Lecture Notes on
Semantics of Programs

15-316: Software Foundations of Security & Privacy
Frank Pfenning

Lecture 4
September 5, 2024

1 Introduction

We begin by completing our initial tour of dynamic logic. Then we develop a semantics of programs so we can prove our rules sound and invertible (where it applies) with respect to the given semantic definition.

Unfortunately, the second part of this lecture is a bit technical, with a lot of new notation. We go into this level of rigor because it is important to understand the semantics of programs and how they relate to formal proofs. Only in this way can we be confident that the verification of a programs actually guarantees safety and other properties. It also provides the background and tools to consider extensions, variations, and implementations.

2 Loops

How does a loop **while** $P\ \alpha$ execute? If P is true then we execute the loop body α once, followed again by **while** $P\ \alpha$. If P is false we just exit the loop. The following rule suggests itself:

$$\frac{\Gamma \vdash [\mathbf{if}\ P\ \mathbf{then}\ (\alpha\ ;\ \mathbf{while}\ P\ \alpha)\ \mathbf{else}\ \mathbf{skip}]Q, \Delta}{\Gamma \vdash [\mathbf{while}\ P\ \alpha]Q, \Delta}\ [\mathbf{unwind}]R$$

Here we made up a new program **skip** that doesn't do anything. It behaves like the unit of parallel composition in that **skip** ; α is equivalent to α. We could use $x := x$ instead, but that seems more complicated because it mentions a variable.

The problem with our first rule is that it replaces a program with a larger one, so it is not reductive. We can use the rules we already have to simplify it a bit to

$$\frac{\Gamma, P \vdash [\alpha]([\mathbf{while}\ P\ \alpha])Q, \Delta \quad \Gamma, \neg P \vdash Q, \Delta}{\Gamma \vdash [\mathbf{while}\ P\ \alpha]Q, \Delta}\ [\mathbf{unfold}]R$$

This is better, but in the first premise we still have to reason about exactly the same program. So while these two rules are sound, their application is somewhat limited as we will see in the next lecture.

If you think back to 15-122 *Principles of Imperative Computation* you may remember how we reasoned about loops: we used *loop invariants*. In that course, loop invariants (like pre- and post-conditions for functions) where themselves *executable*. Here they are formulas and subject to logical reasoning. How do loop invariants work? Let's look at a trivial program:

$$\textbf{while}\ (x > 1)\ x := x - 2$$

under the precondition that (say) $x \geq 6$. After the loop we know that if the initial x was even, then in the poststate x must be 0, and if the initial x was odd, then in the poststate x must be 1. For safety properties that may a bit specific, so here we only want to ascertain that $0 \leq x \leq 1$ in the poststate.

In dynamic logic we express this as the proposition

$$x \geq 6 \rightarrow [\textbf{while}\ (x > 1)\ x := x - 2]\, 0 \leq x \leq 1$$

But how do we prove it? What is the loop invariant? Recall:

- The loop invariant must be true initially.
- The loop invariant must be preserved by the loop body, under the assumption that the loop guard is true.
- The postcondition of the loop must be implied by the loop invariant together with the negated loop guard.

If we can prove all three of these then we can conclude the postcondition of the loop. In this example, we pick the loop invariant J to be $x \geq 0$. Then we have to prove:

True Initially $x \geq 6 \vdash x \geq 0$

Preserved $x \geq 0, x > 1 \vdash [x := x - 2]\ x \geq 0$

Implies Postcondition $x \geq 0, \neg(x > 1) \vdash 0 \leq x \leq 1$

These are all easy to prove (with an oracle for arithmetic), reducing the one in the middle in one step (using rule $[:=]R^{x'}$) to

$$x \geq 0, x > 1, x' = x - 2 \vdash x' \geq 0$$

Summarizing all of this with a rule yields the following, for an arbitrary loop invariant J.

$$\frac{\Gamma \vdash J, \Delta \quad J, P \vdash [\alpha]J \quad J, \neg P \vdash Q}{\Gamma \vdash [\mathbf{while}\ P\ \alpha]Q, \Delta}\ [\mathbf{while}]R$$

Sadly, since J is an arbitrary formula, this rule is not reductive. It would be okay, however, if we forced the programmer to write J in the program (as we do in C0), because then each premise only refers to components of the conclusion. When we want to emphasize this point we may write

$$[\mathbf{while}_J\ P\ \alpha]Q$$

where J is the loop invariant.

