Scribe: Tom Roeder, Chavdar Botev Lecturer: Dexter Kozen

1 Propositional Dynamic Logic

Pratt made several contributions in the area of dynamic logic.

- One of them is having formal semantics for Hoare Logic - a formal binary relation semantics for programs and propositions.

- The second contribution was the connection with modal logic. The partial correctness assertion $\{A\}c\{B\}$ can be written in dynamic logic as an implication $A \to [c]B$ involving the precondition A and the weakest precondition $wp(c, B)$ written as $[c]B$ - dynamic logic actually allows you to express the weakest precondition. This is the weakest precondition for the postcondition B under the command c and it behaves exactly as modal logic.

- The next major contribution is the separation between the propositional reasoning and first order reasoning. Hoare logic mixes these things up. Everything is over a first order domain, typically the natural numbers as seen in Winskell's book. You have some locations that can have some values assigned to them - these are the states. You have an assignment statement and Hoare's assignment axiom. Everything is closely tied to first order logic, and to the domain of computation that you are reasoning over. Dynamic logic, however, separates that interpretation over the domain of computation from the purely propositional reasoning about the structure of programs. When you look at the Hoare rules for the **while** loop

$$\frac{\{A \wedge b\}c\{A\}}{\{b\}\,\mathbf{while}\,b\,\mathbf{do}\,c\{A \wedge \neg b\}}$$

  That rule says nothing about how c is to be interpreted, nor what domain we are interpreting over. It can be computed over the real numbers, over the natural numbers, or over any domain whatsoever and this rule will still be valid. So, this is really propositional. Hoare logic doesn't distinguish between these two levels of reasoning. Dynamic does very nicely.

- The next and very beautiful contribution of Pratt and the other people that worked with him is that they came up with the system called PDL (Propositional Dynamic Logic). This contribution is the simplification of program operators to much, much simpler mathematical objects, mathematical operators. You don't see $\mathbf{while}\,b\,\mathbf{do}\,c$, you don't see $\mathbf{if}-\mathbf{then}-\mathbf{else}$ anymore. It's reduced to some very simple operators: the union operator, which is often called $+$, the composition operator (this is same as before), but then, instead of the **while** loop, we have this star thing, which is much simpler. Now, it turns out that with these much simpler operators (the same operators that you've seen in regular expressions) and with tests you can build the **while** loop from a much simpler thing. For example:

$$\mathbf{if}\,b\,\mathbf{then}\,p\,\mathbf{else}\,q \equiv b?;p + \overline{b}?;q$$

$$\mathbf{while}\,b\,\mathbf{do}\,p \equiv (b?;p)^* b?$$

Let me give you an idea of how we can handle Hoare logic in PDL.

1.1 Axioms of PDL

1. Axioms for propositional logic
2. $[\alpha](\varphi \to \psi) \to ([\alpha]\varphi \to [\alpha]\psi)$
3. $[\alpha](\varphi \wedge \psi) \leftrightarrow [\alpha]\varphi \wedge [\alpha]\psi$

4. $[\alpha \cup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
5. $[\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$
6. $[\psi?]\varphi \leftrightarrow (\psi \rightarrow \varphi)$
7. $\varphi \wedge [\alpha][\alpha^*]\varphi \leftrightarrow [\alpha^*]\varphi$
8. $\varphi \wedge [\alpha^*](\varphi \rightarrow [\alpha]\varphi) \rightarrow [\alpha^*]\varphi$

Rules of inference:

- (MP) $\dfrac{\varphi \quad \varphi \rightarrow \psi}{\psi}$
- (Gen) $\dfrac{\varphi}{[\alpha]\varphi}$

The first person who came up with these axioms was Segerberg. He had a faulty proof of completeness. The first correct proof of completeness was given by Parikh and independently given by Gabbay (maybe Parikh's was a little bit earlier). When I say completeness, I mean just propositional completeness: I'm abstracting away from the fact that we are reasoning over the domain of natural numbers; we are just reasoning purely propositionally where we interpret programs as binary relations on some set and propositions as subsets of that set. (Over all possible interpretations of this form) Under the appropriate semantics of these operators, this set of axioms is complete for deriving all tautologies.

I'd like to point out that some of these things come from propositional logic, so propositional logic is contained in the system - Axiom (1) and the rule of inference MP — Modus Ponens. We know that propositional logic is complete. Similarly, there are two axioms here that are purely modal logic axioms — (2) and (3); and the rule of inference Gen — the generalization rule. The rest of the rules are particular to PDL and there's roughly one axiom for each operator. Intuitively, (4) takes two programs and non-deterministically chooses whether to do α or β. It's necessary after non-deterministically choosing whether to do α or β that φ is true iff it's necessary after doing α that φ is true and it's necessary after doing β that φ is true. (5) is for the sequential operator. It is the PDL way of saying $wp(c_1;c_2,B) = wp(c_1, wp(c_2,B))$. (6) is for the test operator. It passes or fails depending whether ψ is true.

