The Expressive Power of the Modal μ-calculus $L\mu$

Matthew Wetmore

Abstract

A variety of modal logics are very useful for expressing properties of transition systems one would like to verify for the sake of program correctness. In this paper, we review the Propositional Dynamic Logic and discuss some of its strengths and limitations. We then introduce Dexter Kozen's modal μ-calculus $L\mu$, motivate its use of fixed-point operators, and show that $L\mu$ is strictly more expressive than PDL despite its relative syntactic simplicity.

1 Introduction

Labeled transition systems serve as a powerful abstraction of the behaviour of processes over time, including the execution of computer programs. Under such an abstraction, we view the state of our system at a given point as an element in a space of possible states, with a transition relation ordaining the possible ways to move between states upon occurrences of some set of actions.

By viewing our processes through the lens of transition systems, we get workable models for verifying the behaviours of our systems. A whole zoo of logics can be given meaning in terms of transition systems; this means we can use such a logic to write down properties of the transition system we would like to verify. Given that there exists an algorithm to decide whether or not a given system satisfies a particular formula, logics and transition systems give us convenient methods for verifying properties we would like our processes to have.

In particular, variants of propositional modal logic have proven quite useful. Such logics provide ways of reasoning about the truth of propositions over what may be thought of as possible futures – in our case, possible behaviours of a given process – and let us express properties such as "something eventually happens" or "something must happen". In 1983, Dexter Kozen introduced $L\mu$, a propositional modal logic whose possible formulas included fixed-point operators[Koz83]. As we will explore in this paper, the addition of such operators lead to a rich and expressive logic suitable for formal verification.

2 Propositional Dynamic Logic

To get a sense of why one would want fixed-point formulas in the first place, it is helpful to look at other logics predating the introduction of $L\mu$. One example is Propositional Dynamic Logic, or PDL, which gives us ways to build up simple programs and reason about their executions. Since it is a fairly natural way to express properties we'd like to verify in our programs, PDL is a useful logic, and has a few extensions with new allowable formulas and program connectives. These extensions add expressive power; one can introduce a looping operator Δ, for example, to capture the notion of infinite looping, which is impossible with vanilla PDL.

Kozen's $L\mu$ also allows us to talk about these notions, and as we'll be showing over the course of this paper, it gives us even more power than PDL, and at least as much power as PDL with looping.

2.1 Syntax

Propositional Dynamic Logic has the syntax one would expect of a propositional modal logic:

$$\varphi, \psi ::= p \mid \bot \mid \varphi \vee \psi \mid \neg\varphi \mid \langle\pi\rangle\varphi$$

All the usual suspects are here: atomic propositions $p \in AP$ are formulas, as is $\bot$[1]. We can apply the familiar connectives of disjunction and negation to formulas, and by de Morgan duality we can obtain conjunctions since $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$. Of course, we have modalities as well – along with $\langle\pi\rangle\varphi$ we have its dual $[\pi]\varphi = \neg\langle\pi\rangle\neg\varphi$.

What makes PDL special are the labels inside the modalities – π represents a (possibly non-deterministic) program, and we read $\langle\pi\rangle\varphi$ as "*some* terminating execution of π results in a state in which φ holds". Dually, $[\pi]\varphi$ is read as "*every* execution of π ends in a state in which φ holds". If these were just basic programs with no inherent structure, than the logic would not let us express that much. However, PDL lets us construct complex programs with operations familiar from regular expressions.

Assuming we have a set of basic programs, say $a, b, c, \ldots \in \Pi$, we can define possible programs in PDL with the grammar:

$$\pi_1, \pi_2 ::= a \mid \pi_1 \cup \pi_2 \mid \pi_1; \pi_2 \mid \pi_1^*$$

We interpret executions of these compound programs in ways their origins in regular expressions would suggest. An execution of $\pi_1 \cup \pi_2$ is a non-deterministic execution of either

[1] We mention $\bot$ separately from the atomic propositions since we consider atomic propositions as properties of states. Semantically we say no states satisfy $\bot$, i.e. $\bot$ is not a property of any state.

π_1 or of π_2. This is called "choice"; an execution of a "composition", or $\pi_1;\pi_2$, is an execution of π_1, then an execution of π_2. Finally, executing the Kleene star program π^* amounts to executing π a finite number of times (possibly 0). This captures the notion of iteration. Crucially, these constructs are not strong enough to talk about infinite looping; this limitation motivates the looping operator Δ mentioned earlier, and is one reason we'll be able to show that $L\mu$ is strictly more expressive than PDL.

