**15-414: Bug Catching: Automated Program Verification**

# Lecture Notes on Loops

Frank Pfenning

Carnegie Mellon University
Lecture 7
February 8, 2022

## 1 Introduction

At a high level, here is what our formalization efforts have been so far:

1. We defined a formal language of *arithmetic expressions* $e$ and *states* $\omega$ as a map from variables to integer values. The meaning of an expression in a given state is some integer $c$. We used $\omega[\![e]\!] = c$ to define the meaning of expressions by cases based on the form of $e$. This definition may seem superfluous, but it provides a means to interpret the *syntax* of expressions with their mathematical meaning as integers.

2. We defined a formal language of *programs* $\alpha$ with statements for assignment, sequential composition, conditionals, while loops, and guards. Because we would like to allow for cases where programs are nondeterministic the meaning of a program is given as a *relation* between an prestate $\omega$ and a poststate $\nu$, written as $\omega[\![\alpha]\!]\nu$. This is defined by analyzing the structure of the program $\alpha$.

3. We defined a formal language of *formulas* $P$ with the usual logical operators like conjunction, disjunction, implication, negation and also universal and existential quantification over integers. The basic formulas are equality and inequality of expressions. Formulas do double duty: they are used in our language of programs for conditionals, while loops, and guards, but they are also intended to be used for *reasoning about* the meanings of programs. The meaning of formulas is defined mathematically with $\omega \models P$ which tells us when the formula $P$ is true in state $\omega$.

4. We add to the language of formulas two modal operators that reference programs, $[\alpha]P$ ($P$ is true in every possible poststate of $\alpha$) and $\langle\alpha\rangle P$ ($P$ is true in some post-

state of $\alpha$). We extend the definition for the meaning of formulas to account for the new constructs. We call this new logic *(deterministic) dynamic logic*.

5. In order to facilitate reasoning, we developed some *axioms* that allow us to break down formulas $[\alpha]P$ that speak about programs $\alpha$ into properties of subprograms of $\alpha$. As axioms they must be *valid*, that is, be true in all possible states.

The tasks remaining after the last lecture are axioms for assignments and while loops. Also, we only gave one small example of an axiom for $\langle\alpha\rangle P$, focusing on reasoning about partial correctness instead of total correctness. In this lecture we first analyze assignments and then loops, both with respect to partial correctness, that is, writing axioms for $[\alpha]P$.

Informally, we already understand how this is done, for example, in Why3. We hypothesize a *loop invariant* and then prove that (a) it holds initially, (b) it is preserved, and (c) it implies the postcondition. To arrive at this form of reasoning for loops will take us all lecture, partly because we take a detour via *nondeterministic dynamic logic*. It turns out this formalism allows us to explore the issues surrounding loops in a simplified form to understand the essence of the problem and its solution. We then port the solution back.

**Learning goals.** After this lecture, you should be able to:

- Express and prove programs in (nondeterministic) dynamic logic
- Reason about repetition in dynamic logic
- Justify reasoning principles for loops as modal axioms
- Prove while loops in (deterministic) dynamic logic

## 2 Summary: Axioms for Dynamic Logic So Far

$$\begin{array}{lcl} [\alpha\,;\,\beta]Q & \leftrightarrow & [\alpha][\beta]Q \\ [?P]Q & \leftrightarrow & (P \to Q) \\ [\text{if } P\,\alpha\,\beta]Q & \leftrightarrow & (P \to [\alpha]Q) \land (\neg P \to [\beta]Q) \end{array}$$

## 3 Assignment

The first instinct might be the following axiom for assignment

$$[x \leftarrow e]P \leftrightarrow (x = e \to P) \qquad \text{(WRONG)}$$

However, this is *not valid* and could therefore lead to unsound reasoning. The cause is the same as why in the informal generation of verification conditions we modeled assignment by creating a fresh "primed" variable. For example, the following is certainly not valid

$$\not\models x = 3 \to [x \leftarrow x + 1](x = 17)$$

since computing $x \leftarrow x+1$ will set $x$ to $4$ but the postcondition requires $x$ to be $17$. With the wrong axiom we could prove

$$(x = 3 \rightarrow [x \leftarrow x + 1]x = 17) \leftrightarrow (x = 3 \rightarrow ((x = x + 1) \rightarrow x = 17))$$

and the right-hand side is true since $x = x + 1$ is contradictory.