An important point about this rule is that we drop Γ and Δ in the second and third premise. This is because we don't know how often we may have to go around the loop. Preservation (the second premise) has to hold for any state we might reach during the iteration, but the antecedents in Γ are only guaranteed *before* the first iteration.

In our example, $x \geq 6$ is only known to be true before the loop starts, and not after each iteration. In fact, it may be false after the first iteration and therefore we cannot use it to prove for the second and third premise. Similarly, the additional succedents Δ also must be dropped. Otherwise we could reformulate the goal

$$\cdot \vdash [\mathbf{while}\ (x > 1)\ x := x - 2]\, 0 \leq x \leq 1, \neg(x \geq 6)$$

and then use $\neg R$ rule to turn the succedent $\neg(x \geq 6)$ into the (unwarranted) antecedent $x \geq 6$.

Putting all of this together, we can prove our program as follows.

$$\dfrac{\dfrac{\text{(by arithmetic)}}{x \geq 6 \vdash x \geq 0} \quad \dfrac{\dfrac{\text{(by arithmetic)}}{x \geq 0, x > 1, x' = x - 2 \vdash x' \geq 0}}{x \geq 0, x > 1 \vdash [x := x - 2]\, x \geq 0}\,[:=]R^{x'} \quad \dfrac{\text{(by arithmetic)}}{x \geq 0, \neg(x > 1) \vdash 0 \leq x \leq 1}}{\dfrac{x \geq 6 \vdash [\mathbf{while}\ (x > 1)\ x := x - 2]\, 0 \leq x \leq 1}{\cdot \vdash x \geq 6 \to [\mathbf{while}\ (x > 1)\ x := x - 2]\, 0 \leq x \leq 1}\,{\to}R}\,[\mathbf{while}]R^{*}$$

$(*)$ with loop invariant $J(x) = (x \geq 0)$

You can find a summary of the relevant rules we have intuited in Figure 1. The $[\mathbf{unfold}]R$ rule has a special status because it is not reductive (but will still turn out to be sound and useful). The $[\mathbf{while}]R$ rules is reductive only if the loop invariant J is specified in the syntax, as indicated by the subscript.

$$\frac{\Gamma, P \vdash [\alpha]Q, \Delta \quad \Gamma, \neg P \vdash [\beta]Q, \Delta}{\Gamma \vdash [\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]Q, \Delta}\,[\textbf{if}]R \qquad \frac{\Gamma, x' = e \vdash Q(x'), \Delta}{\Gamma \vdash [x := e]Q(x), \Delta}\,[:=]R^{x'}$$

$$\frac{\Gamma \vdash [\alpha]([\beta]Q), \Delta}{\Gamma \vdash [\alpha\ ;\ \beta]Q, \Delta}\,[;]R \qquad \frac{\Gamma \vdash J, \Delta \quad J, P \vdash [\alpha]J \quad J, \neg P \vdash Q}{\Gamma \vdash [\textbf{while}_J\ P\ \alpha]Q, \Delta}\,[\textbf{while}]R$$

$$\frac{\Gamma, P \vdash [\alpha]([\textbf{while}\ P\ \alpha])Q, \Delta \quad \Gamma, \neg P \vdash Q, \Delta}{\Gamma \vdash [\textbf{while}\ P\ \alpha]Q, \Delta}\,[\textbf{unfold}]R$$

Figure 1: Some Rules for Dynamic Logic

3 Semantics of Expressions

Recall from Lecture 3

Variables	x, y, z		
Constants	c	$::=$	$\dots, -1, 0, 1, \dots$
Expressions	e	$::=$	$c \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid \dots$
Programs	α, β	$::=$	$x := e \mid \alpha\ ;\ \beta \mid \textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta \mid \textbf{while } P\ \alpha$
Formulas	P, Q	$::=$	$e_1 \leq e_2 \mid e_1 = e_2 \mid \dots$
		$\mid$	$P \wedge Q \mid P \vee Q \mid P \to Q \mid P \leftrightarrow Q \mid \neg P \mid \top \mid \bot$
		$\mid$	$[\alpha]Q \mid \langle\alpha\rangle Q$

In order to give semantics to formulas we also need to give semantics to programs and expressions. We start with expressions. We assume (for now) that the value of an expression is always an integer. We also need to recall that *states* map all variables to integers and write $\omega(x)$ for the value of variable x in state ω.

