15-414: Bug Catching: Automated Program Verification

Lecture Notes on Predicate Transformers

Matt Fredrikson

Carnegie Mellon University
Lecture 12
Tuesday, February 20, 2025

1 Introduction

We have now studied dynamic logic in some depth, providing a semantics for programs and also a set of valid axioms we can use to reason about the programs. These axioms concerned propositions of the form $[\alpha]Q$ and were organized so as to break down the structure of the *program* α. The goal was for them to be complete enough so we can break down questions about a program's correctness into a purely logical question. Such a purely logical statement for the correctness of a program is called a *verification condition*.

In this lecture we complete this line of investigation by providing *algorithms* for calculating verification conditions. We discover that there are two principal algorithms for such a calculation. In one algorithm we are given a program α and a postcondition Q we calculate the *weakest precondition* P such that $P \to [\alpha]Q$. Here, P is *weakest* in the sense that for another other correct precondition P' we have $P' \to P$, that is, P is both necessary and sufficient to ensure the postcondition. Using the weakest precondition is the dominant way that verifiers such as Why3 work.

Conversely, given a precondition P and program α we can calculate the *strongest postcondition* Q such that $P \to [\alpha]Q$. It is *strongest* in the sense that for any other correct postcondition Q' we have $Q \to Q'$, that is, Q is a necessary and sufficient postcondition. The strongest postcondition is closely related to *symbolic execution* which is used in other tools, for example, in *model checking*.

Our presentation of weakest precondition and strongest postcondition is largely based on the paper by Gordon and Collavizza [GC10] which has additional references and some historical notes.

We ignore questions of variants and invariants for loops (or repetition), which must in practice be present to compute the weakest precondition and strongest postconditions. It is not difficult to extend the algorithms using what we learned about them in dynamic logic.

Remarkably, we can almost mechanically "read off" rules for calculating weakest preconditions from the axioms for dynamic logic. This also provides a path towards proving the algorithm correct.

On the other hand, using our intuition about program execution makes it mostly straightforward to construct a strongest postcondition, but the relationship to dynamic logic is not obvious. Streett [Str82] studied this, introducing a converse program operator α^- in the context of propositional dynamic logic (without quantifiers or variable assignment). The meaning of α^- is given by $\omega[\![\alpha^-]\!]\nu$ iff $\nu[\![\alpha]\!]\omega$. Streett observed that the strongest postcondition of P is $\langle\alpha^-\rangle P$, but did not develop this observation further. We come at this more directly, and show that there is a new modality $\langle\!\langle\alpha\rangle\!\rangle P$ that expresses the strongest postcondition for P. We define it semantically and "read off" its axioms from the algorithm for calculating it. We conjecture that $\langle\!\langle\alpha\rangle\!\rangle P$ is a new modality, not already definable in dynamic logic beyond Streett's work.

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

- Calculate the weakest precondition for a program given a postcondition
- Calculate the strongest postcondition for a program given a precondition
- Relate calculational rules for pre/postconditions to modalities in dynamic logic
- Prove the correctness of algorithms for calculating verification conditions

2 Hoare Triples

It is often helpful to think of verification in terms of *Hoare Triples* $P\{\alpha\}Q$ which are true if we start executing α in a state satisfying P then any final state will satisfy Q. Hoare's original language [Hoa69] was deterministic, so the final state (if one existed) was also unique. It is important that P and Q here are purely logical formulas that don't refer to a program, although they do contain variables and expressions (in our case, arithmetic expressions). We can easily translate this into dynamic logic as $P \to [\alpha]Q$ which has the same meaning.

Hoare then defined inference rules directly operating on triples, such as

$$\frac{P\{\alpha\}R \quad R\{\beta\}Q}{P\{\alpha\,;\,\beta\}Q}$$

which are valid in the sense that if the two *premises* are valid, then so is the *conclusion*. We could verify this, for example, in dynamic logic by checking that if $\models P \to [\alpha]R$ and

$\models R \to [\beta]Q$ then $\models P \to [\alpha\ ;\ \beta]Q$. This follows by a short chain of reasoning using the semantic definition of validity for dynamic logic (see Exercise 1).

The definition of the *weakest precondition* $\mathsf{wp}(\alpha)Q$ has two parts:

1. $(\mathsf{wp}(\alpha)Q)\{\alpha\}Q$ (it is a precondition, or: it is *sufficient* for Q)
2. If $P\{\alpha\}Q$ then $P \to \mathsf{wp}(\alpha)Q$ (it is a *weakest* precondition, or: it is *necessary* for Q).

