Abstract

We introduce an extension for Differential Dynamic Logic (dL) called Controller Aware dL. Controller Aware dL is designed to give programmers additional resources for defining and honoring the inherent hardware limitations of cyber-physical systems with minimal conceptual overhead. In addition to providing a formal syntax and semantics for Controller Aware dL, we define a translation from Controller Aware dL to dL and prove that it honors the semantics of both logics. This translation serves as the formal foundation that would be needed to implement a transpiler from Controller Aware dL to dL, enabling the practical verification of Controller Aware dL formulas.

1 Introduction

1.1 Motivation

One challenge that is highly prevalent in CPS modeling is the preservation of fidelity. Soundness results for various logics such as dL or dGL can ensure that proven formulas are genuinely valid, but proving a formula valid is useless if that formula does not meaningfully describe reality.

In one sense, there is no solution to this problem [2]. Modeling cyber physical systems is hard, and as long as we're attempting to model something as complex as real physics, there are bound to be fidelity issues. That said, even if it's impossible to get rid of fidelity issues entirely, perhaps we can make it easier to avoid some common errors.

Consider the following dL formula, taken from recitation 5 of this course:

$$\begin{aligned}
&l \leq x \leq r \wedge v \geq 0 \wedge T > 0 \wedge l + 2vT \leq r \\
&\big[\big(\textbf{if } x + vT < l \wedge v \leq 0 \vee x + vT > r \wedge v \geq 0 \textbf{ then } v := -v; \\
&t := 0; \{x' = v, t' = 1 \,\&\, 0 \leq t \leq T\}\big)^*\big] \\
&l \leq x \leq r
\end{aligned}$$

This formula is meant to model ping-pong in one dimensional space, where the goal is to ensure that the ball remains between positions l and r. Time controls are used to model the fact that the players may not be able to hit the ball exactly when it hits a precise distance. At first glance, it is not entirely obvious whether there are any critical fidelity issues.

But indeed, although we took care to model the fact that the players may be unable to hit the ball exactly when it hits a precise distance, we still assume that the players are able to return the ball at exactly its negative velocity. And in fact, the validity of this formula critically relies on this assumption. If the players were to accidentally increase the ball's speed by any amount, then the formula would no longer be valid.

Broadly speaking, it can be said that this example contained an issue of fidelity in which the model failed to accurately describe the real world. But more specifically, the fundamental problem was that our model failed to account for our real-world inability to perfectly implement the designed control. In particular, no technology can measure a projectile's velocity exactly, nor apply the perfect amount of force needed to achieve an exact velocity.

This shortcoming is hardly unique to our ping pong example. All cyber physical systems that are actually implemented in the real world have hardware limitations. And although it is relatively easy to imagine how we might modify the toy example above to resolve the identified fidelity issue,

the conceptual overhead required to accurately model the hardware limitations of a full-fledged cyber physical system only increases with the complexity of the system being verified.

Since all hardware has inherent limitations, modeling such limitations and verifying the safety of cyber physical systems in spite of them is a fundamental problem. To help programmers avoid fidelity issues stemming from the complexity of modeling hardware limitations, we propose an extension to dL called Controller Aware dL. We intend for Controller Aware dL to have all the same language features as dL, plus some additional features to make it easier for programmers to define and honor hardware limitations.

1.2 Key Ideas of Controller Aware dL

Broadly speaking, interaction between a cyber physical system's controller and the physical world can be classified into two general categories. First, the physical world can interact with the system's sensors, allowing the controller to measure and learn various facts about the world. The relationship between the actual interaction and the learned fact may be very direct, such as when thermal scanners are used to measure temperature, or it may require more inference, such as when cameras are used to estimate the positions of objects. In either case, we refer to such phenomena as measurement. Second, the controller can instruct its system to perform some sort of action, resulting in physical changes in the world. Regardless of the type of physical change that results, we refer to such phenomena as control.

In both measurement and control phenomena, hardware limitations are unavoidable. The purpose of Controller Aware dL is to provide features that make defining and enforcing these limitations straightforward and explicit. The central concepts that Controller Aware dL introduces to achieve this aim are measurable and controllable variables.

In the formal definition of dL, all variables are simply elements of some set $\mathscr{V}$. But in KeYmaera X, there exists a distinction between definitions and program variables. Definitions are treated as constants that can be read, but not modified, while program variables can be read or modified with any of the constructs of dL. Although the distinction between definitions and program variables is not strictly necessary (we could instead make everything a program variable and then simply choose to never modify some subset of them), it is helpful to the programmer because it gives them a way to prevent themselves from accidentally modifying something they shouldn't.

Measurable and controllable variables are meant to help the programmer in a similar manner. To model hardware limitations inherent to measurement phenomena, the programmer can designate certain variables as measurable. Doing so will give the programmer access to constructs for approximating the value of said variable within an interval of uncertainty that depends on the variable being measured, circumstances of the world, and hardware being used to conduct the measurement.