Generally, we write $[\![$something$]\!]$ for the semantic meaning of "something", so this notation will be overloaded. For expressions, it depends on a state and returns an integer. We write this as $\omega[\![e]\!] = c$.

$$\begin{array}{lcl} \omega[\![c]\!] & = & c \\ \omega[\![x]\!] & = & \omega(x) \\ \omega[\![e_1 + e_2]\!] & = & \omega[\![e_1]\!] + \omega[\![e_2]\!] \\ \omega[\![e_1 * e_2]\!] & = & \omega[\![e_1]\!] \times \omega[\![e_2]\!] \end{array}$$

There may be more cases if we consider additional operators. The operators inside the semantic brackets are syntax, the operators outside the semantic brackets are operations on integers. For example, in a state ω where $\omega(x) = 4$ we calculate

$$\omega[\![x + x]\!] = \omega[\![x]\!] + \omega[\![x]\!] = 4 + 4 = 8$$

4 Semantics of Programs

We interpret a program as denoting a relation between a prestates and poststates. Because of loops, for a given prestate there may be zero or one poststates. We again use semantic brackets, writing $\omega[\![\alpha]\!]\nu$ if the program α relates the prestate ω to the poststate ν. We write the program in the middle because, generally in mathematics, we like to write relations in infix form (like $e_1 \leq e_2$).

The definition follows, mostly, the way a program would execute and we only really have to think much when we come to loops.

Assignment. An assignment $x := e$ will evaluate e and then *update* the state so it now maps x to the value of e. We define the notation $\omega[x \mapsto c]$ with

$$\begin{array}{rcll} (\omega[x \mapsto c])(x) & = & c & \\ (\omega[x \mapsto c])(y) & = & \omega(y) & \text{provided } x \neq y \end{array}$$

Note that the notation $\omega[x \mapsto c]$ has nothing to do with the formula $[\alpha]Q$, square brackets are overloaded as well.

Then we define

$$\omega[\![x := e]\!]\nu \quad \text{iff} \quad \omega[x \mapsto c] = \nu \text{ where } \omega[\![e]\!] = c$$

Sequential Composition. We execute $\alpha\ ;\ \beta$ by first executing α and then β in the poststate of α.

$$\omega[\![\alpha\ ;\ \beta]\!]\nu \quad \text{iff} \quad \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\beta]\!]\nu \text{ for some state } \mu$$

This definition if implies that if α has no poststate (that is, doesn't terminate) then $\alpha\ ;\ \beta$ doesn't either, which is intuitively correct.

Conditionals. For conditionals we need to appeal to the meaning of formulas, because we need to know if the condition is true or false.

$$\begin{array}{lll} \omega[\![\textbf{if } P \textbf{ then } \alpha \textbf{ else } \beta]\!]\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\nu \quad \text{or} \\ & & \omega \not\models P \text{ and } \omega[\![\beta]\!]\nu \end{array}$$

To be consistent, the meaning of a proposition should really be a truth value denoted by $\omega[\![P]\!]$, but there is a long tradition of writing $\omega \models P$ which means that P is true in state ω. Such a state is often called a *model* in which P is true.

Loops. As you might expect, loops are the trickiest. Here is a possible *recursive* definition.

$$\begin{array}{lll} \omega[\![\textbf{while } P\ \alpha]\!]\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\textbf{while } P\ \alpha]\!]\nu \quad \text{or} \\ & & \omega \not\models P \text{ and } \omega = \nu \end{array}$$

This definition is recursive in the sense that the program also appears on the right. It could then be ambiguous whether and for which states $\omega[\![\mathbf{while}\ \top\ \mathbf{skip}]\!]\nu$ should be true.

We can use a more explicit *inductive definition* using an auxiliary relation $[\![\mathbf{while}\ P\ \alpha]\!]^n$ that prescribes the number of iterations to be n.

$$\begin{array}{lll} \omega[\![\mathbf{while}\ P\ \alpha]\!]\nu & \text{iff} & \omega[\![\mathbf{while}\ P\ \alpha]\!]^n\nu \quad \text{for some } n \in \mathbb{N} \\ \\ \omega[\![\mathbf{while}\ P\ \alpha]\!]^{n+1}\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\mathbf{while}\ P\ \alpha]\!]^n\nu \\ \omega[\![\mathbf{while}\ P\ \alpha]\!]^0\nu & \text{iff} & \omega \not\models P \text{ and } \omega = \nu \end{array}$$

If there is no such n, then there is no poststate for the given prestate.