Finally, for star, we have two axioms that say star behaves as a reflexive and transitive closure. (7) says that it's necessary after any number of iterations, perhaps zero, that φ is true iff it's true after zero iterations and it's necessary after one iteration that it's still necessary after any finite number of iterations.

Number (8) is called the PDL induction axiom. If you look at it, it really looks like induction, as you know it. It says:

- If φ is true after zero iterations of α (that's the basis of the induction), and
- If after any number of iterations of α it's true that if φ is true, then it will be true after one more step (so that's the induction step), then
- From these two things, you can conclude that it is true after any number of steps.

In PDL you have

$$\varphi \wedge [\alpha^*](\varphi \rightarrow [\alpha]\varphi) \rightarrow [\alpha^*]\varphi$$

An induction, as you know it in natural numbers , would be:

$$(\psi(0) \wedge \forall n.\psi(n) \rightarrow \psi(n+1)) \rightarrow \forall n.\psi(n)$$

1.2 Equivalence with Hoare Logic

Using these axioms, you can actually prove all axioms of Hoare logic. Let's just go through a simple proof of one of the rules. Let's take the **while** rule. The first thing I am going to do is prove a *loop invariance rule*.

$$\frac{\varphi \to [\alpha]\varphi}{\varphi \to [\alpha^*]\varphi}$$

Using the loop invariance rule, you can prove the induction rule, and vice versa. So, these are equivalent in the presence of the other axioms of dynamic logic.

Proof. By the generalization rule

$$\frac{\varphi \to [\alpha]\varphi}{[\alpha^*](\varphi \to [\alpha]\varphi)}$$

We curry

$$(\varphi \wedge [\alpha^*](\varphi \to [\alpha]\varphi)) \to (\varphi \to [\alpha^*]\varphi)$$

and we get

$$([\alpha^*](\varphi \to [\alpha]\varphi)) \to (\varphi \to [\alpha^*]\varphi)$$

Now, we're done:

$$\frac{\dfrac{}{([\alpha^*](\varphi \to [\alpha]\varphi)) \to (\varphi \to [\alpha^*]\varphi)} \quad \dfrac{\varphi \to [\alpha]\varphi}{[\alpha^*](\varphi \to [\alpha]\varphi)}}{\varphi \to [\alpha^*]\varphi}\ MP$$

Intuitively, the loop invariance rule says that if φ is an invariant of α, then it's an invariant of α^*.

Let's use that to prove the **while** rule of Hoare Logic.

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\textbf{while}\, b\, \textbf{do}\, c\{A \wedge \neg b\}}$$

Let's translate this to dynamic logic.

$$\frac{A \wedge b \to [c]A}{A \to [(b?c)^*\overline{b}?](A \wedge \overline{b})}$$

Proof. We have by axiom (5)

$$[(b?c)^*\overline{b}?](A \wedge \overline{b}) \equiv [(b?c)^*][\overline{b}?](A \wedge \overline{b})$$

Then, we have

$$[(b?;c)^*](\overline{b} \to (A \wedge \overline{b})) \equiv [(b?;c)^*](b \vee A)$$

Now, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \wedge b \to [c]A}{A \to (b \to [c]A)}\ \textit{Curry}}{A \to [b?c]A}\ \textit{Axiom 6}}{A \to [(b?c)^*]A}\ \textit{Loop Invariance Rule}}{A \to [(b?c)^*](\overline{b} \to (A \wedge \overline{b}))}\ \textit{Monotonicity}}{A \to [(b?;c)^*][\overline{b}?](A \wedge \overline{b})}\ \textit{Equivalence of notations}}{A \to [(b?;c)^*;\overline{b}?](A \wedge \overline{b})}\ \textit{Combining boxes}$$

When you start working with dynamic logic, after a while, you begin to realize you only use the full power of modalities very rarely. A lot of the time, you do a lot of reasoning about the equivalence of programs. It would be nice to reason about the equivalence of programs without using the full power of modalities. There's a nice system for doing that, which is called Kleene algebra.

2 Kleene algebra

In PDL, programs p and q are equivalent (representing the same binary relation under all interpretations) if $\vdash [p]A \equiv [q]A$ is provable for an atomic proposition symbol A. PDL is complete in this sense, i.e. if the programs represent the same binary relation, the above statement can be proven.

It turns out that to do that you only need some equational reasoning on basic program operators, and you need tests to define **while** loops and **if – then – else**. Kleene algebra is a system for doing that. This is the algebra of regular expressions.