2.2 Semantics

Since logics like PDL arose from a need to talk about transition systems, naturally we give the semantics of PDL in terms of transition systems whose transitions are labeled with our basic programs $a, b, \ldots \in \Pi$. If we consider these basic programs as the atomic actions in our system – say, assigning a value to a variable, or performing a primitive operation like addition – then we can look at a transition system whose state space consists of all states reachable from a set of initial states by performing these atomic actions. In this setting, compound programs like a^* let us express properties that may hold after multiple actions are performed.

We can specify a transition system with a set of states S, a set of starting states $S_0 \subseteq S$, a set of labels for our transitions $\mathcal{L}$, and a transition relation $\rightarrow \subseteq S \times \mathcal{L} \times S$ (if $(s, a, t) \in \rightarrow$ we write $s \xrightarrow{a} t$, i.e. "it is possible to make an a transition from s to t"). Hence a transition system is a quadruple $TS = (S, \rightarrow, S_0, \mathcal{L})$. In our case, since the transition relation is labeled by atomic programs/actions, the set of labels $\mathcal{L}$ is our set of atomic programs Π.

This is enough to describe the structure of the system, but if we want to talk about what atomic propositions hold in what states, we need to introduce a denotation $D : AP \rightarrow 2^S$ as well, mapping atomic propositions to the states in which they hold. We define the semantics of a formula φ in terms of states in which the formula holds, a set which we will denote $[\![\varphi]\!]_D^{TS}$, or the *interpretation* of φ – since this set depends on both the transition system and denotation, we should carry them around in the notation, but usually it is clear what they are and we drop them, simply writing $[\![\varphi]\!]$.

In order to formalize our notions of executing a program inside a modality, we will inductively define a program valuation ρ as well[2], where $\rho(\pi)$ maps the well-formed program π to a set of pairs (s, t) such that a terminating execution of π from state s results in state

[2]Note that the definition of ρ will depend on a given transition system. We could write ρ_{TS} to make this obvious, but it's not particularly nice to look at.

t. We define ρ as follows:

$$\begin{aligned}\rho(a) &= \{(s,t) \mid s \xrightarrow{a} t\} \\ \rho(\pi_1 \cup \pi_2) &= \rho(\pi_1) \cup \rho(\pi_2) \\ \rho(\pi_1;\pi_2) &= \{(s,t) \mid \exists (s,x) \in \rho(\pi_1) \text{ and } \exists (x,t) \in \rho(\pi_2)\}\end{aligned}$$

In order to define $\rho(\pi^*)$, we will let $\rho(\pi^0) = \{(s,s) \mid s \in S\}$ for any program π, and define $\rho(\pi^{n+1}) = \rho(\pi;\pi^n)$. This lets us write

$$\rho(\pi^*) = \bigcup_{n=0}^{\infty} \rho(\pi^n)$$

to define ρ for iterated programs. Now we are set to define the semantics of formulas; we define $[\![\varphi]\!]_D^{TS}$ inductively as follows:

$$\begin{aligned}[\![p]\!]_D^{TS} &= D(p) & [\![\varphi \vee \psi]\!]_D^{TS} &= [\![\varphi]\!]_D^{TS} \cup [\![\psi]\!]_D^{TS} \\ [\![\bot]\!]_D^{TS} &= \varnothing & [\![\neg\varphi]\!]_D^{TS} &= S \setminus [\![\varphi]\!]_D^{TS} \\ [\![\langle\pi\rangle\varphi]\!]_D^{TS} &= \{s \in S \mid \exists t \in [\![\varphi]\!]_D^{TS} \text{ s.t. } (s,t) \in \rho(\pi)\}\end{aligned}$$

With the semantics defined, we can say what it means for a state to satisfy a formula; we write $s \models \varphi$ if $s \in [\![\varphi]\!]$, and if $s \models \varphi$ for every initial state $s \in S_0$, we may extend our notion of satisfaction to the whole transition system TS, writing $TS \models \varphi$.

Our presentation of the syntax and semantics is a slightly modified take on the approach outlined in Michael Fischer and Richard Ladner's original paper[FL79] on PDL, with some inspiration from another paper on PDL[Ber81].