There are two ways out: one is to *carefully* substitute $e$ for $x$ with a so-called *uniform substitution*. The other is to *rename* the variable $x$, something we also did when generating a verification condition for a loop. This is handled by quantification over a fresh variable that does not occur in $P$. We write $P(x)$ for a formula $P$ with (possible) occurrences of $x$, and then $P(x')$ for the result of renaming all occurrences of $x$ to $x'$. Then our axiom becomes

$$[x \leftarrow e]P(x) \leftrightarrow (\forall x'.\, x' = e \rightarrow P(x'))$$

It is important for soundness that $x'$ is a variable that does not already occur in $e$ or $P(x)$. We often refer to this as a "fresh variable".

Our example no longer gives us a contradiction, because

$$(x = 3 \rightarrow [x \leftarrow x + 1]x = 17) \leftrightarrow (x = 3 \rightarrow \forall x'.\, x' = x + 1 \rightarrow x' = 17)$$

is false as it should be.

Let's use our swap program as an example for generating a verification condition using the two axioms we already have. We would like to to prove

$$x = a \land y = b \rightarrow [x \leftarrow x + y\ ;\ y \leftarrow x - y\ ;\ x \leftarrow x - y]x = b \land y = a$$

We use the axiom for sequential composition twice, to reduce this to

$$x = a \land y = b \rightarrow [x \leftarrow x + y]([y \leftarrow x - y]([x \leftarrow x - y](x = b \land y = a)))$$

Now we can use the axiom for assignment (and pull out the quantifier)

$$x = a \land y = b \rightarrow x' = x + y \rightarrow [y \leftarrow x' - y]([x' \leftarrow x' - y](x' = b \land y = a))$$

We use it once more

$$x = a \land y = b \rightarrow x' = x + y \rightarrow y' = x' - y \rightarrow [x' \leftarrow x' - y'](x' = b \land y' = a)$$

and a third time

$$x = a \land y = b \rightarrow x' = x + y \rightarrow y' = x' - y \rightarrow x'' = x' - y' \rightarrow (x'' = b \land y' = a)$$

At this point we have eliminated the programs and have a formula in pure arithmetic. This is the *verification condition* for the original program that no longer references any code. Substituting out the assumptions we find $x' = a + b$, $y' = a$, $x'' = b$ so the conclusion $x'' = b \land y' = a$ is true and the whole formula is valid.

## 4 While Loops

It is easy to come up with an axiom, based on the intuition for conditionals and sequences, embodying the semantics of the while loop.

$$[\mathsf{while}\, P\, \alpha]Q \leftrightarrow (P \rightarrow [\alpha][\mathsf{while}\, P\, \alpha]Q) \wedge (\neg P \rightarrow Q)$$

Unfortunately, this does not reduce the complexity of the program, since $\mathsf{while}\, P\, \alpha$ reappears on the right-hand side.

In the next lecture we learn how to address this issue and come up with a better axiom to reason about while loops.

## 5 Nondeterministic Dynamic Logic

What we call nondeterministic dynamic logic is what most sources just call "dynamic logic". The idea is to replace the conditionals by nondeterministic choice $\alpha \cup \beta$, and while loops by nondeterministic repetition $\alpha^*$.

$$\text{Programs} \quad \alpha, \beta \quad ::= \quad x \leftarrow e \mid \alpha \,;\, \beta \mid ?P \mid \alpha \cup \beta \mid \alpha^*$$

The nondeterministic choice $\alpha \cup \beta$ executes either $\alpha$ or $\beta$. The repetition $\alpha^*$ executes $\alpha$ one of $0, 1, 2, \ldots$ number of times in succession.

This is an effort to reduce conditionals and while loops to simpler building blocks and also exploit what has been learned in the study of regular expressions and Kleene algebra.