5 Semantics of Formulas

The semantics of formulas in a given state must appeal to the meaning of expressions and the meaning of programs. Therefore the meanings of programs and formulas mutually depend on each other. We start with some simple cases.

$$\begin{array}{lllll} \omega \models e_1 \leq e_2 & \text{iff} & \omega[\![e_1]\!] \leq \omega[\![e_2]\!] \\ \omega \models e_1 = e_2 & \text{iff} & \omega[\![e_1]\!] = \omega[\![e_2]\!] \\ \\ \omega \models P \wedge Q & \text{iff} & \omega \models P & \text{and} & \omega \models Q \\ \omega \models P \vee Q & \text{iff} & \omega \models P & \text{or} & \omega \models Q \\ \omega \models P \rightarrow Q & \text{iff} & \omega \models P & \text{implies} & \omega \models Q \\ \omega \models \neg P & \text{iff} & \omega \not\models P \\ \omega \models P \leftrightarrow Q & \text{iff} & \omega \models P & \text{iff} & \omega \models Q \end{array}$$

For programs, we have to recall the informal definition from the previous lecture. $[\alpha]Q$ is true if Q is true in *every* poststate of α. Because a nonterminating program does not have a poststate, this is a statement about the *partial correctness* of the program α. Conversely, $\langle\alpha\rangle Q$ is true if Q is true in *some* poststate of α.

$$\begin{array}{lll} \omega \models [\alpha]Q & \text{iff} & \text{for every } \nu \text{ with } \omega[\![\alpha]\!]\nu \text{ we have } \nu \models Q \\ \omega \models \langle\alpha\rangle Q & \text{iff} & \text{there is a } \nu \text{ with } \omega[\![\alpha]\!]\nu \text{ and } \nu \models Q \end{array}$$

Now we say P is *valid* (written as $\models P$) if $\omega \models P$ for every state ω. (Recall that all states are defined on all variables, so this is well-defined.)

A sequent $P_1, \ldots, P_n \vdash Q_1, \ldots, Q_m$ is valid if for every state ω, whenever for all antecedents P_i we have $\omega \models P_i$ then for some succedent Q_j we have $\omega \models Q_j$.

6 Quantification and Substitution[1]

[1]Not covered in lecture

So far, we haven't introduced or used quantifiers, except the implicit quantification over all states in the definition of validity. It is very tempting to define that $\omega \models \forall x.\, P(x)$ if $\omega \models P(c)$ for every $c \in \mathbb{Z}$. Here, $P(c)$ is the notation of substituting c for every occurrence of x in $P(x)$.

This makes sense if $P(x)$ is a formula of pure arithmetic. But if $P(x)$ makes reference to programs this doesn't quite work. For example, given our earlier proof we might expect that

$$\forall x.\, x \geq 6 \rightarrow [\textbf{while}\ (x > 1)\ x := x - 2]\, 0 \leq x \leq 1$$

is true in every state. But we cannot substitute an integer for x for the same reason we needed to drop the antecedents Γ (and the succedents Δ) in the rule $[\textbf{while}]R$: the guard of the loop implicitly refers to the x in many states (every state reachable by executing loop) and not just the initial state.

We therefore define instead:

$$\begin{array}{llll} \omega \models \forall x.\, P(x) & \text{iff} & \omega[x \mapsto c] \models P(x) & \text{for every } c \in \mathbb{Z} \\ \omega \models \exists x.\, P(x) & \text{iff} & \omega[x \mapsto c] \models P(x) & \text{for some } c \in \mathbb{Z} \end{array}$$

The proof rules then become a bit strange, but fortunately we will often be in the quantifier-free fragment, or can delegate quantifier reasoning to the arithmetic oracle. In both of these rules, the x' has to be chosen fresh (that is, it doesn't appear in the conclusion or in e). Also, for technical reasons we need to keep a copy of the existential in the $\exists R$ rule, just as a universally quantified antecedent may be needed more than once.