2.3 The small model property

When we wrote $s \models \varphi$, we left it unsaid that such a judgement depends on some TS and D; when we wrote $TS \models \varphi$ the denotation D was left implicit as well. It would be more proper to write $(TS, D) \models \varphi$, but often when we talk about a given transition system, we also fix a particular D, so we drop it in our notation.

The tuple (TS, D) contains all the information about a structure needed to interpret formulas in PDL; (TS, D) is a *model* in PDL. In the literature on PDL, models are often triples (S, D, ρ), where the structure of the transition system is determined by the states S and relations in ρ. In order to maintain similarity between models in PDL and models we will see later for $L\mu$, we've defined ρ in terms of a transition system TS instead, so our

models look slightly different, but still encode the same information. For the rest of this paper, we will use "transition system" and "model" interchangeably and refer to models with TS, with some arbitrary denotation D left implicit.

While the syntax of PDL allows us to write down a fairly large variety of formulas, they might not all be satisfiable; for example, the formula $\varphi \wedge \neg\varphi$ can never be satisfied by any state, no matter the model. We say such a formula is not *satisfiable*; a formula φ is satisfiable if there exists a model TS such that $TS \models \varphi$.

PDL has a rather desirable property known as the *small model property*, which states that is φ is a satisfiable PDL formula, there exists a finite model TS (that is, $|S| < \infty$) which satisfies φ. The size of this finite model is proportional to the size of the particular formula, which makes satisfiability decidable given that model-checking is decidable (check every model up to the finite bound)[3]. A number of logics have the small model property; many modal logics do.

In fact, PDL has a slightly stronger property, often called the finite quotient property, which we will see as we prove the small model property. The small model property is shown by using a standard technique called *filtration*, in which a model is "filtered through" a particular set of formulas to give an equivalent finite model.

Theorem (Small model property for PDL)**.** *If φ is a satisfiable theorem in PDL, then there exists a finite model TS such that $TS \models \varphi$.*

Proof. Let TS' be a model satisfying φ, with state space S'. We will use TS' to construct a model TS which also satisfies φ. Let Γ be the Fischer-Ladner closure of φ; this consists of the sub-formulas of φ and their negations. For modalities, $\langle\pi_1\rangle\varphi$ and $\langle\pi_2\rangle\varphi$ are sub-formulas of $\langle\pi_1 \cup \pi_2\rangle\varphi$, $\langle\pi_2\rangle\varphi$ as a sub-formula of $\langle\pi_1;\pi_2\rangle\varphi$, and $\langle\pi;\pi^*\rangle\varphi, \langle\pi\rangle\varphi$ as sub-formulas of $\langle\pi^*\rangle\varphi$. The size of Γ is proportional to the length of φ, so it's finite. Now define an equivalence relation $\sim$ on S' by letting $s \sim t$ if $\forall\psi \in \Gamma, s \models \psi \iff t \models \psi$. There are at most $2^{|\Gamma|}$ equivalence classes in $S'/\!\sim$, since each sub-formula has only 2 possible truth values. If we let $S = S'/\!\sim$, we can define transitions between equivalence classes such that $[s] \xrightarrow{a} [t]$ if there exist $s' \sim s$ and $t' \sim t$ such that $s' \xrightarrow{a} t'$.

The rest of the proof shows that $TS \models \varphi$ in a fairly straightforward case analysis, omitted for brevity's sake. The full proof is given in [FL79]. ■

The transition system constructed in the proof of the small model property is the finite quotient we mentioned before. The quotient transition system can be interpreted in a fairly intuitive way; we identify similar states, preserving the transitions, so if we have an idea

[3] A careful reader will notice that this assumes we can recursively enumerate the possible models. We can do this because if we want to generate models which may satisfy φ, we only need to worry about transition systems whose transition labels (i.e. atomic programs) only range over the set of atomic programs used in the modalities φ, which is finite. See discussion of decidability in chapter 6 of [BdRV02].

of the structure of the original transition system which satisfied a given formula, we can reason about the structure of the quotient system.

2.4 Other extensions of PDL

As we mentioned before, there are formulations of PDL which extend the logic with new operators. One such extension by Streett[Str82] adds a delta operator Δ, where $\Delta\pi$ is a formula (not a program) asserting that there is an infinite sequence of π executions – an infinite looping of the program. We add Δ semantically by saying $s \in [\![\Delta\pi]\!]$ if there exists a sequence of states $(s_n)_{n=0}^{\infty}$ such that $s_0 = s$ and $(s_n, s_{n+1}) \in \rho(\pi)$ for all n.