Likewise, to model hardware limitations inherent to control phenomena, the programmer can designate certain variables as controllable. Doing so will give the programmer access to constructs for attempting to set the value of the variable in question. While the programmer can designate the exact value that they aim to set the variable to, the actual value that the variable will receive will depend on the variable in question, the circumstances of the world, the hardware being used to affect the physical world, and of course, the value that the programmer is attempting to set the variable to. Although there are many factors that influence the potential outcome of measuring a measurable variable or setting a controllable variable, having built-in constructs for these operations in the language itself should minimize these considerations' conceptual overhead.

2 Syntax and Semantics

2.1 Syntax

The grammar for Controller Aware dL is given below. e, e_1, and e_2 denote terms, $x \in \mathscr{V}$ is used to denote variables, P and Q denote formulas, and α and β denote hybrid programs. Constructs that are particular to Controller Aware dL (i.e. that do not appear in ordinary dL) are given in red[1].

term	e	$::=$	$x \mid c \mid e_1 + e_2 \mid e_1 * e_2 \mid \sim x$
formula	P, Q	$::=$	$e_1 = e_2 \mid e_1 \leq e_2 \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid \forall x P \mid \exists x P \mid \langle \alpha \rangle P \mid [\alpha] P$
hybrid program	α, β	$::=$	$x := e \mid x := * \mid ?P \mid x' = e \;\&\; P \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid$ measure $x \mid$ set $x\ e$

The syntax of Controller Aware dL contains all the syntax of ordinary dL, plus three additional constructs. The first, $\sim x$, is meant to refer to the approximate or measured value of some variable x[2]. When Controller Aware dL was first conceived, it was imagined that the value of $\sim x$ would be nondeterministically chosen at each occurrence. However, there were several issues with this idea.

First, nondeterministically assigning a value to $\sim x$ at every occurrence would drastically complicate the semantics of Controller Aware dL. In the semantics of dL, terms have the type $\mathscr{S} \rightarrow \mathbb{R}$, which means they depend deterministically only on state. This makes defining the semantics of compositional terms like $e_1 * e_2$ straightforward. But to accommodate terms that are nondeterministically assigned values at each occurrence, it would be necessary to make terms elements of $\mathcal{P}(\mathscr{S} \times \mathbb{R})$, which would significantly complicate not only the semantics for terms, but also the semantics for formulas and hybrid programs.

In addition to significantly complicating the semantics, this decision would have extremely counterintuitive consequences. For example, the term $\sim x * \sim x$ would not necessarily be nonnegative. This is because if x is a measurable variable whose value is close to zero, then fallible measurements may be unable to determine whether its true value is positive or negative. Then, in the term $\sim x * \sim x$, the first occurrence of $\sim x$ might be nondeterministically assigned a negative value, while the second occurrence might be nondeterministically assigned a positive value, yielding a negative product. This is obviously undesirable.

So if the value of $\sim x$ shouldn't be nondeterministically chosen at each occurrence, when should it be chosen? One idea would be to automatically reassign the value of $\sim x$ any time the real variable x changes. This would resolve the issue of $\sim x * \sim x$ potentially receiving a negative value. However, this could easily lead to significant ambiguity. For instance, consider the hybrid program: $z := \sim x; ((x := 2) \cup (y := 2)); z := \sim x$. Should the second assignment of $z := \sim x$ be able to yield a different result from the first assignment, even if y is assigned to 2 rather than x? We believe that neither potential answer to this question is particularly satisfying, because the idea of automatically assigning the value of $\sim x$ is fundamentally ambiguous.

Therefore, to avoid any ambiguity in when $\sim x$ is assigned to, we include the construct "measure x". This hybrid program causes the value of $\sim x$ to be nondeterministically chosen based on the current state. After running "measure x", the term $\sim x$ is given a single, consistent value until "measure x" is run again. This allows the programmer to explicitly identify when their cyber physical system does or doesn't update its beliefs about particular variables based on new measurements.

[1] Technically, the nondeterministic assignment $x := *$ is not part of dL proper. It is instead a derived construct, that can be interpreted either as $x' = 1 \cup x' = -1$ or $x' = 1; x' = -1$. But it does no harm to view it as part of dL itself, and this choice simplifies some of the later discussion.

[2] Note that the $\sim$ operator can only be applied to a variable, not an arbitrary term. This is indicated in the syntax by describing the term as $\sim x$ rather than $\sim e$.

Finally, "set x e" is the Controller Aware dL construct for assigning to controllable variables. When a variable is assigned to via the ordinary assignment operator, "$x := e$", that variable receives the exact value of the expression e. This is fine when discrete assignment is used to model a physics phenomenon that is too rapid to bother modeling with a continuous program, but it is problematic when modeling the impact a controller has on the external world. The operation "set x e" allows the controller to attempt to assign the value of e to the variable x, subject to the inherent uncertainty of the system's fallible hardware.