Note that the weakest precondition will be unique up to logical equivalence. Note that $\bot$ (falsehood) is a precondition for any α and Q, so the second condition is needed to make the definition interesting.

Let's also remember that the weakest precondition just defined, and also the strongest postcondition to come next, are formulas in pure logic and do not reference programs.

The *strongest postcondition* $\mathsf{sp}(\alpha)P$ also has two parts:

1. $P\{\alpha\}(\mathsf{sp}(\alpha)P)$ (it is a postcondition, or: it is a necessary consequence of P)
2. If $P\{\alpha\}R$ then $\mathsf{sp}(\alpha)P \to R$ (it is a *strongest* postcondition for P, or: is it sufficient for all consequences of P)

Here we can observe that $\top$ (truth) is always a postcondition for any α and P, so the second condition is needed to make the definition interesting.

3 Weakest Precondition

Translated into the our semantic framework, the weakest precondition can be specified as follows:

(i) $\mathsf{wp}(\alpha)Q$ is a precondition for Q (it is *sufficient* for Q):
If $\omega \models \mathsf{wp}(\alpha)Q$ and $\omega[\![\alpha]\!]\nu$ then $\nu \models Q$

(ii) $\mathsf{wp}(\alpha)Q$ is the weakest precondition for Q (it is *necessary* for Q):
Whenever $\omega[\![\alpha]\!]\nu$ implies $\nu \models Q$ for all ν, then $\omega \models \mathsf{wp}(\alpha)Q$.

These two together are precisely the semantic definition of $[\alpha]Q$, namely

$\omega \models [\alpha]Q$ iff for all ν with $\omega[\![\alpha]\!]\nu$ we have $\nu \models Q$

The second clause implies that it is the weakest precondition:

Assume that P is a precondition for Q, that is,
for all μ and ν, if $\mu \models P$ and $\mu[\![\alpha]\!]\nu$ then $\nu \models Q$ (1)
To show: for all ω, $\omega \models P \to \mathsf{wp}(\alpha)Q$
So assume $\omega \models P$ for an arbitrary ω (2)
To show: $\omega \models \mathsf{wp}(\alpha)Q$
We have that for all ν, $\omega[\![\alpha]\!]\nu$ implies $\nu \models Q$ by (1) for $\mu = \omega$ and (2)
Now $\omega \models \mathsf{wp}(\alpha)Q$ follows by condition (ii).

The upshot is that $[\alpha]Q$ is logically equivalent to the weakest precondition $\mathsf{wp}(\alpha)Q$, the difference being that the latter is purely logical, while $[\alpha]Q$ contains a reference to α. This means we can now derive rules for the computation of $\mathsf{wp}(\alpha)Q$ from the axioms for $[\alpha]Q$ that decompose α.

We now examine each program construct in term, exploiting

$$\models \mathsf{wp}(\alpha)Q \leftrightarrow [\alpha]Q$$

Sequential composition. Recall the axiom for sequential composition:

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

This give us the equation:

$$\mathsf{wp}(\alpha\,;\,\beta)Q = \mathsf{wp}(\alpha)(\mathsf{wp}(\beta)Q)$$

Nondeterministic choice.

$$\models [\alpha \cup \beta]Q \leftrightarrow [\alpha]Q \wedge [\beta]Q$$

This yields

$$\mathsf{wp}(\alpha \cup \beta)Q = \mathsf{wp}(\alpha)Q \wedge \mathsf{wp}(\beta)Q$$

Guard.

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

$$\mathsf{wp}(?P)Q = (P \to Q)$$

Iteration.

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

$$\mathsf{wp}(\alpha^*)Q = Q \wedge \mathsf{wp}(\alpha)(\mathsf{wp}(\alpha^*)Q)$$

As a straightforward recursive definition, this may not terminate, so in practice we use alternative of the induction axiom with invariants. This is explored in Exercise 2.

Assignment. In some ways, assignment is the most interesting clause in the definition. Recall our axiom:

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

Here, we rename $Q(x)$ by changing all occurrence of x to x'. The soundness requires that x' does not occur in e or $Q(x)$, which we sometimes summarize by saying that x' is fresh. This axiom was partly inspired by our translation from imperative to functional programs where we bind fresh variables instead of assigning to existing ones.