Adding this operator (obtaining "delta-PDL") gives a strictly more expressive logic; for example we can write $\neg\Delta a$, a formula which is true at s if and only if there are no infinite a paths starting at s. As we will see in the next section, such a notion is impossible to express with vanilla PDL. However, this expressive power comes at the expense of the finite quotient property – while delta-PDL satisfies the small model property, the proof does not use the same filtration technique.

3 The Modal μ-Calculus $L\mu$

Propositional Dynamic Logic gave us a nice way to reason about the behaviour of simple programs, but fell short when it came to expressing inherently infinite properties like looping. We saw a way to extend PDL to address this shortcoming. However, extensions to PDL only add syntactic complexity to a system which is already somewhat complex. A rough rule of thumb is that the syntactic complexity of a logic is inversely proportional to its usefulness – while this complexity allows greater levels of expressivity, it comes at the cost of making the logic harder to reason about, and adds more cases to proofs.

Kozen was not the first to use fixed-point operators in a logic; he notes that Dana Scott and Jaco de Bakker introduced the idea (in a paper which is unfortunately unpublished) in 1969, adding the least fixed-point operator μ to propositional modal logic. Pratt expanded on the idea by introducing $P\mu$[Pra81], where μ is a least-root operator instead of least fixed-point, in a boolean algebra. That is, given a boolean algebraic term $\varphi(\vec{X})$ with free variables $\vec{X} = (X_1, \ldots, X_k)$ which may take on term values, $\mu\vec{X}.\varphi(\vec{X})$ is the least vector of terms $\vec{\psi}$ such that $\varphi[\vec{\psi}/\vec{X}] = 0$, or false (so the least root[4] of $\varphi(\vec{X})$). Pratt's logic also enforced a particularly strong syntactic restriction on allowable formulas which rendered illegal a number of useful formulas.

[4]Least as in at the bottom of a lattice of roots generated by the partial order on terms such that $x \leq y$ if $x \vee y = y$.

The approach taken with $P\mu$ gave a syntactically simpler logic which subsumed PDL while maintaining the finite quotient property. However, upon its introduction, it was unclear whether $P\mu$ was strictly more expressive than PDL, and by rendering μ as a least-root operator instead of a least fixed-point operator, Pratt's logic left behind a number of proof techniques applicable to least fixed-points. Kozen's $L\mu$ relaxed the syntactic restraints and represented μ as a least fixed-point operator, and upon its introduction it was shown that $L\mu$ was strictly more expressive than PDL.

3.1 Syntax

The syntax of $L\mu$ is, for the most part, familiar from our discussion of PDL:

$$\varphi, \psi ::= p \mid \bot \mid X \mid \varphi \vee \psi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mu X.\varphi(X)$$

As before, p is an atomic proposition in the set AP. Here, instead of programs, our modalities are labeled with actions from our labelling set $\mathcal{L}$, which will correspond to transition actions when we give the semantics.

The obvious difference which jumps out quite quickly is the addition of the least-fixed point operator μ from which the logic derives its name. However, its introduction is contingent on another feature which PDL lacked: formulas with variables which one may quantify over, which we will refer to as *parametric* formulas. Propositional variables like X may be added to formulas, and bound by a μ, an action which carries with it all the familiar notions of free variables, capture, scope, closed formulas, etc from first-order predicate logic. The manner in which these variables are interpreted depends on the semantics, which we will introduce soon.

In order for a parametric formula $\varphi(X)$ to appear in $\mu X.\varphi(X)$, it must satisfy an additional syntactic restriction known as *positivity* – that is, X must appear free, and in the scope of an even number of negations. Keeping this in mind, we note that the greatest fixed-point operator may be obtained as the dual of μ: $\nu X.\varphi(X) = \neg\mu X.\neg\varphi(\neg X)$.