2.2 Semantics of Constructs From dL

Since Controller Aware dL is meant to have all the same language features as dL, the semantics of Controller Aware dL ought to coincide with the semantics of dL. By this, we mean that if a Controller Aware dL formula is given that only uses the subset of the language that can be written in dL, then the validity of that formula ought to be the same regardless of whether it is interpreted as a dL formula or Controller Aware dL formula. The simplest way to do this is to leave the semantics of all old constructs unmodified, but this isn't quite possible with our current setup. The semantics of dL have no notion of hardware limitations, and so, if we left the semantics of old constructs unchanged, it wouldn't be clear how to define the new constructs.

Although leaving the semantics of old constructs entirely unmodified isn't feasible, we can still procure a semantics that very obviously coincides with the original semantics of dL. To do this, we use an interpretation I[3] that handles all information required of constructs particular to Controller Aware dL. This allows us to continue to use dL's notion of state without having to change its type of $\mathscr{V} \to \mathbb{R}$ (also referred to as $\mathscr{S}$). Then, since the subset of Controller Aware dL that is common to dL shouldn't depend on I, we can provide a semantics that mirrors the original semantics of dL.

Semantics of Terms Common to dL: The semantics of a term e in state $\omega \in \mathscr{S}$ is its value $\omega[\![e]\!]$ in $\mathbb{R}$. It is defined inductively as follows:

1. $\omega[\![x]\!] = \omega(x)$ for variable $x \in \mathscr{V}$
2. $\omega[\![c]\!] = c$
3. $\omega[\![e_1 + e_2]\!] = \omega[\![e_1]\!] + \omega[\![e_2]\!]$
4. $\omega[\![e_1 * e_2]\!] = \omega[\![e_1]\!] * \omega[\![e_2]\!]$

Semantics of Formulas: The semantics of a formula P in an interpretation I is the subset $I[\![P]\!] \subseteq \mathscr{S}$ of states in which P is true. It is defined inductively as follows:

1. $I[\![e_1 = e_2]\!] = \{\omega \mid \omega[\![e_1]\!] = \omega[\![e_2]\!]\}$
2. $I[\![e_1 \leq e_2]\!] = \{\omega \mid \omega[\![e_1]\!] \leq \omega[\![e_2]\!]\}$
3. $I[\![\neg P]\!] = (I[\![P]\!])^C = \mathscr{S} \setminus I[\![P]\!]$
4. $I[\![P \wedge Q]\!] = I[\![P]\!] \cap I[\![Q]\!]$
5. $I[\![P \vee Q]\!] = I[\![P]\!] \cup I[\![Q]\!]$
6. $I[\![P \to Q]\!] = (I[\![P]\!])^C \cup I[\![Q]\!]$
7. $I[\![\forall x P]\!] = \{\omega \mid \omega' \in I[\![P]\!]$ for all states ω' where ω' agrees with ω on all variables except $x\}$

[3] Using an interpretation in this way is inspired by (but distinct from) the approach used in "A Complete Uniform Substitution Calculus for Differential Dynamic Logic" [5] to define the semantics of function symbols.

8. $I[\![\exists x P]\!] = \{\omega \mid \omega' \in I[\![P]\!]$ for some state ω' where ω' agrees with ω on all variables except $x\}$
9. $I[\![\langle\alpha\rangle P]\!] = I[\![\alpha]\!] \circ I[\![P]\!] = \{\omega \mid \omega' \in I[\![P]\!]$ for some state ω' where $(\omega, \omega') \in I[\![\alpha]\!]\}$
10. $I[\![[\alpha]P]\!] = I[\![\neg\langle\alpha\rangle\neg P]\!] = \{\omega \mid \omega' \in I[\![P]\!]$ for all states ω' where $(\omega, \omega') \in I[\![\alpha]\!]\}$

Semantics of Hybrid Programs Common to dL: The semantics of a hybrid program α in an interpretation I is a binary transition relation $I[\![\alpha]\!] \subseteq \mathscr{S} \times \mathscr{S}$ on states. It is defined inductively as follows:

1. $I[\![x := e]\!] = \{(\omega, \omega') \mid \omega' = \omega$ except that $\omega'(x) = \omega[\![e]\!]\}$
2. $I[\![x := *]\!] = \{(\omega, \omega') \mid \omega' = \omega$ except that $\omega'(x)$ can have any real value$\}$
3. $I[\![?P]\!] = \{(\omega, \omega) \mid \omega \in I[\![P]\!]\}$
4. $I[\![x' = e \ \&P]\!] = \{(\omega, \omega') \mid \varphi(0) = \omega$ except at x' and $\varphi(r) = \omega'$ for a solution $\varphi : [0, r] \to \mathscr{S}$ of any duration r satisfying $I, \varphi \vDash x' = e \ \& \ P$[4]$\}$
5. $I[\![\alpha \cup \beta]\!] = I[\![\alpha]\!] \cup I[\![\beta]\!]$
6. $I[\![\alpha; \beta]\!] = I[\![\alpha]\!] \circ I[\![\beta]\!] = \{(\omega, \omega'') \mid (\omega, \omega') \in I[\![\alpha]\!], (\omega', \omega'') \in I[\![\beta]\!]$ for some $\omega' \in \mathscr{S}\}$
7. $I[\![\alpha^*]\!] = \bigcup_{n\in\mathbb{N}} I[\![\alpha^n]\!]$ with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?true$