One might think it is possible to perform instead a *substitution*. If we write $Q(e)$ for the result of substituting e for x in $Q(x)$ (which is the same as e for x' in $Q(x')$), then this implicitly relies on

$$\models (\forall x'.\, x' = e \to Q(x')) \leftrightarrow Q(e)$$

At first, one might think this is correct if we are just being careful about substitution. This *capture-avoiding substitution*, written $(e/x)P$, ensures that no variable in e is "captured" by a binding construct or imperative update in P and may require the renaming of some internal variables. For example:

$$\begin{array}{lcll}(x+1/x)([x \leftarrow x+2]Q(x)) & = & [x \leftarrow (x+1)+2]Q(x) & \\ (y+1/x)([y \leftarrow 3]Q(x,y)) & = & [y' \leftarrow 3](Q(y+1,y')) & y' \notin Q(x,y)\end{array}$$

Unfortunately, that's not sufficient. For example,

$$(3/x)([(?(x>0)\ ;\ y \leftarrow y+1\ ;\ x \leftarrow x-1)^*]Q(x,y))$$

We cannot replace any of the occurrences of x by 3 and retain the meaning of the program. This would come up when reasoning about this program:

$$[x \leftarrow 3\ ;\ (?(x>0)\ ;\ y \leftarrow y+1\ ;\ x \leftarrow x-1)^*]Q(x,y)$$

In other words, *we could not use an axiom for assignment that carries out substitution this case*. With the axiom we currently have, this would correctly become

$$\forall x'.\, x' = 3 \rightarrow [(?(x'>0)\ ;\ y \leftarrow y+1\ ;\ x' \leftarrow x'-1)^*]Q(x',y)$$

When we then reason by about the loop using the induction axiom, the $\Box$ modality will make the assumption $x' = 3$ unavailable, for example, when reasoning about the postcondition (where we would expect it to be $x' = 0$ instead).

All these considerations then yield

$$\mathsf{wp}(x \leftarrow e)Q(x) = \forall x'.x' = e \rightarrow Q(x') \quad (x' \notin e, Q(x'))$$

However, in this case $Q(x)$ is a purely logical formula without programs, we can rewrite it as

$$\mathsf{wp}(x \leftarrow e)(Q(x)) = (e/x)(Q(x))$$

where the latter is capture-avoiding substitution. This is now always defined, because we can rename the quantified variables in $Q(x)$ as needed when it has no dynamic modalities.

4 Strongest Postconditions

Let's recall the definition of *strongest postcondition* $\mathsf{sp}(\alpha)P$ in two parts:

1. $P\{\alpha\}(\mathsf{sp}(\alpha)P)$ (it is a postcondition, or: it is a necessary consequence of P)
2. If $P\{\alpha\}R$ then $\mathsf{sp}(\alpha)P \rightarrow R$ (it is a *strongest* postcondition for P, or: is it sufficient for all consequences of P)

Since this concerns "executing the program" and seeing how much we might know afterwards, let's go in the opposite direction and postulate a definition based on the definition

(i) $\mathsf{sp}(\alpha)P$ is a postcondition for P (it is *necessarily* true after executing α in any state satisfying P):

For all ν and ω, if $\omega \models P$ and $\omega[\![\alpha]\!]\nu$ then $\nu \models \mathsf{sp}(\alpha)P$

(ii) $\mathsf{sp}(\alpha)P$ is sufficient for all postconditions of P (it implies all other postconditions):

Whenever $\nu \models \mathsf{sp}(\alpha)P$ then there is an ω such that $\omega \models P$ and $\omega[\![\alpha]\!]\nu$.

We should check that the second clause implies that it is indeed a strongest postcondition.

Assume that Q is a postcondition for P, that is,
for all μ and ν, if $\mu \models P$ and $\mu[\![\alpha]\!]\nu$ then $\nu \models Q$ (1)
To show: for all ν', if $\nu' \models \mathsf{sp}(\alpha)P$ then $\nu' \models Q$
So assume $\nu' \models \mathsf{sp}(\alpha)P$ (2)
To show: $\nu' \models Q$
There is an ω such that $\omega \models P$ and $\omega[\![\alpha]\!]\nu'$ by (ii) for $\nu = \nu'$ and (2)
Then $\nu' \models Q$ by (1) for $\mu = \omega$ and $\nu = \nu'$