The formal semantics of these is a straightforward simplification of the semantics for conditionals and while loops. That's because we already *set up* the semantics to be relation between states.

| | |
|---|---|
| $\omega[\![\alpha \cup \beta]\!]\nu$ | iff $\omega[\![\alpha]\!]\nu$ or $\omega[\![\beta]\!]\nu$ |
| $\omega[\![\alpha^*]\!]\nu$ | iff there exists an $n \geq 0$ such that $\omega[\![\alpha]\!]^n\nu$ |
| $\omega[\![\alpha]\!]^0\nu$ | iff $\omega = \nu$ |
| $\omega[\![\alpha]\!]^{n+1}\nu$ | iff there exists $\mu$ such that $\omega[\![\alpha]\!]\mu$ and $\mu[\![\alpha]\!]^n\nu$ |

We can express the original conditionals while loops using guards and the new constructs.

$$\begin{array}{lcl} \mathsf{if}\, P\, \alpha\, \beta & \triangleq & (?P \,;\, \alpha) \cup (\neg ?P \,;\, \beta) \\ \mathsf{while}\, P\, \alpha & \triangleq & (?P \,;\, \alpha)^* \,;\, ?\neg P \end{array}$$

You should convince yourself that the left-hand and right-hand sides of these notational definitions have the same meaning, that is, they relate the same states $\omega[\![-]\!]\nu$.

Furthermore, we can capture the meaning with new axioms, again simplifying the old ones for conditionals and while loops.

$$\begin{array}{lcl} [\alpha \cup \beta]P & \leftrightarrow & [\alpha]P \wedge [\beta]P \\ [\alpha^*]P & \leftrightarrow & P \wedge [\alpha][\alpha^*]P \end{array}$$

We observe that the axiom for repetition $\alpha^*$ has the same flaw as the axiom for while loops.

## 6 The Induction Axiom for Repetition

As a simple example of repetition, consider

$$[?(n = 0) \; ; \; (n \leftarrow n + 2)^*]\mathsf{even}(n)$$

The even predicate is easy to define in arithmetic ($\mathsf{even}(n) \triangleq \exists k.\, 2k = n$). We see that the property above should hold, because no matter how often $n$ is incremented by 2 it will always remain even. The question is how to prove that in dynamic logic—mathematically we can always do an induction over the number of iterations. We can break off the precondition so it becomes

$$n = 0 \rightarrow [(n \leftarrow n + 2)^*]\mathsf{even}(n)$$

An attempt might be

$$[\alpha^*]Q \stackrel{?}{\leftrightarrow} Q \wedge (Q \rightarrow [\alpha]Q)$$

with the idea that $Q$ on the right-hand side expresses that is must be true initially, and that $Q \rightarrow [\alpha]Q$ shows that $Q$ is preserved by one iteration of the loop. Unfortunately, this gives us exactly that—it does not show that $Q$ is preserved by an arbitrary number of iterations. In order to get that, we need to say that, after an arbitrary number of iterations, $Q$ is still preserved by one more iteration.

$$[\alpha^*]Q \leftrightarrow Q \wedge [\alpha^*](Q \rightarrow [\alpha]Q)$$

This seems plausible, but it suffers from the same defect we had worried about before: $\alpha^*$ appears on both sides. Please, have some faith in me for a moment that we'll be able to address that while we show that this axiom is indeed valid.

**Proof.** While not strictly necessary, it is perhaps easiest to understand the proof if we reformulate it in mathematics as follows:

> (For all $n \geq 0$, $\omega \models [\alpha^n]Q$)
> iff
> ($\omega \models Q$ and for all $k \geq 0$, $\omega \models [\alpha^k](Q \rightarrow [\alpha]Q)$

Here $[\alpha^n]Q$ means that $Q$ is true after $n$ iterations of $\alpha$. We prove each direction separately.

"$\longrightarrow$"

Assume for all $n \geq 0$ we have $\omega \models [\alpha^n]Q$ (1)
To show $\omega \models Q$ we use (1) for $n = 0$.
It remains to show that $\omega \models [\alpha^k](Q \rightarrow [\alpha]Q)$.
For that, it is sufficient to prove $\omega \models [\alpha^k][\alpha]Q$ (ignore the additional assumption $Q$).
But that's the same as $\omega \models [\alpha^{k+1}]Q$ which follows from (1) with $n = k + 1$.

"$\longleftarrow$"

Assume $\omega \models Q$ (1)
  and for all $k \geq 0$, $\omega \models [\alpha^k](Q \to [\alpha]Q)$. (2)
We prove by induction on $n$ that for all $n \geq 0$, $\omega \models [\alpha^n]Q$.
Base: $n = 0$. Then $[\alpha^n]Q = Q$ and $\omega \models Q$ is exactly (1).
Step: $n = m + 1$. Assume $\omega \models [\alpha^m]Q$. (3)
We use (2) with $k = m$ to obtain $\omega \models [\alpha^m](Q \to [\alpha]Q)$.
The modality distributes over implication (see below)
  so we obtain $\omega \models [\alpha^m]Q$ implies $\omega \models [\alpha^m][\alpha]Q$.
From this implication and (3), we get $\omega \models [\alpha^m][\alpha]Q$
  and that is the same as $\omega \models [\alpha^{m+1}]Q$.
This is what we needed to complete the induction step.