2.3 Semantics of ~, Measure, and Set

In order to provide a formal semantics for Controller Aware dL's new constructs, we must first formalize our notion of the interpretation I. Let $\mathscr{V}_{\text{measure}} \subseteq \mathscr{V}$ be a set of variables designated as measurable, and let $\mathscr{V}_{\text{control}} \subseteq \mathscr{V}$ be a set of variables designated as controllable. Any variable $x \in \mathscr{V}$ may be measurable, controllable, both, or neither.

The interpretation I serves two purposes. First, when the programmer measures the value of a measurable variable via "measure x," I is needed to determine what value to assign to $\sim x$. In order to do this, I must be able to provide a function $I_{measure}$ that can take as input some measurable variable x, and return a nonempty set of values that $\sim x$ may be assigned to. More specifically, we require that $I_{measure}$ returns a range (a, b) in which $\sim x$ can be assigned any real value c such that $a \leq c \leq b$[5].

The previous discussion suggests that perhaps $I_{measure}$ should have the type $\mathscr{V}_{measure} \to \mathbb{R}^2$. However, a function of this type would be unable to provide different ranges that $\sim x$ may be assigned to for different values of x. Consequently, the program "measure x" wouldn't actually depend on the value of x itself, a highly counterintuitive and undesirable outcome.

To rectify this, we might attempt to give $I_{measure}$ the type $\mathscr{V}_{measure} \to \mathbb{R} \to \mathbb{R}^2$. This would allow $I_{measure}$ to depend not only on the variable being measured, but also on the real value of said variable. For some applications, this might be sufficient. Already, we would have the machinery necessary to cause "measure x" to assign any value to $\sim x$ that is within the range $(x-\varepsilon, x+\varepsilon)$ (for any real $\varepsilon \geq 0$).

Although this type is much better than the previous proposal, it is still problematically limiting to force the result of $I_{measure}$ to depend only on the variable being measured. To see why this is, consider the task of estimating the position of a moving object. If the object is moving slowly, or not

[4] $I, \varphi \vDash x' = e \ \& \ P$ holds if and only if, for all times $0 \leq z \leq r$: $\varphi(z) \in I[\![x' = e \wedge P]\!]$ with $\varphi(z)(x')$ defined as $(d\varphi(t)(x)/dt)(z)$ and $\varphi(z) = \varphi(0)$ except at x and x'

[5] Notably, we do not require that the current value of x be within the range returned by $I_{measure}$. This is to facilitate measurement functions that cannot produce values outside of a certain threshold. If a speedometer, for instance, is only designed to be capable of returning values between 0 and 120 mph, we want to be able to represent the fact that it will never state that a speeding car is going 130 mph, even if it actually is.

at all, it may be possible to estimate its position with a high degree of precision. However, the more quickly the object is moving, the harder it may be to estimate its position, to the point where an extremely fast-moving object may be impossible to see properly at all. This simple example illustrates how the precision with which we are able to measure a variable may easily depend on more than just the variable itself.

To accommodate $I_{measure}$ functions that depend on multiple variables, we could give $I_{measure}$ the type $\mathscr{V}_{measure} \to \overrightarrow{(\mathscr{V} \times \mathbb{R})} \to \mathbb{R}^2$. In this type, aside from the variable being measured, $I_{measure}$ would be given as input an arbitrary vector of variables and their corresponding values, and the range $I_{measure}$ returns would be allowed to depend on any or all of them. This is totally viable, and the only reason we don't take this approach is because it results in a slightly more cumbersome semantics.

Ultimately, we choose to give $I_{measure}$ the type $\mathscr{V}_{measure} \to \mathscr{S} \to \mathbb{R}^2$. This type provides maximal flexibility in defining an $I_{measure}$ function, as it allows the returned range to depend on anything in the state. However, without any additional restriction on what can and cannot be an interpretation, this typing affords too much flexibility, resulting in a variety of issues. Section 2.4 includes a discussion of said issues and the restrictions we impose on I to circumvent them.

Aside from providing $I_{measure}$, which will be used in the semantics of $\sim x$ and "measure x", the interpretation I is also needed to determine what value to assign to x in the hybrid program "set x e". For this, I must also be able to provide a function $I_{control}$ that can take as input some controllable variable x and some real value r, and return a nonempty set of values that x may be assigned to. As with $I_{measure}$, we require that $I_{control}$ returns a range (a, b) in which x can be assigned any real value c such that $a \leq c \leq b$[6].