$$\frac{\Gamma \vdash P(x'), \Delta}{\Gamma \vdash \forall x.\, P(x), \Delta}\ \forall R^{x'} \qquad \frac{\Gamma, \forall x.\, P(x), x' = e, P(x') \vdash \Delta}{\Gamma, \forall x.\, P(x) \vdash \Delta}\ \forall L^{x'}$$

$$\frac{\Gamma, x' = e \vdash P(x'), \exists x.\, P(x), \Delta}{\Gamma \vdash \exists x.\, P(x), \Delta}\ \exists R^{x'} \qquad \frac{\Gamma, P(x') \vdash \Delta}{\Gamma, \exists x.\, P(x) \vdash \Delta}\ \exists L^{x'}$$

Reading the rules $\forall L^{x'}$ and $\exists R^{x'}$ bottom-up, we can freely choose the expression e to instantiate the quantifier with as long as it doesn't mention x'.

7 Rules versus Axioms

In general, it seems more convient to reason with rules since the rules of the sequent calculus give clear view of the state of an incomplete proof. In some situations, though, we can pack a lot of information into *axioms*, by which we mean valid formulas.

Recall the right rule for sequential composition.

$$\frac{\Gamma \vdash [\alpha]([\beta]Q), \Delta}{\Gamma \vdash [\alpha\ ;\ \beta]Q, \Delta}\ [;]R$$

We reduce the goal of proving $[\alpha\ ;\ \beta]Q$ to the goal of proving $[\alpha]([\beta]Q)$. What would be a corresponding *left* rule $[;]L$? What can we deduce from knowing $[\alpha\ ;\ \beta]Q$? It looks as if we should just be able to replace this with the antecedent $[\alpha]([\beta]Q)$ because $[\alpha\ ;\ \beta]Q$ and $[\alpha]([\beta]Q)$ are equivalent.

$$\frac{\Gamma, [\alpha]([\beta]Q) \vdash \Delta}{\Gamma, [\alpha\ ;\ \beta]Q \vdash \Delta}\ [;]L$$

Here is a general observation: if P and Q are equivalent (in the sense that $P \leftrightarrow Q$ is valid) then rules such as

$$\frac{\Gamma \vdash Q, \Delta}{\Gamma \vdash P, \Delta} \qquad \frac{\Gamma, Q \vdash \Delta}{\Gamma, P \vdash \Delta}$$

are **both sound and invertible**. That's because P and Q are always either both false or both true, regardless of the state because the bi-implication $P \leftrightarrow Q$ is valid.

In order to obtain good left and right rules from valid equivalences we just have to make sure the rules are reductive. As an example, the $[;]R$ and $[;]L$ rules can both be constructed from the following equivalence.

$$\models [\alpha\ ;\ \beta]Q \leftrightarrow [\alpha]([\beta]Q)$$

Let's prove this. We start with the right-to-left direction, which implies the *soundness* of $[;]R$ and *invertibility* of $[;]L$. We set up the proof:

$\omega \vdash [\alpha]([\beta]Q)$ (assumption)
...
$\omega \vdash [\alpha\ ;\ \beta]Q$ (to show)

We now walk through the proof in individual steps, narrowing the gap, sometimes from below and sometimes from above. Typically, one only presents the end result and the reader has to figure out how one might have obtained it.

By definition, the conclusion holds if for every state ν such that $\omega[\![\alpha\ ;\ \beta]\!]\nu$ we have $\nu \models Q$. Now our proof state is (highlighting the new parts in blue):

$\omega \models [\alpha]([\beta]Q)$ (1, assumption)
$\omega[\![\alpha\ ;\ \beta]\!]\nu$ for some ν (2, assumption)
...
$\nu \models Q$ (to show)
$\omega \models [\alpha\ ;\ \beta]Q$ (by defn. of $\models$)

By definition, assumption 2 is true if there is some intermediate state μ such that $\omega[\![\alpha]\!]\mu$ and $\mu[\![\beta]\!]\nu$. Let's write this into the proof as well.

$\omega \models [\alpha]([\beta]Q)$	(1, assumption)
$\omega\llbracket\alpha \,;\, \beta\rrbracket\nu$ for some ν	(2, assumption)
$\omega\llbracket\alpha\rrbracket\mu$ and $\mu\llbracket\beta\rrbracket\nu$ for some μ	(3, from 2 by defn. of $\llbracket-\rrbracket$)
...	
$\nu \models Q$	(to show)
$\omega \models [\alpha \,;\, \beta]Q$	(by defn. of $\models$)

Next: from assumption 1 and the fact that $\omega\llbracket\alpha\rrbracket\mu$ we can conclude that $\mu \models [\beta]Q$, again just by the definition of $\models$.