3.2 Semantics

As with PDL, we give the semantics of $L\mu$ by defining inductively the sets of states at which a particular formula holds. Models of $L\mu$ will have the a similar structure – as before, we will have a transition system TS and denotation function $D : AP \to 2^S$, but we will also need a way to say what variables mean. To this end, we also require a function F mapping propositional variables like X to sets of states. We can write $F[X \mapsto T]$ to describe such a function

which agrees with F on every variable except possibly for X, which it maps to $T \subseteq S$.

$$\begin{aligned}
[\![p]\!]^{TS}_{D,F} &= D(p) & [\![\neg\varphi]\!]^{TS}_{D,F} &= S \setminus [\![\varphi]\!]^{TS}_{D,F} \\
[\![X]\!]^{TS}_{D,F} &= F(X) & [\![\langle a\rangle\varphi]\!]^{TS}_{D,F} &= \{s \in S \mid \exists t \in [\![\varphi]\!]^{TS}_{D,F} \text{ s.t. } s \xrightarrow{a} t\} \\
[\![\bot]\!]^{TS}_{D,F} &= \varnothing & [\![[a]\varphi]\!]^{TS}_{D,F} &= \{s \in S \mid \forall t. s \xrightarrow{a} t \implies t \in [\![\varphi]\!]^{TS}_{D,F}\} \\
[\![\varphi \vee \psi]\!]^{TS}_{D,F} &= [\![\varphi]\!]^{TS}_{D,F} \cup [\![\psi]\!]^{TS}_{D,F} & [\![\mu X.\varphi(X)]\!]^{TS}_{D,F} &= \bigcap\{T \subseteq S \mid T \supseteq [\![\varphi(X)]\!]^{TS}_{D,F[X\mapsto T]}\}
\end{aligned}$$

While we could have obtained $[\![[a]\varphi]\!]$ by writing $[a]\varphi = \neg\langle a\rangle\neg\varphi$ and computing, we include it here explicitly for perspicuity's sake. The notation is unfortunately heavy, but we can, as before, drop the annotations and simply write $[\![\varphi]\!]$.

To understand the semantics of fixed-point formula, it's easiest to consider $[\![\varphi(X)]\!]^{TS}_{D,F[X\mapsto T]}$ as the application of a function $(Q \mapsto [\![\varphi(X)]\!]^{TS}_{D,F[X\mapsto Q]}) : 2^S \to 2^S$ to T. For simplicity, we will consider formulas with only one propositional variable, and abuse the notation[5] a bit to write $(Q \mapsto [\![\varphi(X)]\!]^{TS}_{D,F[X\mapsto Q]})$ as $[\![\varphi(X)]\!]$. Hence, by the Knaster-Tarski theorem applied to the complete powerset lattice of sets of states, $[\![\mu X.\varphi(X)]\!]$ is the least fixed-point of the function $[\![\varphi(X)]\!]$, which can easily be seen to be monotonic when $\varphi(X)$ obeys the positivity restriction we enforced earlier.

In the general lattice-theoretic setting, if f is a monotonic function on a complete lattice, then $\mu f = \bigvee_n f^n(\bot)$. In our setting and notation, this means that $[\![\mu X.\varphi(X)]\!] = \bigcup_n [\![\varphi^n(\bot)]\!]$. This gives us an iterative method for computing the fixed-point by successively computing $[\![\varphi(\bot)]\!]$, $[\![\varphi(\varphi(\bot))]\!]$, and so on. This process takes at most $|S|+1$ iterations to converge to the actual fixed point, and if a state is in $[\![\mu X.\varphi(X)]\!]$, then it's in $[\![\varphi^n(\bot)]\!]$ for some n, and its inclusion depends on some state (or states) in $[\![\varphi^{n-1}(\bot)]\!]$. Hence for each state in $[\![\mu X.\varphi(X)]\!]$, there is a finite chain of "dependencies", from which we may reap an intuitive view of μ as expressing properties of some finite but possibly arbitrary-length character.

A concrete example of this intuition can be seen by computing the semantic meaning of the formula $\mu X.[a]X$. We start by plugging in $\bot$ for X: $[\![[a]\bot]\!] = \{s \in S \mid \forall t. s \xrightarrow{a} t \implies t \in [\![\bot]\!]\}$. Since $[\![\bot]\!]$ is empty, $[\![[a]\bot]\!]$ is the set of all states with no outgoing a transitions. Iterating, $[\![[a][a]\bot]\!] = \{s \in S \mid \forall t. s \xrightarrow{a} t \implies t \in [\![[a]\bot]\!]\}$, i.e. the set of all states whose a transitions all go to states with no outgoing a transitions. If a state is in $[\![[a][a]\varphi]\!]$, then it "depends" on the states in $[\![[a]\bot]\!]$ to which it may make an a transition.