Just as the precision of measurement phenomena can depend on a variety of factors aside from the variable being measured itself, the precision with which a controller is able to force a variable to obtain a particular value may also depend on various facts about the current state. Therefore, following the example of $I_{measure}$'s type, we give $I_{control}$ the type $\mathscr{V}_{control} \to \mathbb{R} \to \mathscr{S} \to \mathbb{R}^2$. Then $I_{control}\ x\ r\ \omega$ will return a range (a, b) in which x can be assigned any real value c such that $a \leq c \leq b$.

The two purposes an interpretation I serves is to provide a function $I_{measure}$ of type $\mathscr{V}_{measure} \to \mathscr{S} \to \mathbb{R}^2$ and a function $I_{control}$ of type $\mathscr{V}_{control} \to \mathbb{R} \to \mathscr{S} \to \mathbb{R}^2$. Since this is all that an interpretation needs to do, we can define an interpretation I as simply a pair of these two functions, giving I the type $(\mathscr{V}_{measure} \to \mathscr{S} \to \mathbb{R}^2) \times (\mathscr{V}_{control} \to \mathbb{R} \to \mathscr{S} \to \mathbb{R}^2)$. For simplicity, if I is an interpretation, we refer to the first element of its pair as $I_{measure}$ and the second element of the pair as $I_{control}$.

Having finally formalized our notion of an interpretation I, we can provide the semantics for each of the constructs unique to Controller Aware dL. For the semantics of $\sim x$, first recall that $\mathscr{V}$ denotes the set of all variables, and that states $\mathscr{S}$ are functions from $\mathscr{V}$ to $\mathbb{R}$. Then, let $\mathscr{V}^\sim$ define a set of approximation variables that each correspond to variables in $\mathscr{V}_{measure}$. If we have $\mathscr{V}^\sim \subseteq \mathscr{V}$, then each state $\omega \in \mathscr{S}$ also maps approximation variables to real values. In a minor abuse of notation, we refer to the approximation variable in $\mathscr{V}^\sim$ corresponding to an arbitrary variable $x \in \mathscr{V}_{measure}$ as $\sim x$. Then, we can define $\omega[\![\sim x]\!]$ simply as:

5. $\omega[\![\sim x]\!] = \omega(\sim x)$ for variable $\sim x \in \mathscr{V}^\sim \subseteq \mathscr{V}$

[6]Just as we do not require that $I_{measure}$ return a range containing the true value of the variable being measured, we do not require that $I_{control}$ return a range containing the real value that the controller is attempting to set the variable to. This is to facilitate control functions that cannot set variables to values outside of a certain threshold. If an engine, for instance, is only capable of accelerating a car to a certain speed, we want to represent the fact that it cannot make the car go any faster than that, even if a computational oversight causes the controller to attempt to make the car move faster than it can.

For the semantics of "measure x", we have now clarified that $I_{measure}\ x\ \omega$ returns a range (a, b) in which $\sim x$ should be nondeterministically assigned any value between a and b inclusive. Likewise, for the semantics of "set $x\ e$" we stated that $I_{control}\ x\ r\ \omega$ returns a range (a, b) in which x should be nondeterministically assigned any value between a and b inclusive. With this machinery, we can define the semantics of "measure x" and "set $x\ e$" as follows:

8. $I[\![\text{measure } x]\!] =$
   $\{(\omega, \omega') \mid \omega' = \omega \text{ except that } \omega'(\sim x) = c \text{ for any } c \text{ such that } a \leq c \leq b \text{ and } (a, b) = I_{measure}\ x\ \omega\}$
9. $I[\![\text{set } x\ e]\!] =$
   $\{(\omega, \omega') \mid \omega' = \omega \text{ except that } \omega'(x) = c \text{ for any } c \text{ such that } a \leq c \leq b \text{ and } (a, b) = I_{control}\ x\ (\omega[\![e]\!])\ \omega\}$

2.4 Restrictions on Interpretations

It was noted in the previous section that our current typing for I affords too much flexibility. When viewing Controller Aware dL as an abstract mathematical definition, flexibility isn't particularly problematic. But if we view Controller Aware dL as the potential foundation for a real programming language, excessive flexibility may create implementation issues. To give just one example, it's entirely unclear how one might write a function that depends on the entire state, as opposed to just a finite subset of it.

Additionally, in order for Controller Aware dL to be usable, it will require not just a mathematical semantics, but also an axiomatization that enables automated or interactive verification. If interpretations are allowed to be too general, then proving anything about Controller Aware dL formulas may become infeasible.

In order to ensure that Controller Aware dL is practically usable, we aim to give it the same proving power as ordinary dL. All dL formulas are also Controller Aware dL formulas, meaning Controller Aware dL trivially has the proving power of dL. So this consideration simply means that we wish to avoid allowing any of Controller Aware dL's constructs to enable proofs that could not be translated to ordinary dL.

