15-414: Bug Catching: Automated Program Verification

Lecture Notes on Dynamic Logic & Compositional Reasoning

Matt Fredrikson[*]

Carnegie Mellon University

Lecture 7, 8

February 7, 2023

1 Introduction

We continue on our path towards formalization of various necessary components for program reasoning: a language of arithmetic expressions e, a language of formulas P, and a language is simple while programs α. We gave a mathematical semantics of each relative to an assignment ω of integers to variables. The meaning of an expression $\omega[\![e]\!] \in \mathbb{Z}$ is an integer, a formula can be true $\omega \models P$ (or false $\omega \not\models P$), and the meaning of a program is a relation $\omega[\![\alpha]\!]\nu$ between prestates ω and poststates ν. At this point we can reason about programs *semantically*, that is, mathematically reason about the meaning of programs.

None of this gives us a *logic* for reasoning about programs: all reasoning is reduced back to general mathematics. It is therefore difficult to mechanize and automate since general mathematics is difficult to mechanize and automate.

The goal of today's lecture is to develop a *logic* for reasoning about programs called *dynamic logic*. In this lecture we take yet again a *semantic* approach, that is, we specify when a formula that expresses properties of programs is true. But we also write *axioms* to capture in syntax how we can reason in dynamic logic.

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

- Interpret the meaning of formulas in dynamic logic (DL)

[*] Closely adapted from notes written by Frank Pfenning in Spring 2022

- Determine if simple formulas are valid
- Validate DL axioms against the semantics of DL programs
- Design semantics and axioms for simple language extensions

2 The Key Idea: Boxes and Diamonds

Now we proceed to developing dynamic logic (DL), a logic to reason about programs in our little while-loop language. The key idea of dynamic logic is to add two new kinds of formulas.

$$\text{Formulas} \quad P, Q \quad ::= \quad \ldots \mid [\alpha]P \mid \langle\alpha\rangle P$$

It is surprisingly easy to define the meaning of these new formulas. $[\alpha]P$ means that in *every* poststate reachable by α, the formula P is true. And $\langle\alpha\rangle P$ means that in *some* poststate reachable by α, the formula P is true.

For those familiar with *modal logic*: this harkens back to the meaning of $\Box P$ (P is true in every reachable world) and $\Diamond P$ (P is true in some reachable world). The difference here is that the reachable worlds are concretely determined by programs α and not just by a fixed reachability relation. Because of this strong analogy, we may pronounce $[\alpha]P$ as "*box* α *P*" and $\langle\alpha\rangle P$ as "*diamond* α *P*".

We define the meaning of the new constructs rigorously as a relation:

$$\begin{aligned} \omega \models [\alpha]P &\quad \text{iff for every } \nu,\ \omega[\![\alpha]\!]\nu \text{ implies } \nu \models P \\ \omega \models \langle\alpha\rangle P &\quad \text{iff there exists a } \nu \text{ such that } \omega[\![\alpha]\!]\nu \text{ and } \nu \models P \end{aligned}$$

Let's discover whether certain simple formulas are true or not. We may want to recall the definition of the meaning of programs $[\![\alpha]\!]$ in order to apply it to the following questions.

$$\begin{gathered} \omega \models [\mathsf{while}\,\mathsf{true}\,\alpha]P \\ \omega \models \langle\mathsf{while}\,\mathsf{true}\,\alpha\rangle P \\ \omega \models [?\mathsf{true}]P \\ \omega \models \langle ?\mathsf{true}\rangle P \\ \omega \models [\alpha]\mathsf{false} \\ \omega \models \langle\alpha\rangle\mathsf{false} \end{gathered}$$

We suggest you try before moving on to the next page.

$\omega \models [\mathsf{while}\,\mathsf{true}\,\alpha]P$	always
$\omega \models \langle\mathsf{while}\,\mathsf{true}\,\alpha\rangle P$	never
$\omega \models [?\mathsf{true}]P$	iff $\omega \models P$
$\omega \models \langle ?\mathsf{true}\rangle P$	iff $\omega \models P$
$\omega \models [\alpha]\mathsf{false}$	iff α does not terminate (has no poststate)
$\omega \models \langle\alpha\rangle\mathsf{false}$	never

Even though truth depends on a state ω, there are many formulas whose truth does not depend on ω at all. In the example, the truth of the first, second, and last are independent of the ω. Formulas that are true in any state are called *valid*. In the examples above, only the first one is valid. We write

$$\models P$$

to express that P is valid. Here are some other examples:

$\omega[x \mapsto 3] \models [x \leftarrow x+1](x=4)$	true
$\models [x \leftarrow 4](x=4)$	valid
$\models x=3 \rightarrow [x \leftarrow x+1](x=4)$	valid

3 Determinism

We call a program *deterministic* if in any prestate it has at most one poststate. We call a *language* deterministic if every program in it is deterministic. The DL programming language we have shown so far is *deterministic* in this sense. One could prove this rigorously by induction over the structure of the program.

Expressed more mathematically, we say α is *deterministic* if for every ω, $\omega[\![\alpha]\!]\nu$ and $\omega[\![\alpha]\!]\nu'$ imply $\nu = \nu'$. A language is deterministic if every program in the language is deterministic.

Deterministic languages satisfy certain properties that are not true for languages in general. Here is one:

For a deterministic language, $\models \langle\alpha\rangle P \rightarrow [\alpha]P$ for any program α and formula P.

We have used here the notation $\models Q$ (omitting the state ω) to express that Q is valid. This property can easily be proved by appeal to the meaning of formulas and the definition of determinism.

In a deterministic language we say that $\langle\alpha\rangle P$ establishes *total correctness* (the program α has to satisfy terminate and satisfy the postcondition P), while $[\alpha]P$ establishes *partial correctness* (if α has to satisfy P, but only if it terminates).