$\omega \models [\alpha]([\beta]Q)$	(1, assumption)
$\omega\llbracket\alpha \,;\, \beta\rrbracket\nu$ for some ν	(2, assumption)
$\omega\llbracket\alpha\rrbracket\mu$ and $\mu\llbracket\beta\rrbracket\nu$ for some μ	(3, from 2 by defn. of $\llbracket-\rrbracket$)
$\mu \models [\beta]Q$	(4, from 1 and 3(a) by defn. of $\models$)
...	
$\nu \models Q$	(to show)
$\omega \models [\alpha \,;\, \beta]Q$	(by defn. of $\models$)

Now we use the same argument knowing that $\mu\llbracket\beta\rrbracket\nu$ and $\mu \models [\beta]Q$ to conclude that $\nu \models Q$. But that's what we needed to show!

$\omega \models [\alpha]([\beta]Q)$	(1, assumption)
$\omega\llbracket\alpha \,;\, \beta\rrbracket\nu$ for some ν	(2, assumption)
$\omega\llbracket\alpha\rrbracket\mu$ and $\mu\llbracket\beta\rrbracket\nu$ for some μ	(3, by defn. of $\llbracket-\rrbracket$ from 2)
$\mu \models [\beta]Q$	(4, from 1 and 3(a) by defn. of $\models$)
$\nu \models Q$	(5, from 4 and 3(b) by defn. of $\models$)
$\omega \models [\alpha \,;\, \beta]Q$	(from 5 and 2 by defn. of $\models$)

We see the proof is actually quite straightforward. We just have to carefully unwind the definitions.

Here is the proof in the other direction.[2] We set up:

$\omega \models [\alpha \,;\, \beta]Q$	(1, assumption)
...	
$\omega \models [\alpha]([\beta]Q)$	(to show)

We show the filled-in proof. You can probably walk through it in the order we made the deductions.

$\omega \models [\alpha \,;\, \beta]Q$	(1, assumption)
$\omega\llbracket\alpha\rrbracket\mu$ for some μ	(2, assumption)
$\mu\llbracket\beta\rrbracket\nu$ for some ν	(3, assumption)
$\omega\llbracket\alpha \,;\, \beta\rrbracket\nu$	(4, from 2 and 3 by defn. of $\llbracket-\rrbracket$
$\nu \models Q$	(5, from 1 and 4 by defn of $\models$)
$\mu \models [\beta]Q$	(6, from 5 and 3 by defn of $\models$)
$\omega \models [\alpha]([\beta]Q)$	(7, from 6 and 2 by defn of $\models$)

[2]not shown in lecture

At this point we have shown that the right and left rules for the sequential composition of programs in a box modality are sound and invertible.

8 Some Axioms for Dynamic Logic[3]

Based on the insights from the previous section and the rules for programs in dynamic logic, we can conjecture the following axioms. And, indeed, they are all valid, even though we don't show the proofs. We write the postfix "A" to indicate that what we are naming is not a rule but an axiom.

$[{:=}]A$	$[x := e]Q(x) \leftrightarrow \forall x'.x' = e \rightarrow Q(x')$ (x' not in e or $Q(x)$)
$[;]A$	$[\alpha \,;\, \beta]Q \leftrightarrow [\alpha]([\beta]Q)$
$[\mathbf{if}]A$	$[\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]Q \leftrightarrow (P \rightarrow [\alpha]Q) \wedge (\neg P \rightarrow [\beta]Q)$
$[\mathbf{unfold}]A$	$[\mathbf{while}\ P\ \alpha] \leftrightarrow (P \rightarrow [\alpha][\mathbf{while}\ P\ \alpha]Q) \wedge (\neg P \rightarrow Q)$

Because these axioms are valid biconditionals, we obtain correct left and right rules for the sequent calculus, with the rules for $[\mathbf{unfold}]$ being somewhat unsatisfactory since they are not reductive.

Unfortunately, the rule $[\mathbf{while}]R$ including loop invariants can't be easily turned into an axiom. Recall:

$$\frac{\Gamma \vdash J, \Delta \quad J, P \vdash [\alpha]J \quad J, \neg P \vdash Q}{\Gamma \vdash [\mathbf{while}_J\ P\ \alpha]Q, \Delta}\ [\mathbf{while}]R$$

We won't propose a corresponding left rule for this because it takes us into the realm of total correctness and proving termination, which is beyond the scope of this course.