If we do not place any restrictions on I, then it would be possible to embed all sorts of computations in the $I_{measure}$ and $I_{control}$ functions, and there would be no guarantee that these functions could be translated to dL. On the other hand, if we place too many restrictions on I, we may inadvertently prevent users from defining realistic hardware constraints, as we saw when we considered types for $I_{measure}$ that depended only on the variable being measured. Therefore, we seek a restriction that can be placed on interpretations that is as flexible as possible without undermining our ability to translate Controller Aware dL into dL.

One restriction we might consider is to, following the example from "A Complete Uniform Substitution Calculus for Differential Dynamic Logic" [5], enforce that $I_{measure}$ and $I_{control}$ are smooth functions (i.e. functions with derivatives of any order). This restriction may succeed in ensuring that Controller Aware dL does not exceed the proving power of dL, but it would also be too restrictive to define some reasonable hardware constraints.

To see why this constraint would be too restrictive, consider the case of a camera that is used to estimate the velocities of nearby cars. When a car is within the field of view of the camera, we may wish to model that the camera measures a relatively straightforward approximation of the car's true velocity. However, once a car exits the camera's field of view, no information about the car's velocity can be obtained. We might wish to express this by saying that if the car is within the camera's field of view, "measure v" sets $\sim v$ to some value between $v - \epsilon$ and $v + \epsilon$ inclusive (for some positive value of

ϵ), and otherwise, "measure v" sets $\sim v$ to the hard coded failure value of -1. This function is a totally reasonable approximation of the hardware constraints involved in using a camera to measure velocities, but is not smooth, and therefore would not be permitted by the previously described constraint.

Another restriction we might consider is to permit any range (a, b) that can be expressed via two Controller Aware dL terms. The idea would be that every $I_{measure}$ function would have to be defined via $I_{measure}\ x\ \omega = (\omega[\![e_1]\!], \omega[\![e_2]\!])$ for some Controller Aware dL terms e_1 and e_2 where $\omega[\![e_1]\!] \leq \omega[\![e_2]\!]$ for all $\omega \in \mathscr{S}$. This would have the benefit of facilitating a very straightforward translation from Controller Aware dL to dL, and it would also obviously ensure that the proving power of Controller Aware dL does not exceed dL. However, this restriction faces the same issue as the previous one. Since terms don't have access to any constructs analogous to the test operator of hybrid programs, there would be no means of defining the previously described camera constraint.

The restriction we ultimately impose on $I_{measure}$ and $I_{control}$ is that they each must be definable in terms of two dL hybrid programs[7] for each measurable or controllable variable. The intuition for this is that the programmer can write a hybrid program manipulating $\sim x$ (in the case of measure x), or x (in the case of set $x\ e$), and the final value of $\sim x$ (in the case of measure x), or x (in the case of set $x\ e$), will be the value that is used for a or b of the output range (a, b).

To give some concrete examples, if $I_{measure}\ x$ were defined via $\sim x := x - 0.5$ and $\sim x := x + 0.5$, then $I_{measure}\ x\ \omega$ would always return $(\omega[\![x - 0.5]\!], \omega[\![x + 0.5]\!])$ as its result. The previously described camera constraint could be defined via:

- $(?(\text{object in field of view}); \sim v := v - 0.5) \cup (?(\neg(\text{object in field of view})); \sim v := -1)$
- $(?(\text{object in field of view}); \sim v := v + 0.5) \cup (?(\neg(\text{object in field of view})); \sim v := -1)$

If $I_{measure}\ v$ were defined with the hybrid programs above, then $I_{measure}\ v\ \omega$ would return $(\omega[\![v - 0.5]\!], \omega[\![v + 0.5]\!])$ when the object is in the camera's field of view, and $(-1, -1)$ otherwise.

We formalize this intuition with the following restriction on I. For any $x \in \mathscr{V}$, $I_{measure}\ x$ can be defined by α and β, and $I_{control}\ x$ can be defined by α' and β' if and only if the following holds:

- α and β are ordinary dL hybrid programs.
- For every $\omega \in \mathscr{S}$, there is a minimum value in the set $\{c \mid \omega'(\sim x) = c \text{ where } (\omega, \omega') \in [\![\alpha]\!]^8\}$, and there is a maximum value in the set $\{c \mid \omega'(\sim x) = c \text{ where } (\omega, \omega') \in [\![\beta]\!]\}$. Furthermore, the minimum value in the former set is less than or equal to the maximum value in the latter set.
- α' and β' are dL hybrid programs that, in addition to being able to use variables from $\mathscr{V}$, can use a special symbol $\# \notin \mathscr{V}$ as a variable. In α' and β', $\#$ will serve the purpose of being able to refer to r, which is passed into $I_{control}$.
- For every $\omega \in \mathscr{S}$, there is a minimum value in the set $\{c \mid \omega'(x) = c \text{ where } (\omega, \omega') \in [\![\# := r; \alpha']\!]^9\}$, and there is a maximum value in the set $\{c \mid \omega'(x) = c \text{ where } (\omega, \omega') \in [\![\# := r; \beta']\!]\}$. Furthermore, the minimum value of the former set is less than or equal to the maximum value in the latter set.