It is easy to show that the axiom

$$[\alpha](P \to Q) \to ([\alpha]P \to [\alpha]Q)$$

is valid, that is, the box modality distributes over implication: if in every poststate of $\alpha$ we have both $P \to Q$ and $P$, then we also have $Q$ in the same poststate.

## 7 Validity and Loop Invariants

We now return to the induction axiom—thank you for your patience!

$$[\alpha^*]Q \leftrightarrow Q \wedge [\alpha^*](Q \to [\alpha]Q)$$

How can we actually use this? Let's think back to the early lectures and loop invariants. We verified that the invariant (here $Q$) is true initially (the proof of $Q$ on the right-hand side), and then we verified the loop invariant is preserved *forgetting the concrete information we had when we first arrived at the loop*. Here, this would correspond to proving that $Q \to [\alpha]Q$ is *valid*, which means it is true for any state. This is important because we want to show the preservation of $Q$ no matter how many times we have already been around the loop.

In order to express this kind of reasoning *as an axiom* we need to be able to say that "$P$ *is valid*" inside the logic. This is the purpose of the necessity modality $\Box P$, which was actually inspiration for $[\alpha]P$ in dynamic logic, except $P$ has to be *true for any state*, not just for the poststates of $\alpha$.

$$\text{Formulas} \quad P \quad ::= \quad e_1 \leq e_2 \mid \ldots \mid [\alpha]P \mid \langle\alpha\rangle P \mid \Box P$$

We define

$$\omega \models \Box P \text{ iff } \nu \models P \text{ for any } \nu$$

We then can prove an axiom

$$\Box P \to [\alpha]P$$

Our axiom for reasoning with invariants then becomes

$$[\alpha^*]Q \leftarrow Q \wedge \Box(Q \to [\alpha]Q)$$

This is no longer a bi-implication, but only a right-to-left implication ($Q \leftarrow P$ means $Q$ is implied by $P$). That's because there are other ways to prove a loop (for example, unrolling it a finite number of times). The new axiom is clearly sound, which we can establish directly:

$[\alpha^*]Q \leftrightarrow (Q \wedge [\alpha^*](Q \rightarrow [\alpha]Q))$
and $\quad (Q \wedge [\alpha^*](Q \rightarrow [\alpha]Q)) \leftarrow (Q \wedge \Box(Q \rightarrow [\alpha]Q))$

Returning to our earlier example, we can now prove

$$n = 0 \rightarrow [(n \leftarrow n + 2)^*]\mathsf{even}(n)$$

by reducing it to

$$n = 0 \rightarrow \mathsf{even}(n) \wedge \Box(\mathsf{even}(n) \rightarrow [n \leftarrow n + 2]\mathsf{even}(n))$$

Critically, there is no longer any iteration involved, and we can eliminate the remaining references to programs. The first conjunct is easy since $\mathsf{even}(0)$. Then we have to prove

$$n = 0 \rightarrow \Box(\mathsf{even}(n) \rightarrow [n \leftarrow n + 2]\mathsf{even}(n))$$

Because we have to show *validity*, we lose the assumption $n = 0$. Using the axiom for assignment, this comes down to

$$\mathsf{even}(n) \rightarrow (\forall n'.\, n' = n + 2 \rightarrow \mathsf{even}(n'))$$

which is clearly valid, that is, true for any value of $n$. We can even map this back to plain arithmetic as the obvious

$$\forall n.\, \mathsf{even}(n) \rightarrow (\forall n'.\, n' = n + 2 \rightarrow \mathsf{even}(n'))$$

## 8 Strengthening the Loop Invariant

As your experience with Why3 has undoubtedly shown, we sometimes need to strengthen the loop invariant to make our verifications go through. This is the same phenomenon as having to generalize an induction hypothesis. Let's return to everyone's favorite example, the computation of Fibonacci numbers. This time, we write a nondeterministic loop.

$$[a \leftarrow 0 \,;\, b \leftarrow 1 \,;\, (a, b \leftarrow b, a + b)^*](\exists i.\, a = \mathsf{fib}(i))$$

