Affine Dynamic Logic

Sree Vishant Prabhakaran
sreevisp@andrew.cmu.edu

May 9, 2017

1 Abstract

d$\mathcal{L}$ as we have seen in class is one of many dynamic multimodal logic systems used to model and prove statements about various real-world scenarios. Changes in a d$\mathcal{L}$ system are typically modeled and represented via differential equations, where a variable will develop according to an ODE and a domain constraint. Normally, these systems are used to model physical equations of continuous motion with relative ease. However, systems with discrete methods of evolution where truth changes over time are not as straightforward to model in this semantics, as we will see. To solve this problem, we propose a dynamic logic in the style of affine logic in order to model truth and safety as resources that must be expended and evolve as a system evolves.

2 Introduction

We begin by considering the traditional Towers of Hanoi puzzle. In this puzzle, we have three towers and three disks of increasing size. Initially, all three disks are placed on the leftmost tower, with the smallest disk on the medium one, the medium one on top of the largest one, and the largest one at the base of the tower. The goal of this game is to move this stack to one of the other towers, while following the rules that only one disk can be moved at a time (because picking up and moving the whole stack would make for a rather uninteresting problem) and that no disk can be placed on top of a smaller disk. Suppose we want to show that we can achieve this goal from the initial starting state. In principle, we would like to prove a statement (using d$\mathcal{L}$ syntax) of the form:

$$P \rightarrow \langle Q^* \rangle S$$

where P represents the initial state of the game, Q is a nondeterministic choice over possible moves during the game, and S is the "winning" state with the stack of disks on a different tower.

Trying to represent this in traditional d$\mathcal{L}$ poses a challenge - how do we represent a state and the logic used to make moves? One idea would be to enumerate all possible configurations of the game, assign each one a unique number, and change the value of some variable x based on the possible states we could reach from a given state. For example, we could say $x = 1$ when we are in the initial configuration. Then we could reach, say, $x = 2$, representing the same state but with the smallest disk now at the base of the second tower, or $x = 3$ with the smallest disk at the third tower. It is possible to complete this model and prove that we can win the game, but it is clunky and reveals nothing about the actual behavior or logic of the game. In fact, by enumerating all of the states and their transitions beforehand, we have already essentially answered our question. Proving anything beyond this point would simply be a chore that tells us nothing we did not already know from having to enumerate the states in the first place. This example reveals a fundamental shortcoming of states as we are used to in d$\mathcal{L}$, containing only assignables and their associated numerical values.

In this paper, we will develop a new system of d$\mathcal{L}$ (hereafter referred to as "AD$\mathcal{L}$") using affine logic, a type of substructural logic, motivated by a need to address problems of this sort. In this system, states will be comprised of formulas rather than numerical variables, like logical contexts.

3 Affine Logic Basics

A fundamental drawback of classical logic is persistence of truth. Consider the following statements:

$$A = \text{"I have a dollar"} \quad B = \text{"I have a soda"}$$

Now suppose we have a dollar (A), and access to a vending machine ($A \to B$). By classical logic, since we have $A \wedge A \to B$, we have by equivalence $A \wedge B$. But this is clearly not how vending machines work! After we use the vending machine, A should not be true anymore. In this sense, we want truth to operate as a resource. That is, by "expending" A in the implication $A \to B$, only B should be true afterwards.

Affine logic (and by inclusion, linear logic) attempts to provide a solution to this inconsistency. The key idea behind affine logic is that it does not allow contraction as a substructural logic. In this sense, it is equivalent to linear logic with weakening, and in fact, the affine logic connective $A \to B$ can be expressed in linear logic as $A \multimap B \otimes \top$. For convenience, we will use linear logic principles to develop our theory, but it should be understood that the statements we want to prove about, for example, the Towers of Hanoi game, will be affine.

