, the proof of our loop invariant is concluded by observing that $\{0 \leq x \leq N + 1 \wedge x \leq N\}$ (what we know holds at 12) implies $\{0 \leq x + 1 \leq N + 1\}$ (what we must prove at 12, see step 3).

Further, the condition $\{0 \leq x \leq N + 1 \wedge x \leq N\}$ at 12 implies also $\{0 \leq x \leq N\}$, so we can conclude that x is always within the array bounds in the statement following 12.

3.2 The question

Consider now the following program:

```
a:  array [0..N] of integer
x,y,c: integer

10: x := 0
11: y := N
12: while x != y {
13:   if a[x] < a[y] {
14:     a[x] := a[y]
   } else {
15:     x := x + 1
16:     y := y - 1
   }
17: }
```

1. Draw the flow chart of the program.
2. Is the program correct? Can it generate array accesses with the index out of bounds? If you believe the program is not correct, propose a fix that changes it “as little as possible”.
3. For your new program, come up with a loop invariant for location 11, and prove that it is a loop invariant.
4. Use your proof to show that all array accesses are within bounds.