2.1 Axioms of KA

1. $(p+q)+r = p+(q+r)$
2. $p+q = q+p$
3. $p+p = p$
4. $p+0 = p$
5. $(pq)r = p(qr)$
6. $p1 = 1p = p$
7. $p0 = 0p = 0$
8. $p(q+r) = pq+pr$
9. $(p+q)r = pr+qr$
10. $p^*q =$ least x such that$q+px \leq x$
11. $pq^* =$ least x such that$q+xp \leq x$
    $x \leq y$ iff $x+y=y$

The choice operator $+$ represents non-deterministic choice between two programs p and q. 0 is the fail program - it just fails. 1 is what is called **skip** in the Winskell book. It's the identity for concatenation. The set K of programs is an idempotent semi-ring under $+, ., 0, 1$. It's also a monoid under composition (concatenation). It's not commutative necessarily, but it's associative and has an identity.

The axiom for $*$ is the same as in PDL. It says that the linear inequality for x has a unique least solution and it's expressed as p^*q.

You also want to add tests so you can do programs. This is simply done by the Kleene algebra with tests.

2.2 Kleene algebra with tests (KAT)

$$KAT = (K, B, +, ., *, 0, 1, \text{-})$$

K are the programs, B are the tests, such that:

- $(K, +, ., *, 0, 1)$ is a KA.
- $(B, +, ., \text{-}, 0, 1)$ is a Boolean algebra.
- The semi-ring of $(B, +, ., 0, 1)$ is a subalgebra of $(K, +, ., 0, 1)$.

This may look a little weird because it looks like I'm overloading the symbols heavily. I am interpreting $+$ as non-deterministic choice, and . as a sequential composition of programs. I am also interpreting $+$ as disjunction, and . as conjunction. How can I do this safely? It turns out that there is no conflict. It works out very cleanly and very safely. If you think about a test in binary relations, elements of K might be interpreted as binary relations. Elements of B are interpreted as subsets of the identity relation. They are of the form $\{(u,u)|u \in A, A \subseteq B\}$. A boolean element is a test that may pass or fail depending whether u is in A. Sequential composition of boolean elements of that form is like testing two guards sequentially. It's the same as testing their intersection, the and of the two things. Testing one or the other is like testing their join.

So, we have a very succinct notation for doing all the stuff we are doing in Hoare Logic and PDL.

2.3 Sample proofs

Let's look at the axiom for $*$ operator.

- $1 + aa^* = a^*$
- $b + ax \leq x \rightarrow a^*b \leq x$

We have another axiom, equivalent to the second:

$$ax \leq x \rightarrow a^*x \leq x$$

This is equivalent to the loop invariant axiom in dynamic logic.

The system is complete over regular sets. So, here are some axioms over regular sets:

- $(ab)^*a = a(ba)^*$ - the sliding rule
- $(a+b)^* = (a^*b)^*a^*$

Let me give you an example of a proof in this system. Let's prove one direction of the sliding rule $(ab)^*a \leq a(ba)^*$, because it's symmetric. It's suffices to show that $a + ab(a(ba)^*) \leq a(ba)^*$. I just plugged in ab for a and p for q. Now, we have $a + aba(ba)^* = a(1 + ba(ba)^*) = a(ba)^*$ $\square$

The partial correctness assertion $\{b\}p\{c\}$ is encoded by $bp\overline{c} = 0$, which is equivalent to $bp = bpc$. Intuitively, the first says that there is no execution where the first state satisfies b and the second state satisfies $\overline{c}$. The second says that if you do p with a precondition b then the condition c afterwards is always redundant.

Let's take the **while** loop.

$$\frac{\{bc\}p\{c\}}{\{c\}\textbf{while}\, b\, \textbf{do}\, p\{\overline{b}c\}}$$

If we translate this to KA, we get:

$$bcp = bcpc \rightarrow c(bp)^*\overline{b} = c(bp)^*\overline{b}\overline{b}c$$

This is a universal Horne formula. All the symbols are universally quantified. This can be done in 5-6 lines.

One nice thing is that it is P-complete, as is propositional logic. PDL is exponential time complete. Hoare logic is trivially incomplete because it can only go from simpler programs to more complicated programs. You can prove this incompleteness, for example by considering:

$$\frac{\{b\}\textbf{if}\, c\, \textbf{then}\, p\, \textbf{else}\, p\{d\}}{\{b\}p\{d\}}$$

This is always true but you can not prove this in Hoare logic, since it doesn't allow you to go from more complicated premises to simpler conclusions. However, this is trivially provable in Kleene algebra.

$$b(cp + c\overline{p})\overline{d} = 0 \rightarrow bp\overline{d} = 0$$

It can be proven that KA is complete for any valid Hoare rule of the form

$$\frac{\{b_1\}p_1\{d_1\} \quad ... \quad \{b_n\}p_n\{d_n\}}{\{b\}p\{d\}}$$