4 AD$\mathcal{L}$ Formulation

In this new theory, we will redefine the grammar of programs and formulas. Instead of hybrid programs, we will set up a framework to prove properties of linear logic programs:

$$\alpha ::= A \mid 1 \mid \alpha \otimes \alpha \mid \alpha \multimap \alpha \mid \alpha \mathbin{\&} \alpha \mid \alpha^*$$

And an AD$\mathcal{L}$ formula P is defined inductively as:

$$P ::= A \mid P^{\perp} \mid P \mathbin{\&} P \mid P \otimes P \mid P \to P \mid [\alpha]P \mid \langle\alpha\rangle P$$

In this logic, we reason with facts as opposed to assignables and real numbers, so we omit expressions of the form $e \geq e'$ or $\forall x P$ from the grammar for formulas. Another consequence of this is that states are now more similar to logical contexts than traditional d$\mathcal{L}$ states. We can think of these new states as partial functions from logical statements to natural numbers. Note that the $\perp$, & and $\otimes$ operators in formulas function exactly the same as $\neg$, $\vee$ and $\wedge$ from d$\mathcal{L}$ respectively. We choose to use these operators simply to distinguish formulas in AD$\mathcal{L}$ from d$\mathcal{L}$.

Returning to the soda example from the previous section now, suppose we wanted to prove that, with access to that vending machine, in a final state we can end up with either a soda or our original dollar (but not both). That is, we can choose to use the vending machine or not. The formula to express that with our new syntax would look like:

$$A \to [A \multimap B \mathbin{\&} 1](A \mathbin{\&} B)$$

It is of note that we can express this behavior in standard d$\mathcal{L}$ as well. Letting x be the number of dollars we have, and y be the number of sodas we have, then this can be reduced to:

$$x = 1 \wedge y = 0 \to [(x := x - 1; y := y + 1) \cup ?true](x = 1 \wedge y = 0) \vee (x = 0 \wedge y = 1)$$

5 Axioms and Proof Rules

We now lay out the basic axioms that define the mechanics of our new logic. As we did for d$\mathcal{L}$, we will begin by defining inductively what it means for a formula to hold in a state:

$$[\![A]\!] = \{\omega : \omega[\![A]\!] > 0\}$$

$$
\begin{aligned}
[\![P^{\perp}]\!] &= ([\![P]\!])^{C} \\
[\![P \mathbin{\&} Q]\!] &= [\![P]\!] \cup [\![Q]\!] \\
[\![P \otimes Q]\!] &= [\![P]\!] \cap [\![Q]\!] \\
[\![P \to Q]\!] &= \{\omega : \omega \not\models P \text{ or } \omega \models Q\} \\
[\![[\alpha]P]\!] &= \{\omega : \nu \models P \ \forall\ (\omega,\nu) \in [\![\alpha]\!]\} \\
[\![\langle\alpha\rangle P]\!] &= \{\omega : \nu \models P \text{ for some } (\omega,\nu) \in [\![\alpha]\!]\}
\end{aligned}
$$

Of course, these last two definitions require us to define $[\![\alpha]\!]$, so we will do this next:

$$
\begin{aligned}
[\![A]\!] &= \{(\omega,\nu) : \nu = \omega \text{ except that } \nu[\![A]\!] = \omega[\![A]\!] + 1\} \\
[\![1]\!] &= \{(\omega,\nu) : \nu = \omega\} \\
[\![\alpha \otimes \beta]\!] &= \{(\omega,\nu) : \exists\rho\ (\omega,\rho) \in [\![\alpha]\!] \text{ and } (\rho,\nu) \in [\![\beta]\!]\}
\end{aligned}
$$

$$
[\![\alpha \multimap \beta]\!] = \{(\omega,\nu) : \exists\rho\,(\omega,\rho) \text{ with } \rho = \omega \text{ except } \rho[\![A]\!] = \omega[\![A]\!]{-}1, \rho[\![A]\!] \geq 0 \text{ for each occurrence of A in } \alpha \text{ and } (\rho,\nu) \in [\![\beta]\!]\} \quad (1)
$$

$$
\begin{aligned}
[\![\alpha \mathbin{\&} \beta]\!] &= [\![\alpha]\!] \cup [\![\beta]\!] \\
[\![\alpha^{*}]\!] &= \bigcup_{n=0}^{\infty} [\![\alpha^{n}]\!], \text{ where } \alpha^{n} \equiv \alpha \otimes \alpha \otimes \ldots \otimes \alpha
\end{aligned}
$$

Now we introduce the basic proof rules that will allow us to reason about AD$\mathcal{L}$ programs and formulas.

