15-414: Bug Catching: Automated Program Verification

Lecture Notes on Going Temporal

Matt Fredrikson

Carnegie Mellon University

Lecture 16

1 Introduction

In the previous lecture, we observed a potential difference between whether a data structure invariant holds literally always at all times during all runs of all its operations (which is essentially a prerequisite for uncorrupted concurrent usages) compared to whether the data structure invariant merely holds at the end of each of the operations if it was true before. On second thought, what we have seen so far was, from the very semantics, meant as the dynamic logic for proving properties of (all or some) final states of the program under complete ignorance of what happens in between. There are ways of augmenting dynamic logics to temporal dynamic logics that provide explicit ways of proving formulas that are true, e.g., always throughout an execution.

While this works well and continues the deductive verification principles we saw so far, we will, instead, leverage the motivation of a temporal understanding of programs as a segway into studying temporal logics and their use in model checking.

2 Traces of Programs

If we want to study whether a formula such as a data structure invariant is true all the time always throughout the execution of a program, we will at least have to retain all the states that the program visits in its semantics. One possible approach for that is to make the semantics remember the set of all traces that a program can exhibit where a trace is a sequence of states that the program visited along the way. Another possible approach is to emphasize the structure that the transitions within the program took and then generate possible traces from that. We will do the latter for no particular reason except that we would like to relate this as directly as possible to CTL.

While the technical nuances of the following definitions are somewhat subtle, the main point is quite intuitive. With a little bit of care, we can retain all intermediate states during the execution of a program as an entire trace, instead of just retaining the fact that a particular final state was reachable from a particular initial state by running a program to completion.

A trace is either a finite sequence of states of some length $n \in \mathbb{N}$,

$$(\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_n) \tag{1}$$

or it is an infinite sequence of states, one for each natural number, of length ∞:

$$(\sigma_0, \sigma_1, \sigma_2, \sigma_3, \ldots)$$

A trace terminates iff it is finite and its last state, σ_n in Eq. 1, is not the special failure state Λ, which indicates abortion due to a failed test. The special error state Λ does not provide values for any variables (so no formulas or terms can be evaluated in it), but it is used to mark the end of an aborted execution trace. No program ever continues from the error state Λ. For stylistic reasons, a trace of length n has $n + 1$ states.

Definition 1 (Trace semantics of programs)**.** The *trace semantics, $\tau(\alpha)$, of a program α*, is the set of all its possible traces and is defined inductively as follows:

1. $\tau(x := e) = \{(\omega, \nu) \;:\; \nu = \omega$ except that $\nu(x) = \omega[\![e]\!]$ for $\omega \in \mathcal{S}\}$
2. $\tau(?Q) = \{(\omega) \;:\; \omega \models Q\} \cup \{(\omega, \Lambda) \;:\; \omega \not\models Q\}$
3. $\tau(\texttt{if}(Q)\,\alpha\,\texttt{else}\,\beta) = \{\sigma \in \tau(\alpha) \;:\; \sigma_0 \models Q\} \cup \{\sigma \in \tau(\beta) \;:\; \sigma_0 \not\models Q\}$
4. $\tau(\alpha; \beta) = \{\sigma \circ \varsigma \;:\; \sigma \in \tau(\alpha)\,,\, \varsigma \in \tau(\beta)\}$;
   the composition of $\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots)$ and $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \ldots)$ is
   $$\sigma \circ \varsigma := \begin{cases} (\sigma_0, \ldots, \sigma_n, \varsigma_1, \varsigma_2, \ldots) & \text{if } \sigma \text{ terminates in } \sigma_n \text{ and } \sigma_n = \varsigma_0 \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$
5. $\tau(\texttt{while}(Q)\,\alpha) = \{\sigma^{(0)} \circ \sigma^{(1)} \circ \cdots \circ \sigma^{(n)} \;:\;$ for some $n \geq 0$ such that for all $0 \leq i < n$: ① the loop condition is true $\sigma_0^{(i)} \models Q$ and ② $\sigma^{(i)} \in [\![\alpha]\!]$ and ③ $\sigma^{(n)}$ either does not terminate or it terminates in $\sigma_m^{(n)}$ and $\sigma_m^{(n)} \not\models Q$ in the end$\}$
   $\cup\, \{\sigma^{(0)} \circ \sigma^{(1)} \circ \sigma^{(2)} \circ \ldots \;:\;$ for all $i \in \mathbb{N}$: ① $\sigma_0^{(i)} \models Q$ and ② $\sigma^{(i)} \in [\![\alpha]\!]\}$
   $\cup\, \{(\omega) \;:\; \omega \not\models Q\}$
   That is, the loop either runs a nonzero finite number of times with the last iteration either terminating or running forever, or the loop itself repeats infinitely often and never stops, or the loop does not even run a single time.
6. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$ where $\alpha^{n+1} \overset{\text{def}}{\equiv} (\alpha^n; \alpha)$ for $n \geq 1$, and $\alpha^1 \overset{\text{def}}{\equiv} \alpha$ and $\alpha^0 \overset{\text{def}}{\equiv} (?\mathit{true})$.