We can summarize (i) and (ii) by

$\nu \models \mathsf{sp}(\alpha)P$ iff there exists an ω such that $\omega \models P$ and $\omega[\![\alpha]\!]\nu$

We obtain the right-to-left implication of this definition by rewriting (i), exploiting that ω does not appear in $\nu \models \mathsf{sp}(\alpha)P$ and the logical law $(\forall x.\, P(x) \to Q) \leftrightarrow (\exists x.\, P(x)) \to Q$

Sequential composition. This time, without the benefit of natural axioms in dynamic logic, let's assume we know P and think about how $\alpha\ ;\ \beta$ executes. First, we run α. Anything we can know about the resulting state is implied by the $\mathsf{sp}(\alpha)P$. So anything we can know about the final state after β is the strongest postcondition for that.

$$\mathsf{sp}(\alpha\ ;\ \beta)P = \mathsf{sp}(\beta)(\mathsf{sp}(\alpha)P)$$

Perhaps not surprising in hindsight, that's just the opposite order of propagation from the weakest precondition.

Nondeterministic choice. We need to make sure the $\mathsf{sp}(\alpha \cup \beta)P$ is true no matter whether α or β is executed. Since we don't control which one, the best thing we can say is the disjunction of the two strongest postconditions.

$$\mathsf{sp}(\alpha \cup \beta)P = \mathsf{sp}(\alpha)P \vee \mathsf{sp}(\beta)P$$

Guard. The strongest postcondition of P should hold in every poststate of $?Q$, starting from a state where P is true. Since $?Q$ does not change the state, P will continue to be true. Furthermore, Q must be true if we are to reach the poststate at all, so:

$$\mathsf{sp}(?Q)P = Q \wedge P$$

At this point we can calculate the strongest postcondition of a conditional. Recall

$$\mathsf{if}\ Q\ \alpha\ \beta \triangleq (?Q\ ;\ \alpha) \cup (?\neg Q\ ;\ \beta)$$

Then

$$\begin{array}{rcl} \mathsf{sp}(\mathsf{if}\ Q\ \alpha\ \beta)P & = & \mathsf{sp}((?Q\ ;\ \alpha) \cup (?\neg Q\ ;\ \beta))P \\ & = & \mathsf{sp}(?Q\ ;\ \alpha)P \vee \mathsf{sp}(?\neg Q\ ;\ \beta)P \\ & = & \mathsf{sp}(\alpha)(\mathsf{sp}(?Q)P) \vee \mathsf{sp}(\beta)(\mathsf{sp}(?\neg Q)P) \\ & = & \mathsf{sp}(\alpha)(Q \wedge P) \vee \mathsf{sp}(\beta)(\neg Q \wedge P) \end{array}$$

Repetition. Again, we piece together the clause for repetition from nondeterministic choice, guard, and sequential composition. We don't treat invariants or variants here (see Exercise 3).

$$\mathsf{sp}(\alpha^*)P = P \vee \mathsf{sp}(\alpha^*)(\mathsf{sp}(\alpha)P)$$

Assignment. The case for assignment is again somewhat tricky. Let's assume our precondition is $P(x)$ and we assign to x. Now P no longer holds of x! Let's consider some examples:

$$\begin{array}{lcl} \mathsf{sp}(x \leftarrow 3)(x = 4) & = & x = 3 \\ \mathsf{sp}(x \leftarrow x + 1)(x = 4) & = & x = 5 \\ \mathsf{sp}(x \leftarrow x + 1)(0 \leq x \leq 3) & = & 1 \leq x \leq 4 \end{array}$$

We see from the second and third example, that we cannot lose the information from P entirely, but it is transformed. We need to know that it holds for the value of x before we carried out the assignment! Since we have no concrete way of referring to the old value of x, we have to say that there exists some old value x', and what the know about x' comes from the relationship established by the assignment. That is:

$$\mathsf{sp}(x \leftarrow e(x))(P(x)) = \exists x'.x = e(x') \wedge P(x') \quad (x' \notin e(x), P(x))$$