In general, if $s \in [\![[a]^{n+1}\bot]\!]$, then every a path starting at s is length n. Hence a state satisfies $\mu X.[a]X$ if every a path starting at s is of finite length. In terms of the looping

[5] In the presence of a single variable, and with the understanding that we will be considering parametric formulas which will be bound by a fixed-point operator, this is not ambiguous.

operator for PDL we defined earlier, $\mu X.[a]X$ is equivalent to $\neg\Delta a$ – there are no infinite a paths starting from a state which satisfies either formula.

Our presentation of $L\mu$ is derived mostly from [BS06], as well as [Koz83], which introduced the logic.

4 Expressing PDL in $L\mu$

Since PDL and $L\mu$ are both modal logics over transition systems, we can discuss equivalence of formulas in either language. Two formulas, say φ in PDL and ψ in $L\mu$, are said to be equivalent if they agree on every model, i.e. $TS \models \varphi \iff TS \models \psi$. We can express PDL in $L\mu$, which means that every PDL formula has an $L\mu$ equivalent. Since PDL and $L\mu$ are largely similar in syntax and semantics, every translation of a formula without a modality is trivial, e.g. $p \vee q$ is the same in either logic. However, PDL has a structure to its modalities which $L\mu$ lacks, at least on the surface.

We can express the modalities of PDL in $L\mu$ by "unpacking" them. If we use $\langle\!\langle \pi \rangle\!\rangle$ to denote the $L\mu$ equivalent of $\langle \pi \rangle$, then we can inductively define this translation by $\langle\!\langle a \rangle\!\rangle \varphi = \langle a \rangle \varphi$, $\langle\!\langle \pi_1 \cup \pi_2 \rangle\!\rangle \varphi = \langle\!\langle \pi_1 \rangle\!\rangle \varphi \vee \langle\!\langle \pi_2 \rangle\!\rangle \varphi$ and $\langle\!\langle \pi_1; \pi_2 \rangle\!\rangle = \langle\!\langle \pi_1 \rangle\!\rangle \langle\!\langle \pi_2 \rangle\!\rangle \varphi$. So, for example, the translation of $\langle a; (b \cup c) \rangle \varphi$ into $L\mu$ is $\langle a \rangle (\langle b \rangle \varphi \vee \langle c \rangle \varphi)$. We could have made these translations in a bunch of modal logics, but expressing $\langle \pi^* \rangle \varphi$ requires the power of $L\mu$. This is done by letting $\langle\!\langle \pi^* \rangle\!\rangle \varphi = \mu X. \varphi \vee \langle\!\langle \pi \rangle\!\rangle X$.

To understand the translation, it helps to realize that programs define possible paths in the underlying transition system. Hence $\langle\!\langle \pi \rangle\!\rangle \varphi$ is satisfied at a state s if there is a path described by π from s to somewhere where φ holds. Since atomic programs correspond to single transitions, the translation of $\langle a \rangle \varphi$ from PDL doesn't change the formula. For the program $\pi_1 \cup \pi_2$, if we assume by the induction hypothesis that $\langle\!\langle \pi_1 \rangle\!\rangle \varphi$ and $\langle\!\langle \pi_2 \rangle\!\rangle \varphi$ are proper translations of the PDL formulas $\langle \pi_1 \rangle \varphi$ and $\langle \pi_2 \rangle \varphi$, then $\langle\!\langle \pi_1 \rangle\!\rangle \varphi \vee \langle\!\langle \pi_2 \rangle\!\rangle \varphi$ asserts there is a path described by π_1 or a path described by π_2 ending in a state in which φ holds, which is what defined $\langle \pi_1 \cup \pi_2 \rangle \varphi$. The translation of $\langle \pi_1; \pi_2 \rangle \varphi$ follows similarly.

The interesting case is the translation of $\langle \pi^* \rangle \varphi$ from PDL to $L\mu$. We can understand it in the same manner we used to understand $\mu X.[a]X$, via fixed-point iteration. Starting at the bottom, so to speak, $[\![\varphi \vee \langle\!\langle \pi \rangle\!\rangle \bot]\!]$ is the set of all states at which φ holds, or there is a path described by π ending in a state where $\bot$ holds. As there are no such paths ($\bot$ never holds), $[\![\varphi \vee \langle\!\langle \pi \rangle\!\rangle \bot]\!] = [\![\varphi]\!]$. Iterating, $[\![\varphi \vee \langle\!\langle \pi \rangle\!\rangle (\varphi \vee \langle\!\langle \pi \rangle\!\rangle \bot)]\!]$ is the set of all states which satisfy φ, or from which there is a path described by π to a state at which $\varphi \vee \langle\!\langle \pi \rangle\!\rangle \bot$ holds, i.e. a state at which φ holds.