[7] We require that they be definable in terms of dL hybrid programs, rather than Controller Aware dL hybrid programs, to avoid potential issues of self-referentiality. If, for instance, $I_{measure}\ x\ \omega$ were defined with a program that involves "measure x", then it would be unclear how to resolve the definition.

[8] $[\![\alpha]\!]$ is defined by the semantics of hybrid programs given in the *Logical Foundations for Cyber-Physical Systems* textbook [4].

[9] Technically, $(\# := r; \alpha')$ is an ill-formed hybrid program, as $\# \notin \mathscr{V}$. Therefore, we define $[\![\# := r; \alpha']\!]$ as the subset of $(\mathscr{V} \cup \{\#\})^2$ that would be obtained by dL's semantics if $\#$ were in $\mathscr{V}$.

Let α, β, α', and β' satisfy the previously given constraints. Then, if $I_{measure}\ x$ is defined by α and β, and $I_{control}\ x$ is defined by α' and β', we state that $I_{measure}\ x\ \omega = (a, b)$ and $I_{control}\ x\ r\ \omega = (a', b')$ if and only if the following holds:

- a is the minimum value in the set $\{c \mid \omega'(\sim x) = c \text{ where } (\omega, \omega') \in [\![\alpha]\!]\}$
- b is the maximum value in the set $\{c \mid \omega'(\sim x) = c \text{ where } (\omega, \omega') \in [\![\beta]\!]\}$
- a' is the minimum value in the set $\{c \mid \omega'(x) = c \text{ where } (\omega, \omega') \in [\![\# := r; \alpha']\!]\}$
- b' is the maximum value in the set $\{c \mid \omega'(x) = c \text{ where } (\omega, \omega') \in [\![\# := r; \beta']\!]\}$

3 Translation to dL

In order for Controller Aware dL to be practically usable as the foundation for a modelling language, it will require not just a mathematical semantics, but also something that enables automated or interactive verification. Traditionally, this role would be filled by an axiomatization and proof calculus. However, since dL already has a well-researched axiomatization and proof calculus [6], we can leverage this prior work by instead defining a semantics honoring translation from Controller Aware dL to dL. In effect, this would enable us to write formulas in Controller Aware dL, then prove properties about them with tools like KeYmaera X after transpiling them to dL. Although we do not implement a transpiler in this work, we provide the formal foundation necessary to do so in the future.

3.1 Freshening

We begin by defining some helper functions pertaining to variable management. For all $V \subseteq \mathscr{V}$ such that V is finite, let $\mathsf{freshen\_var}\ V$ be an injective mapping from $V \cup \{\#\}$ to $\mathscr{V} \setminus V$. The purpose of this function is to provide a means of deterministically mapping all free variables in an ordinary dL program, term, or formula in which # may appear, to unique variables that do not appear in said program, term or formula. With this, we define the mutually recursive $\mathsf{freshen\_term}$, $\mathsf{freshen\_formula}$, and $\mathsf{freshen\_program}$ below[10]:

Freshening terms: $\mathsf{freshen\_term}$ takes in a finite set $V \subseteq \mathscr{V}$ and ordinary dL term that may contain # in place of a variable, and outputs an ordinary dL term that does not contain any variables in V. It is defined inductively below:

1. $\mathsf{freshen\_term}\ V\ \# = \mathsf{freshen\_var}\ V\ \#$
2. $\mathsf{freshen\_term}\ V\ x = \mathsf{freshen\_var}\ V\ x$
3. $\mathsf{freshen\_term}\ V\ c = c$
4. $\mathsf{freshen\_term}\ V\ (e_1 + e_2) = (\mathsf{freshen\_term}\ V\ e_1) + (\mathsf{freshen\_term}\ V\ e_2)$
5. $\mathsf{freshen\_term}\ V\ (e_1 * e_2) = (\mathsf{freshen\_term}\ V\ e_1) * (\mathsf{freshen\_term}\ V\ e_2)$

[10]Technically our definitions for $\mathsf{freshen\_term}$, $\mathsf{freshen\_formula}$, and $\mathsf{freshen\_program}$ are incomplete. This is because our definitions assume that all variables that appear as input for $\mathsf{freshen\_term}$ are elements of V. However, this incompleteness is not a problem for our purposes, because we only refer to these functions in contexts where we have assumed that all variables that can appear in the input are elements of V.