Let's revisit the examples:

$$\begin{array}{lcll} \mathsf{sp}(x \leftarrow 3)(x = 4) & = & \exists x'.\, x = 3 \wedge x' = 4 & \text{iff}\ \ x = 3 \\ \mathsf{sp}(x \leftarrow x + 1)(x = 4) & = & \exists x'.\, x = x' + 1 \wedge x' = 4 & \text{iff}\ \ x = 5 \\ \mathsf{sp}(x \leftarrow x + 1)(0 \leq x \leq 3) & = & \exists x'.\, x = x' + 1 \wedge 0 \leq x' \leq 3 & \text{iff}\ \ 1 \leq x \leq 4 \end{array}$$

Unlike in the case of the weakest precondition, we cannot always eliminate the existential quantifier over x' because we may not be able to invert the equation $x = e(x')$. This is often cited as the reason by weakest precondition calculations are preferred over strongest postconditions: we can always eliminate the universal quantifier of the former, but not the existential quantifier of the latter.

Nevertheless, because we follow the execution of the program in construction of the strongest postcondition it encapsulates *symbolic execution* which is important in program analysis tools and compiler optimization. This can be seen most clearly when we just use the ordinary $P \rightarrow [\alpha]Q$ or $P \rightarrow \langle\alpha\rangle Q$ and pack P with precise information about all variables that might be changed by Q. In that situation, the existential quantifier of the pure strongest postcondition can be eliminated because $P(x')$ in the formula determines x' uniquely to be the value of x' in the prestate. This is developed in some detail by Platzer [Pla04].

4.1 A New Modality in Dynamic Logic

Recall the semantic definition of the strongest postcondition:

$\nu \models \mathsf{sp}(\alpha)P$ iff there exists an ω such that $\omega \models P$ and $\omega[\![\alpha]\!]\nu$

We can see that, if we think of $\mathsf{sp}(\alpha)$ as a modality then it "looks into the past" because the truth of proposition in a state ν depends on the truth of P in a prestate of ω of ν under α. Because is requires the existence of a state it is a relative of the modal operator $\blacklozenge A$ which is true if A was true at some time in the past. By analogy, we write $\langle\!\langle\alpha\rangle\!\rangle P$. The semantic definition is exactly the one above:

$\nu \models \langle\!\langle\alpha\rangle\!\rangle P$ iff there exists an ω such that $\omega \models P$ and $\omega[\![\alpha]\!]\nu$

So what are the laws for $\langle\!\langle\alpha\rangle\!\rangle P$? This time, we can write the out following the intuition for the calculation of the strongest postcondition. Of course, this does not constitute a proof of their validity, but we have recruited Why3 to do that for us (shown in the next section). Given their validity we can then post-hoc justify the rules above for calculating the strongest postcondition.

$$
\begin{array}{lcll}
\langle\!\langle\alpha\,;\,\beta\rangle\!\rangle P & \leftrightarrow & \langle\!\langle\beta\rangle\!\rangle(\langle\!\langle\alpha\rangle\!\rangle P) & \\
\langle\!\langle\alpha\cup\beta\rangle\!\rangle P & \leftrightarrow & \langle\!\langle\alpha\rangle\!\rangle P \vee \langle\!\langle\alpha\rangle\!\rangle P & \\
\langle\!\langle ?Q\rangle\!\rangle P & \leftrightarrow & Q \wedge P & \\
\langle\!\langle\alpha^*\rangle\!\rangle P & \leftrightarrow & P \vee \langle\!\langle\alpha^*\rangle\!\rangle(\langle\!\langle\alpha\rangle\!\rangle P) & \\
\langle\!\langle x \leftarrow e(x)\rangle\!\rangle P(x) & \leftrightarrow & \exists x'.\, x = e(x') \wedge P(x') & (x' \notin e(x), P(x))
\end{array}
$$

Each of these can be proved from the semantic interpretation. We show the last one in some detail.

"$\longrightarrow$"

$\nu \models \langle\!\langle x \leftarrow e(x)\rangle\!\rangle P(x)$	Assumption
$\omega[\![x \leftarrow e(x)]\!]\nu$ and $\omega \models P(x)$ for some ω	By definition
$\omega[x \mapsto \omega[\![e(x)]\!]] = \nu$	By definition of $[\![x \leftarrow e(x)]\!]$
Let $a = \omega(x)$	
$\omega[\![e(x)]\!] = \omega[\![e(a)]\!]$	By definition of $\omega[\![e(x)]\!]$
$\omega[x \mapsto \omega[\![e(a)]\!]] \models x = e(a)$	By definition of $=$ and $x \notin e(a)$
$\omega \models P(a)$	Since $\omega \models P(x)$
$\omega[x \mapsto \omega[\![e(x)]\!]] \models P(a)$	Since $x \notin P(a)$
$\omega[x \mapsto \omega[\![e(x)]\!]] \models x = e(a) \wedge P(a)$	By definition of $\wedge$
$\nu \models x = e(a) \wedge P(a)$	By equality for ν
$\nu \models \exists x'.\, x = e(x') \wedge P(x')$	By definition of $\exists$