We won't pursue it further, but for the curious, it is possible to obtain an axiom for the direction that corresponds to soundness of $[\mathbf{while}]R$, but we need an additional logical operator $\Box P$.

$$[\mathbf{while}]A \quad [\mathbf{while}\ P\ \alpha]Q \leftarrow J \wedge \Box(J \wedge P \rightarrow [\alpha]J) \wedge \Box(J \wedge \neg P \rightarrow Q)$$

The new component here is the modality $\Box P$ which is defined semantically by $\omega \vdash \Box P$ iff $\nu \vdash P$ for every state ν. Needing to prove this is what removes Γ and Δ in the bottom-up reading of $[\mathbf{while}]R$.

As an aside, the problem with substitution and the $\Box$ modality from modal logic is nothing new but has concerned philosophers long before computer science and dynamic logic. See, for example, Garson [2000, Section 16].

9 Summary

We summarize the semantic definitions and axioms; the sequent calculus rules can be found in Figure 1.

[3]Only $[;]A$ covered in lecture; the remainder may be useful for reference or further reading.

$$\begin{array}{lcl}
\omega[\![c]\!] & = & c \\
\omega[\![x]\!] & = & \omega(x) \\
\omega[\![e_1 + e_2]\!] & = & \omega[\![e_1]\!] + \omega[\![e_2]\!] \\
\omega[\![e_1 * e_2]\!] & = & \omega[\![e_1]\!] \times \omega[\![e_2]\!]
\end{array}$$

Figure 2: Semantics of Expressions

$$\begin{array}{lll}
\omega[\![x := e]\!]\nu & \text{iff} & \omega[x \mapsto c] = \nu \text{ where } \omega[\![e]\!] = c \\
\omega[\![\alpha \,;\, \beta]\!]\nu & \text{iff} & \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\beta]\!]\nu \text{ for some state } \mu \\
\omega[\![\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]\!]\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\nu \quad \text{or} \\
 & & \omega \not\models P \text{ and } \omega[\![\beta]\!]\nu \\
\omega[\![\mathbf{while}\ P\ \alpha]\!]\nu & \text{iff} & \omega[\![\mathbf{while}\ P\ \alpha]\!]^n\nu \quad \text{for some } n \in \mathbb{N} \\
\\
\omega[\![\mathbf{while}\ P\ \alpha]\!]^{n+1}\nu & \text{iff} & \omega \models P \text{ and } \omega[\![\alpha]\!]\mu \text{ and } \mu[\![\mathbf{while}\ P\ \alpha]\!]^n\nu \\
\omega[\![\mathbf{while}\ P\ \alpha]\!]^0\nu & \text{iff} & \omega \not\models P \text{ and } \omega = \nu
\end{array}$$

Figure 3: Semantics of Programs

$$\begin{array}{lll}
\omega \models e_1 \leq e_2 & \text{iff} & \omega[\![e_1]\!] \leq \omega[\![e_2]\!] \\
\omega \models e_1 = e_2 & \text{iff} & \omega[\![e_1]\!] = \omega[\![e_2]\!] \\
\\
\omega \models P \wedge Q & \text{iff} & \omega \models P \quad \text{and} \quad \omega \models Q \\
\omega \models P \vee Q & \text{iff} & \omega \models P \quad \text{or} \quad \omega \models Q \\
\omega \models P \rightarrow Q & \text{iff} & \omega \models P \quad \text{implies} \quad \omega \models Q \\
\omega \models \neg P & \text{iff} & \omega \not\models P \\
\omega \models P \leftrightarrow Q & \text{iff} & \omega \models P \quad \text{iff} \quad \omega \models Q \\
\\
\omega \models [\alpha]Q & \text{iff} & \text{for every } \nu \text{ with } \omega[\![\alpha]\!]\nu \text{ we have } \nu \models Q \\
\omega \models \langle\alpha\rangle Q & \text{iff} & \text{there is a } \nu \text{ with } \omega[\![\alpha]\!]\nu \text{ and } \nu \models Q
\end{array}$$

Figure 4: Semantics of Formulas

References

James Garson. Modal logic. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Spring 2024 edition edition, 2000. URL `https://plato.stanford.edu/archives/spr2024/entries/logic-modal/`.