In general, $s \models \mu X. \varphi \vee \langle\!\langle \pi \rangle\!\rangle X$ if and only if there is a sequence of paths described by π from s to a state satisfying φ. This is precisely the condition defining $\langle \pi^* \rangle \varphi$ in PDL.

5 $L\mu$ is strictly more expressive

Now that we know every PDL formula has an $L\mu$ equivalent, it's natural to ask if the converse is true. Having spoiled the answer already in the paper's abstract, it's time to give a proof of what we've claimed multiple times already. Kozen's $L\mu$ is strictly more expressive than PDL, in that there exist formulas in $L\mu$ with no PDL equivalent. This is a result of the following theorem:

Theorem. *The $L\mu$ formula $\mu X.[a]X$ has no PDL equivalent.*

Proof. Recall two formulas are equivalent if they have the same truth value on every model. Assume for a contradiction that φ is a PDL formula equivalent to $\mu X.[a]X$, so that $TS \models \varphi \iff TS \models \mu X.[a]X$. In particular, consider the transition system TS pictured in Figure 1. For any natural number n, the transition system contains an initial a path of length n, but there are no infinite paths. Hence $TS \models \mu X.[a]X$, and consequently $TS \models \varphi$. While TS is an infinite transition system, the small model property of φ allows us to filter the system and obtain TS_{FIN}, a finite transition system such that $TS_{FIN} \models \varphi$ as well.

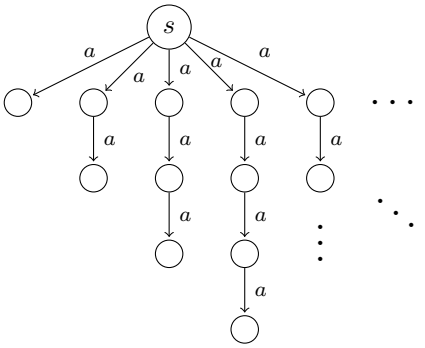

Figure 1: The transition system TS

However, since TS_{FIN} is finite, the filtration process must have introduced a loop. Suppose TS_{FIN} has n states and consider the a path of length n in TS, with states $s_0, \ldots, s_n$ where $s_0 = s$, such that $s_k \xrightarrow{a} s_{k+1}$ for $0 \leq k < n$. By the pigeonhole principle, at least 2 states, say s_i and s_j, for $i < j$, must be identified to a single state in TS_{FIN}, introducing a loop: in TS_{FIN} there is a path from $[s_i]$ to $[s_{j-1}]$, and $[s_{j-1}] \xrightarrow{a} [s_j]$, but $[s_j] = [s_i]$.

Since TS_{FIN} contains a loop, there is an infinite a path in TS_{FIN}. Hence we find that $TS_{FIN} \not\models \mu X.[a]X$ – a contradiction, since $TS_{FIN} \models \varphi$ and φ was assumed to be equivalent to $\mu X.[a]X$. ■

As we noted earlier, $\mu X.[a]X$ is equivalent to the delta-PDL formula $\neg\Delta a$, so this proof serves to show that delta-PDL is strictly more expressive than PDL as well (otherwise it wouldn't be a particularly useful extension!). In fact, while this proof appeared in Kozen's original paper on $L\mu$, [Koz83], he credits it to Streett, who introduced delta-PDL. While we've only looked into one example of expressing a particular modal logic in $L\mu$, one can express many more in $L\mu$ as well, such as CTL*, as well as its fragments CTL and and LTL. Such results lend further credence to our claim that $L\mu$ is an expressive, powerful logic.

References

[BdRV02] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2002.

[Ber81] Francine Berman. Semantics of looping programs in propositional dynamic logic. *Mathematical systems theory*, 15(1):285–294, 1981.

[BS06] Julian Bradfield and Colin Stirling. Modal mu-calculi. In J. van Benthem P. Blackburn and F. Wolter, editors, *The Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.

[FL79] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

[Koz83] Dexter Kozen. Results on the propositional -calculus. *Theoretical Computer Science*, 27(3):333 – 354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[Pra81] V.R. Pratt. A decidable mu-calculus: Preliminary report. In *Foundations of Computer Science, 1981. SFCS '81. 22nd Annual Symposium on*, pages 421–427, Oct 1981.

[Str82] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(12):121 – 141, 1982.