All cases in this definition are under the assumption that the respective compositions are defined. For example

$$\tau(\alpha; \beta) = \{\sigma \circ \varsigma \;:\; \sigma \in \tau(\alpha)\,,\, \varsigma \in \tau(\beta) \text{ when } \sigma \circ \varsigma \text{ is defined}\}$$

3 Linear Temporal Logic

Now that we have a set of traces such as the ones $\tau(\alpha)$ generated by a program α, we have more temporal information about the sequence of states that happened during the run of the program. That enables us to talk more about the way how truth-values change over time along such a trace.

Definition 2 (LTL)**.** The formulas of linear temporal logic (LTL) with formulas over individual states F are defined by the following grammar:

$$P, Q ::= F \mid \neg P \mid P \land Q \mid \mathbf{X}P \mid \Box P \mid \Diamond P \mid P\mathbf{U}Q$$

The formula $\Box P$ means that P is always true in the future. The formula $\Diamond P$ means that P is sometimes true in the future, meaning at least at one point. The formula $\mathbf{X}P$ means that P is true in the next state. And the formula $P\mathbf{U}Q$ means that P is true until Q is true (which also will be true at some point).

The suffix of a trace σ starting at step $k \in \mathbb{N}$ is denoted σ^k and only defined if the trace has at least length k. That is

$$(\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_{k-1}, \sigma_k, \sigma_{k+1}, \sigma_{k+2}, \dots)^k = (\sigma_k, \sigma_{k+1}, \sigma_{k+2}, \dots)$$

In particular σ^0 is the same as σ. Also $(\sigma_0) \circ \sigma^1 = \sigma$ if the trace has at least length 1 so that σ^1 is defined.

Definition 3. The truth of LTL formulas in a trace σ is defined inductively as follows:

1. $\sigma \models F$ iff $\sigma_0 \models F$ for a state formula F provided that $\sigma_0 \neq \Lambda$
2. $\sigma \models \neg P$ iff $\sigma \not\models P$, i.e. it is not the case that $\sigma \models P$
3. $\sigma \models P \land Q$ iff $\sigma \models P$ and $\sigma \models Q$
4. $\sigma \models \mathbf{X}P$ iff $\sigma^1 \models P$
5. $\sigma \models \Box P$ iff $\sigma^i \models P$ for all $i \geq 0$
6. $\sigma \models \Diamond P$ iff $\sigma^i \models P$ for some $i \geq 0$
7. $\sigma \models P\mathbf{U}Q$ iff there is an $i \geq 0$ such that $\sigma^i \models Q$ and $\sigma^j \models P$ for all $0 \leq j < i$

In all cases, the truth-value of a formula is, of course, only defined if the respective suffixes of the traces are defined.

For example, $\mathbf{X}P$ only has a truth-value in trace σ if σ^1 is defined, which means that the trace σ has length 1 (recall that this means it has at least 1+1 states). So $\mathbf{X}P$ is neither true or false but simply meaningless in a trace such as σ_0 that does not actually have a next state. Likewise, $\mathbf{X}p$ is meaningful (and either true or false depending on whether p is true in σ_1) in a trace $\sigma = (\sigma_0, \sigma_1)$, but $\mathbf{XX}p$ is not meaningful in the same trace because it's not long enough to have a successor of a successor.

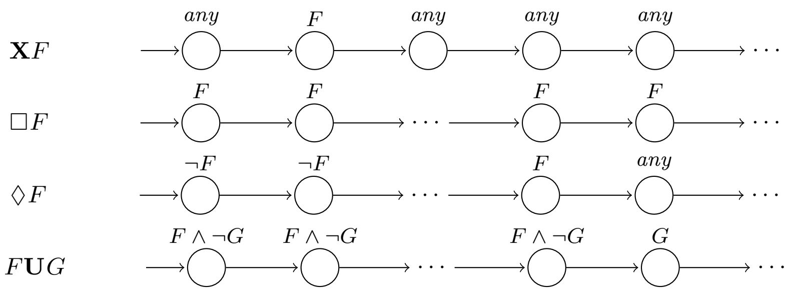

Figure 1: Examples of traces satisfying LTL formulas

Note that the meaning of the box and diamond modalities of LTL is quite analogous to the meaning that the box and diamond modalities already have in dynamic logic. The only difference is that dynamic logic modalities range over the runs of a concrete program while the modalities of LTL range over time (along a fixed trace of something). This is not a coincidence. Both are versions of modal logics, which differ in terms of what the box and diamond modalities range over but are otherwise built similarly.