To save space, we used the simultaneous assignment of $b$ to $a$ and $a + b$ to $b$ (which, by the way, is available in Why3 and is a simple shorthand). It would be nice to say exactly which Fibonacci number we have computed, so also we also compute $i$.

$$[a \leftarrow 0 \,;\, b \leftarrow 1 \,;\, i \leftarrow 0 \,;\, (a, b \leftarrow b, a + b \,;\, i \leftarrow i + 1)^*]a = \mathsf{fib}(i)$$

After a couple of steps of proof, we are left with

$$a = 0 \wedge b = 1 \wedge i = 0 \rightarrow [(a, b \leftarrow b, a + b \,;\, i \leftarrow i + 1)^*]a = \mathsf{fib}(i)$$

Unfortunately, we cannot prove this now, because our loop invariant $a = \mathsf{fib}(i)$ is too weak. We also need to know that $b = \mathsf{fib}(i+1)$.

Before we finish the example, let's consider how we prove $[\alpha^*]Q$ more generally. We want to be able to use an *arbitrary* loop invariant $J$ and then show three properties: $J$ is true initially, $J$ is preserved by the loop, and $J$ implies the postcondition. The last two properties require validity.

$$[\alpha^*]Q \leftarrow J \wedge \Box(J \rightarrow [\alpha]J) \wedge \Box(J \rightarrow Q)$$

For

$$J_{\mathsf{fib}} = (a = \mathsf{fib}(i) \wedge b = \mathsf{fib}(i+1))$$

we obtain the following three proof obligations.

| | |
|---|---|
| $a = 0 \wedge b = 1 \wedge i = 0 \rightarrow J_{\mathsf{fib}}$ | true initially |
| $\Box(J_{\mathsf{fib}} \rightarrow [(a, b \leftarrow b, a+b \,;\, i \leftarrow i+1)]J_{\mathsf{fib}})$ | preserved |
| $\Box(J_{\mathsf{fib}} \rightarrow a = \mathsf{fib}(i))$ | implies postcondition |

## 9 Back to While Loops

Recall the definition

$$\mathsf{while}\, P\, \alpha \quad \triangleq \quad (?P \,;\, \alpha)^* \,;\, ?\neg P$$

We can plug this in to the axiom we have for repetition and reason, assuming we have settled on a loop invariant $J$.

$$\begin{array}{ll} [\mathsf{while}\, P\, \alpha]Q & \leftrightarrow [(?P \,;\, \alpha)^* \,;\, ?\neg P]Q \\ & \leftrightarrow [(?P \,;\, \alpha)^*][?\neg P]Q \\ & \leftrightarrow [(?P \,;\, \alpha)^*](\neg P \rightarrow Q) \\ & \leftarrow J \wedge \Box(J \rightarrow [?P \,;\, \alpha]J) \wedge \Box(J \rightarrow (\neg P \rightarrow Q)) \\ & \leftrightarrow J \wedge \Box(J \rightarrow (P \rightarrow [\alpha]J)) \wedge \Box(J \wedge \neg P \rightarrow Q) \\ & \leftrightarrow J \wedge \Box(J \wedge P \rightarrow [\alpha]J) \wedge \Box(J \wedge \neg P \rightarrow Q) \end{array}$$

This expresses logically and concisely what we have studied earlier regarding reasoning about while loops with loop invariants. It does not yet cover *total correctness*, that is, reasoning about *variants* and the termination of loops. We will return to them in a future lecture.

## 10 Aside: Regular Expressions Revisited[1]

[1] not covered in lecture

Except for assignment, we can recognize that regular expressions are related to programs in dynamic logic as shown in the following table of correspondences.

| Regular Expression | Dynamic Logic |
|---|---|
| $r \cdot s$ | $\alpha \,;\, \beta$ |
| $1$ | $\mathsf{skip} \quad (= ?\mathsf{true})$ |
| $r + s$ | $\alpha \cup \beta$ |
| $0$ | $\mathsf{abort} \quad (= ?\mathsf{false})$ |
| $r^*$ | $\alpha^*$ |
| $a$ | ?? |
| ?? | $x \leftarrow e$ |

The prestate would be the input word and the poststate the remaining word after the program (= regular expression) has matched an initial segment of the word. We see there are no general guards in regular expressions, and the effect of an assignment has been replaced by the effect of reading a character in the input word.