"$\longleftarrow$"

$\nu \models \exists x'.\, x = e(x') \wedge P(x')$	Assumption
$\nu[x' \mapsto a] \models x = e(x') \wedge P(x')$ for some a	By definition of $\exists$

$\nu[x' \mapsto a] \models x = e(x')$ and $\nu[x' \mapsto a] \models P(x')$	By definition of $\wedge$
$\nu[x \mapsto a] \models P(x)$	By renaming
$\nu(x) = \nu[x' \mapsto a][\![e(x')]\!]$	By definition of $=$
$\nu(x) = \nu[x \mapsto a][\![e(x)]\!]$	By renaming
$\nu[x \mapsto a][x \mapsto \nu[x \mapsto a][\![e(x)]\!]] = \nu$	From previous line
$\nu[x \mapsto a][\![x \leftarrow e(x)]\!]\nu[x \mapsto a][x \mapsto \nu[x \mapsto a][\![e(x)]\!]]$	By definition of $[\![x \leftarrow e(x)]\!]$
$\nu[x \mapsto a][\![x \leftarrow e(x)]\!]\nu$	From above by equality
Let $\omega = \nu[x \mapsto a]$	
$\omega \models P(x)$ and $\omega[\![x \leftarrow e(x)]\!]\nu$	From above by equality
$\nu \models \langle\!\langle x \leftarrow e(x) \rangle\!\rangle P(x)$	By definition of $\langle\!\langle - \rangle\!\rangle$

We conjecture that $\langle\!\langle \alpha \rangle\!\rangle P$ is a new modality, not already expressible in dynamic logic, which is suggested by Streett's result in the case of propositional dynamic logic [Str82]. Moreover, we can easily check that

$$\models P \rightarrow [\alpha]Q \quad \text{iff} \quad \models \langle\!\langle \alpha \rangle\!\rangle P \rightarrow Q$$

simply by expanding the definitions so it is related to the prior modality in an interesting way.

5 Verification in Why3

We have extended our development of dynamic logic in Why3 to include the black diamond modality $\langle\!\langle \alpha \rangle\!\rangle P$ and proved the axioms correlating to the computation of the strongest postcondition with the exception of the axiom for assignment.[1]

We begin with the semantic definition of `BDia`, our name for black diamond.

```
axiom models_bdia : forall nu alpha p.
  models nu (BDia alpha p)
  <-> (exists omega. models omega p /\ run omega alpha nu)
```

Proofs of the axioms (again, except assignment) are straightforward and can easily be found automatically by Why3.

```
lemma bdia_guard : forall nu q p.
  models nu (BDia (Guard q) p) <-> models nu (And p q)

lemma bdia_union : forall nu alpha beta p.
  models nu (BDia (Union alpha beta) p)
  <-> models nu (BDia alpha p) \/ models nu (BDia beta p)

lemma bdia_seq : forall nu alpha beta p.
  models nu (BDia (Seq alpha beta) p)
  <-> models nu (BDia beta (BDia alpha p))

lemma bdia_star : forall nu alpha p.
  models nu (BDia (Star alpha) p)
  <-> models nu p \/ models nu (BDia (Star alpha) (BDia alpha p))
```

[1] This involves reasoning with variable renaming and freshness and is in progress as of the writing of these notes.

6 Verification of Assignment

We also completed our treatment of the $[\alpha]Q$ modality (corresponding to the weakest precondition) by formalizing the proof of the assignment axiom

$$\omega \models [x \leftarrow e]Q(x) \leftrightarrow \forall x'.\, x' = e \rightarrow Q(x') \quad (x' \notin e, Q(x))$$

The complication here are the side conditions. We would try to formalize these condition by defining when $x' \notin e$ and $x' \notin Q$ and then proving some lemmas regarding freshness and renaming. This seems doable, but tedious. One of the difficulties here is that our very simple model of the state in the theory of extensional arrays no longer works, because we kept the domain of variables abstract. We would now need to know that there are unboundedly many variables, and that they have enough structure that we can always find fresh ones.