$$
HM\otimes\ \frac{A \to [\alpha]E \qquad E \to [\beta]B}{A \to [\alpha \otimes \beta]B} \qquad [\&]\ [\alpha \mathbin{\&} \beta]P \leftrightarrow [\alpha]P \mathbin{\&} [\beta]P
$$

$$
[\otimes]\ [\alpha \otimes \beta]P \leftrightarrow [\alpha][\beta]P \qquad [*]\ [\alpha^{*}]P \leftrightarrow P \otimes [\alpha][\alpha^{*}]P \leftrightarrow P \otimes [\alpha^{*}](P \to [\alpha]P)
$$

$$
[\multimap]\ [\alpha \multimap \beta]P \leftrightarrow [\alpha \multimap 1][\beta]P
$$

Because we use linear logic instead of classical logic now, some of our logical properties must change as well:

$$(\cdot)^{\perp}L\ \frac{\Delta \Vdash B, \Gamma}{\Delta, B^{\perp} \Vdash \Gamma} \qquad (\cdot)^{\perp}R\ \frac{\Delta, B \Vdash \Gamma}{\Delta \Vdash B^{\perp}, \Gamma} \qquad 1L\ \frac{\Delta \Vdash \Gamma}{\Delta, 1 \Vdash \Gamma} \qquad 1R\ \frac{}{\Vdash 1}$$

$$\otimes L\ \frac{\Delta, B_1, B_2 \Vdash \Gamma}{\Delta, B_1 \otimes B_2 \Vdash \Gamma} \qquad \otimes R\ \frac{\Delta_1 \Vdash B, \Gamma_1 \quad \Delta_2 \Vdash C, \Gamma_2}{\Delta_1, \Delta_2 \Vdash B \otimes C, \Gamma_1, \Gamma_2}$$

$$\&L1\ \frac{\Delta, B_1 \Vdash \Gamma}{\Delta, B_1 \,\&\, B_2 \Vdash \Gamma} \qquad \&L2\ \frac{\Delta, B_2 \Vdash \Gamma}{\Delta, B_1 \,\&\, B_2 \Vdash \Gamma} \qquad \&R\ \frac{\Delta \Vdash B, \Gamma \quad \Delta \Vdash C, \Gamma}{\Delta \Vdash B \,\&\, C, \Gamma}$$

$$\textsc{cut}\ \frac{\Delta \Vdash B, \Gamma \quad \Delta', B \Vdash \Gamma'}{\Delta, \Delta' \Vdash \Gamma, \Gamma'} \qquad \to R\ \frac{\Delta \Vdash A \to B, \Gamma}{\Delta, A \Vdash B, \Gamma}$$

6 Simple Examples

Let us revisit the vending machine example from the formulation section and see if we can prove our original hypothesis.

$$A \to [A \multimap B \,\&\, 1](A \,\&\, B)$$

We begin by applying $\to R$, which brings us to a state ω where $\omega[\![A]\!] \geq 1$. Now we can use the semantic definitions of AD$\mathcal{L}$ formulas to break down the right-hand side. Applying [&]:

$$[A \multimap B \,\&\, 1](A \,\&\, B) \leftrightarrow [A \multimap B](A \,\&\, B) \,\&\, [1](A \,\&\, B)$$

The left-hand side of our new formula brings us to a state ν where $\nu[\![A]\!] \geq 0$ and $\nu[\![B]\!] \geq 1$, so $(\omega, \nu) \in [\![B]\!]$, and thus, is also in $[\![A \,\&\, B]\!] = [\![A]\!] \cup [\![B]\!]$. This is enough to prove the given statement, but we will show the other half as well for the sake of completeness. The right-hand side of this formula brings us to a state ρ where $\rho = \omega$, and we know that $\omega[\![A]\!] \geq 1$, so by the same logic as before, $(\omega, \rho) \in [\![A \,\&\, B]\!]$.