In Why3, we can establish *total correctness* by specifying *variant* contracts that ensure termination and *partial correctness* by using the *diverges* contracts to allow nontermination.

In a deterministic language, to establish total correctness $\langle\alpha\rangle P$ we can first prove partial correctness $[\alpha]P$ and then separately prove termination.

In the remainder of today's lecture we focus on *partial correctness* and therefore the $[\alpha]P$ modality.

4 Axioms

Now that we have a logic with a suitable semantics our next task is to develop some tools for reasoning *within* the logic. Let's think back to how we reasoned about regular expressions in the previous lecture. For each form of regular expression we wrote down an *axiom* specifying its meaning in a way the theorem provers could use. A critical idea in that case study was to make sure we break down the question if $w \in \mathcal{L}(r)$ to some $w' \in \mathcal{L}(r')$ where r' is a subexpression of r. This allows the theorem prover to break down questions about complicated regular expressions into simpler ones.

We'll follow the same strategy here: write down axioms for $[\alpha]P$ that help us break down the structure of the program α by logical reasoning without explicitly appealing to the semantics any more. Of course, the axioms themselves must be justified in terms of the underlying semantics: we don't want to conclude something that is not true!

Because they are *axioms* we need them to be *valid*, not just true in some particular state or even classes of states. We now go through the language constructs one by one, devising axioms.

4.1 Sequential Composition

Which axiom might describe $[\alpha\ ;\ \beta]P$ in terms of $[\alpha]\_$ and $[\beta]\_$? Recall the meaning:

$$\omega\llbracket\alpha\ ;\ \beta\rrbracket\nu \quad \text{iff} \quad \text{there exists } \mu \text{ such that } \omega\llbracket\alpha\rrbracket\mu \text{ and } \mu\llbracket\alpha\rrbracket\nu$$

Intuitively, P is true after α and β if $[\beta]P$ is true after α. So we propose the axiom

$$[\alpha\ ;\ \beta]P \leftrightarrow [\alpha]([\beta]P)$$

In order to prove that this axiom is valid we can decompose it into two implications. We prove the first one of these.

We want to show that $\omega \models [\alpha\ ;\ \beta]P \rightarrow [\alpha]([\beta]P)$
Assume $\omega \models [\alpha\ ;\ \beta]P$ (1)
and show $\omega \models [\alpha]([\beta]P)$
By definition, this holds if for every μ, $\omega\llbracket\alpha\rrbracket\mu$ implies $\mu \models [\beta]P$
So assume $\omega\llbracket\alpha\rrbracket\mu$ for an arbitrary μ (2)
It remains to show that $\mu \models [\beta]P$
By definition, this holds if for every ν, $\mu\llbracket\beta\rrbracket\nu$ implies $\nu \models P$.
So assume $\mu\llbracket\beta\rrbracket\nu$ for an arbitrary ν (3)
It remains to show that $\nu \models P$ (*)
From (2) and (3) we conclude $\omega\llbracket\alpha\ ;\ \beta\rrbracket\nu$ by definition of $\llbracket\alpha\ ;\ \beta\rrbracket$ (4)
From (1) and (4) we conclude $\nu \models P$ by definition of $\omega \models [\alpha\ ;\ \beta]P$
This conclusion is exactly what we needed to show (*)

The other direction works similarly, essentially just unfolding definitions and some shallow logical reasoning.

4.2 Tests

Recall that $\omega[\![?P]\!]\nu$ iff $\omega = \nu$ if $\omega \models P$.

By definition, then, $\omega \models [?P]Q$ iff for all ν with $\omega[\![?P]\!]\nu$ we have $\nu \models Q$. This requires that Q is true in ω if P is, and imposes no requirement if P is false. Therefore, the right axiom is

$$[?P]Q \leftrightarrow (P \to Q)$$

In the case of tests, let's also consider $\langle ?P\rangle Q$. Recall that $\omega \models \langle ?P\rangle Q$ iff there is exists a ν with $\omega[\![?P]\!]\nu$ such that $\nu \models Q$. But that can only be the case if both P and Q are true in ω. So:

$$\langle ?P\rangle Q \leftrightarrow (P \land Q)$$

4.3 Conditionals

Conditionals are straightforward given the intuition we have built up so far. $[\mathsf{if}\ P\ \alpha\ \beta]Q$ should be true if P is true and $[\alpha]Q$, or P is false and $[\beta]Q$.

$$[\mathsf{if}\ P\ \alpha\ \beta]Q \leftrightarrow (P \to [\alpha]Q) \land (\neg P \to [\beta]Q)$$

Assignments and while loops are trickier, so we postpone introducing axioms for them to the next lecture.

5 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

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

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

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

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

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

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

$$x = a \wedge y = b \to [x \leftarrow x + y \,;\, y \leftarrow x - y \,;\, x \leftarrow x - y]x = b \wedge y = a$$

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

$$x = a \wedge y = b \to [x \leftarrow x + y]([y \leftarrow x - y]([x \leftarrow x - y](x = b \wedge y = a)))$$

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

$$x = a \wedge y = b \to x' = x + y \to [y \leftarrow x' - y]([x' \leftarrow x' - y](x' = b \wedge y = a))$$

We use it once more

$$x = a \wedge y = b \to x' = x + y \to y' = x' - y \to [x' \leftarrow x' - y'](x' = b \wedge y' = a)$$

and a third time

$$x = a \wedge y = b \to x' = x + y \to y' = x' - y \to x'' = x' - y' \to (x'' = b \wedge y' = a)$$

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 $x'' = b \wedge y' = a$ is true and the whole formula is valid.

6 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.

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

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.

References