This syntactic approach would be a bit at odds with the semantic approach we have taken throughout and seems it is situated at the wrong level of abstraction. A more abstract formulation explicitly states the properties we need for freshness and renaming as axioms and carries out the proofs from those axioms. If we ever decide to give a concrete executable version of dynamic logic we would clone the module, fixing (for example) variables to be strings or integers, and we would then have to define functions that can guarantee freshness and proper renaming. This means we would have to prove the axioms for any concrete implementation. This approach is inspired by Platzer's Uniform Substitution Calculus [Pla17], although we use only a tiny fragment of it here.

We define, semantically, that a variable x' is fresh for an expression e if the value of e in any state ω does not depend on the value of x'.

> fresh x' e iff $\omega[\![e]\!] = \omega[x' \mapsto a][\![e]\!]$ for all ω and a.

This has some interesting consequences, for example, that x' is fresh in $x'-x'$. Moreover, freshness will be undecidable. On the other hand, the proof of the axiom only depends on this property and not on any more syntactic condition. Furthermore, simple syntactic conditions (like checking for syntactic occurrence of x') will imply the semantic condition we postulate.

Similarly, we state the property of renaming we need for the proof. We write this as rename x x' P P' when P' is the result of renaming x to x' in P.

> rename x x' P P' iff ($\omega[x \mapsto a] \models P$ iff $\omega[x' \mapsto a] \models P'$) for all ω and a

These can be straightforwardly transliterated into Why3 as axioms.

```
predicate fresh_exp (x:var) (e:exp)
predicate rename_ndl (x:var) (x': var) (p:ndl) (p':ndl)

axiom fresh_exp_stable : forall x' e.
  fresh_exp x' e
  <-> forall omega a. eval omega e = eval (write omega x' a) e

axiom rename_ndl_stable : forall x x' q q'.
```

```
9   rename_ndl x x' q q'
10  <-> forall omega a. models (write omega x a) q <-> models (write
        omega x' a) q'
```

With that, we can now state and prove the soundness of the dynamic logic axiom for assignment.

```
1   lemma box_assign : forall omega x e q x' q'.
2     fresh_exp x' e /\ rename_ndl x x' q q' ->
3     (models omega (Box (Assign x e) q)
4      <-> models omega (Forall x' (Implies (Equal (Var x') e) q')))
```

Exercises

Exercise 1.

1. Prove the soundness of the rule for sequential composition in Hoare logic, that is, $\models P \to [\alpha]R$ and $\models R \to [\beta]Q$ then $\models P \to [\alpha\,;\,\beta]Q$
2. Show via a counterexample that we cannot formulate this as purely logical question in NDL in the form of $\models (P \to [\alpha]R) \wedge (R \to [\beta]Q) \to [\alpha\,;\,\beta]Q$
3. Give another rendering of the soundness of Hoare's rule in NDL and prove it correct.

Exercise 2. In this exercise we explore weakest preconditions for partial correctness (often called *weakest liberal precondition*) of loops.

1. Give a version of $\mathsf{wp}(\alpha^*)$ that corresponds to the induction axiom of NDL. Critique this rule and show how it may or may not be used to verify partial correctness of a small program.
2. Give a version of $\mathsf{wp}(\alpha^*_J)Q$ that presupposes a *loop invariant* J. We have annotated the syntax to make J explicit, corresponding to a `invariant { J }` annotation in Why3. Again, critique the rule and revisit your example or a more general form.
3. Prove the correctness of your definition from part 2 by relating it to the induction axiom with invariants in NDL.

Exercise 3. Proceed as in Exercise 2, but for the strongest postcondition $\mathsf{sp}(\alpha^*)P$.

References

[GC10] Mike Gordon and Hélène Collavizza. Forward with Hoare. In Cliff B. Jones, A.W. Roscoe, and Kenneth R. Wood, editors, *Reflections on the Work of C.A.R. Hoare*, chapter 5, pages 101–121. Springer, 2010.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Pla04] André Platzer. Using a program verification calculus for constructing specifications from implementations. Minor Thesis (Studienarbeit), University of Karlsruhe, Department of Computer Science, February 2004. URL: https://lfcps.org/logic/Minoranthe.html.

[Pla17] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *Journal of Automated Reasoning*, 59(2):219–265, 2017.

[Str82] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54:121–141, 1982.