Let us now attempt another example. In finance, buying a put option gives you the right to sell a security at a specific agreed price (or not), as you prefer. We can express this as an AD$\mathcal{L}$ program and show that, after purchasing an option, we can always have the agreed amount afterwards. Using C to denote the cost of the option, S to denote the security, and P for the agreed price:

$$C \to \langle C \multimap (S \otimes ((S \multimap P) \,\&\, 1))\rangle P$$

Using our semantics as before, we apply $\to R$ to bring us to a state ω where $\omega[\![C]\!] \geq 1$. Now we apply the definition of $\multimap$ and reach a new state ν, with $\nu[\![C]\!] \geq 0$ and $\nu[\![S]\!] \geq 1$. Then by the semantics of $\langle\alpha\rangle P$ and &, we can simply take the left-hand choice of $S \multimap P$ from ν to end up in a state where P holds, as desired.

7 Towers of Hanoi, Revisited

We can now try to express the Hanoi problem in this new logic system. We start by defining some basic types and predicates on those types that will be useful in representing this problem.

`disk: type`: Used to represent disks

`smaller disk disk: predicate`: Is disk 1 smaller than disk 2?

`place: type`: Used to represent either the top of a disk or an empty tower

`tower: type`: Used to represent towers

`top_of disk: type`: Convert the top of a disk to a "place"

`bottom tower: place`: Convert the bottom of a tower to a "place"

`on disk place: predicate`: Is the disk on the place?

`clear place: predicate`: Does the place have nothing on top of it?

`arm_free: predicate`: Is the arm moving the disks free?

`arm_holding disk: predicate`: Is the arm holding the given disk?

We now express the rules of the game that allow us to make legal moves:

`pickup D P: clear(top_of D)` ⊗ `on D P` ⊗ `arm_free` ⊸ `arm_holding D` ⊗ `clear P`

`put_down_on_disk D D': arm_holding D` ⊗ `clear(top_of D')` ⊗ `smaller D D'` ⊸ `arm_free` ⊗ `clear(top_of D)` ⊗ `on D (top_of D')`

`put_down_on_tower D T: arm_holding D` ⊗ `clear(bottom T)` ⊸ `arm_free` ⊗ `clear(top_of D)` ⊗ `on D (bottom T)`

Then we can express the Hanoi game with disks $D1, D2, D3$ and towers A, B, C in our new logic as follows:

`smaller D1 D2` ⊗ `smaller D2 D3` ⊗ `on D1 (top_of D2)` ⊗ `on D2 (top_of D3)` ⊗ `on D3 (bottom A)` ⊗ `clear (top_of D1)` ⊗ `clear (bottom B)` ⊗ `clear (bottom C)` ⊗ `arm_free` →

⟨(`pickup D P` & `put_down_on_disk D D'` & `put_down_on_tower D T` & 1)*⟩

`(on D1 (top_of D2)` $\otimes$ `on D2 (top_of D3)` $\otimes$ `on D3 (bottom B)` $\otimes$ `clear (top_of D1)` $\otimes$ `clear (bottom A)` $\otimes$ `clear (bottom C))`

In English, this formula simply describes the general game of Hanoi - we provide an initial starting state, give the rules that allow us to develop the game, and then see if it is possible to reach the desired win condition. The proof from here is fairly straightforward, although tedious. We simply follow the steps that we would take to solve the Hanoi puzzle in real life and see that the postcondition is reachable.

8 Conclusion and Future Work

It appears that AD$\mathcal{L}$ as defined can solve a certain class of problems much more efficiently and elegantly than traditional d$\mathcal{L}$. As noted before, this is not to say that d$\mathcal{L}$ *cannot* handle such problems (as indeed, it can), but rather that proofs in AD$\mathcal{L}$ are more illustrative of the behavior of the original problem where d$\mathcal{L}$ would reduce it to something seemingly unrelated.

That said, there are many identifiable areas of improvement/expansion within this logic. Most notably, the computational complexity of programs in our model is not to be trifled with. In the Towers of Hanoi example, we typically have 3 possible moves in any given state. If we increased the number of disks or towers, reasoning about evolution of these states would grow exponentially more difficult. If we were able to develop an inductive method of reasoning about these programs, as opposed to the generative model shown here, it would be easier to reason about such problems and we could potentially solve entire classes of problems instead of specific examples.

Another omission from this treatment of linear logic programming is the $\oplus$ operator. This operator serves as a sort of dual to & in AD$\mathcal{L}$, and could be used to develop theory about dynamic two-player games in this model.