For the cases $\mathbf{X}P, \square P, \Diamond P, P\mathbf{U}Q$ It is, of course, very important to retain the entire suffix of the trace for the semantics (not just a single state) in case the subformulas P and Q themselves mention further temporal operators. For example, LTL formula

$$\square\Diamond P$$

expresses that P is true infinitely often when referring to an infinite trace. On a finite trace, it merely means that P is true in the last (non-failure) state.

The LTL formula

$$\Diamond\square P$$

expresses that P is eventually true all the time (so is true almost always, so except at finitely many exception states) when referring to an infinite trace. On a finite trace, it also merely means that P is true in the last (non-failure) state.

4 LTL Formulas on Program Traces

The following very clever program solves the issue of subtracting from negative numbers by first turning them into positive numbers and then adding, while ultimately flipping the sign again.

$$\begin{aligned}&x := -x;\\&x := x + 7;\\&x := -x;\end{aligned}$$

This program does correctly subtract 7 from a negative number as witnessed by a corresponding proof of the following dynamic logic formula:

$$x = x_0 \rightarrow [x := -x; x := x + 7; x := -x]\, x = x_0 - 7$$

This formula means that whenever x_0 equals the initial value of variable x then *after* running the program, the resulting value of x will be the result of subtracting 7 from x_0, which, since it didn't change, still is the initial value of x. We can write this as an LTL formula in terms of its traces. In particular, we can say that all traces σ of the program satisfy the following LTL:

$$\sigma \models x = x_0 \rightarrow \Box\Diamond(x = x_0 - 7)$$

Inspecting the semantics of this formula, it says that:

- If $\sigma \models x = x_0$, which is true if and only if $\sigma_0 \models x = x_0$,
- then $\sigma \models \Box\Diamond(x = x_0 - 7)$. In other words, $\sigma^i \models \Diamond(x = x_0 - 7)$ for all $i \geq 0$, so there always exists $j \geq i$ where $\sigma^j \models x = x_0 - 7$.

Because the traces of this program are finite, this imposes the same condition as the DL box modality, that the final state will satisfy $x = x_0 - 7$ when the initial state satisfies $x = x_0$.

The program also satisfies the property that if x is initially negative then x is finally negative:

$$x < 0 \rightarrow [x := -x; x := x + 7; x := -x]\, x < 0$$

But it *does not* satisfy that x is negative always at all times while running the program, because the whole point is that the first assignment flips the sign of x into a positive number. In fact, all traces of this program are of the following form:

$$\tau(x := -x; x := x + 7; x := -x) = \{(\omega, \omega_x^{-\omega(x)}, \omega_x^{-\omega(x)+7}, \omega_x^{-(\omega(x)+7)}) \ : \ \omega \text{ is any state}\}$$

Consequently, if $\sigma \in \tau(x := -x; x := x + 7; x := -x)$ is a trace of this program starting in an initial state σ_0 with negative initial value of x so $\sigma_0(x) < 0$, then the LTL formula $\Box(x < 0)$ is *not* true for it even if it is true initially and in the end. Indeed, all traces $\sigma \in \tau(x := -x; x := x + 7; x := -x)$ of the program satisfy:

$$\sigma \not\models x < 0 \rightarrow \Box(x < 0)$$

That is, the following condition is false for σ:

> if $x < 0$ is true (initially, because there's no temporal operator on the left hand side of the implication), then $x < 0$ is true always in the future (of σ).

But what is, indeed, true for all traces σ of the program is:

$$\sigma \models x < 0 \rightarrow \Box(x \neq 0)$$

That is, if x starts negative then it will always be nonzero at every point in time throughout the entire trace σ.

5 Summary

- A trace is either a finite sequence of states of a given length or an infinite sequence of states;
- A trace terminates iff it is finite and its last state is not a failure state (Λ);
- The trace semantics of a program α ($\tau(\alpha)$) is the set of all its possible traces;
- Linear temporal logic is defined by the following grammar:

$$P, Q ::= p \mid \neg P \mid P \wedge Q \mid \mathbf{X}P \mid \Box P \mid \Diamond P \mid P\mathbf{U}Q$$

  - The formula $\Box P$ means that P is always true in the future;
  - The formula $\Diamond P$ means that P is sometimes true in the future, meaning at least at one point;
  - The formula $\mathbf{X}P$ means that P is true in the next state;
  - The formula $P\mathbf{U}Q$ means that P is true until Q is true (which also will be true at some point).