Freshening formulas: freshen_formula takes in a finite set $V \subseteq \mathscr{V}$ and ordinary dL formula that may contain # in place of a variable, and outputs an ordinary dL formula that does not contain any variables in V. It is defined inductively below:

1. freshen_formula V $(e_1 = e_2) =$ (freshen_term V e_1) $=$ (freshen_term V e_2)
2. freshen_formula V $(e_1 \leq e_2) =$ (freshen_term V e_1) $\leq$ (freshen_term V e_2)
3. freshen_formula V $(\neg P) = \neg$(freshen_formula V P)
4. freshen_formula V $(P \wedge Q) =$ (freshen_formula V P) $\wedge$ (freshen_formula V Q)
5. freshen_formula V $(P \vee Q) =$ (freshen_formula V P) $\vee$ (freshen_formula V Q)
6. freshen_formula V $(P \rightarrow Q) =$ (freshen_formula V P) $\rightarrow$ (freshen_formula V Q)
7. freshen_formula V $(\forall x P) = \forall$(freshen_var V x)(freshen_formula V P)
8. freshen_formula V $(\exists x P) = \exists$(freshen_var V x)(freshen_formula V P)
9. freshen_formula V $(\langle\alpha\rangle P) = \langle$freshen_program V $\alpha\rangle$(freshen_formula V P)
10. freshen_formula V $([\alpha]P) = [$freshen_program V $\alpha]$(freshen_formula V P)

Freshening hybrid programs: freshen_program takes in a finite set $V \subseteq \mathscr{V}$ and ordinary dL hybrid program that may contain # in place of a variable, and outputs an ordinary dL hybrid program that does not contain any variables in V. It is defined inductively below:

1. freshen_program V $(x := e) =$ (freshen_var V x) $:=$ (freshen_term V e)
2. freshen_program V $(x := *) =$ (freshen_var V x) $:= *$
3. freshen_program V $(?P) = ?$(freshen_formula V P)
4. freshen_program V $(x' = e \,\&\, P) =$ (freshen_var V x)$'$ $=$ (freshen_term V e) & (freshen_formula V P)
5. freshen_program V $(\alpha \cup \beta) =$ (freshen_program V α) $\cup$ (freshen_program V β)
6. freshen_program V $(\alpha; \beta) =$ (freshen_program V α); (freshen_program V β)
7. freshen_program V $(\alpha^*) =$ (freshen_program V α)*

There are three important properties to note about the definitions given above. First, a freshened program will not alter any of the variables in the input set V. Second, if a program α would alter some variable x in the input set V, then the freshened α will instead alter freshen_var V x in an analogous way. And third, if a freshened α alters some variable v such that $v =$ freshen_var V x for some $x \in V$, then since freshen_var V is injective, α must be able to alter x in an analogous way. We formalize these properties in the following lemmas.

Lemma 3.1. *If V contains all of the variables that appear*[11] *in a dL program α, and if ω and ω' are states such that $(\omega, \omega') \in [\![$freshen_program V $\alpha]\!]$, then ω and ω' must agree on all variables in V.*

This follows from the fact that no variable in V may appear in (freshen_program V α), a fact that follows immediately from a straightforward induction on α.

[11] By "all of the variables that appear in a program α", we refer to the definition of $V(\alpha)$ given in section 5.6.6 of *Logical Foundations for Cyber-Physical Systems*[4].

Lemma 3.2. *Let $V \subseteq \mathscr{V}$ contain all of the variables that appear in a dL program α, and let ω and ω' be states such that $(\omega, \omega') \in [\![\alpha]\!]$. Then suppose that for each x that appears in α, $\omega(x) = \nu(\mathsf{freshen\_var}\ V\ x)$, and furthermore, if $\#$ appears in α, $\omega(\#) = \nu(\mathsf{freshen\_var}\ V\ \#)$. It follows that there exists a ν' such that $(\nu, \nu') \in [\![\mathsf{freshen\_program}\ V\ \alpha]\!]$ and for each x that appears in α, $\omega'(x) = \nu'(\mathsf{freshen\_var}\ V\ x)$. Additionally, if $\#$ appears in α, then it must also be the case that $\omega'(\#) = \nu'(\mathsf{freshen\_var}\ V\ \#)$.*

Although the statement of this lemma is somewhat complicated, the proof itself follows straightforwardly by induction on α.

Lemma 3.3. *Let $V \subseteq \mathscr{V}$ contain all of the variables that appear in a dL program α, and let ν and ν' be states such that $(\nu, \nu') \in [\![\mathsf{freshen\_program}\ \mathsf{V}\ \alpha]\!]$. Then, suppose that for each x that appears in α, $\nu(\mathsf{freshen\_var}\ V\ x) = \omega(x)$, and furthermore, if $\#$ appears in α, $\nu(\mathsf{freshen\_var}\ V\ \#) = \omega(\#)$. It follows that there exists an ω' such that $(\omega, \omega') \in I[\![\alpha]\!]$ and for each x that appears in α, $\nu'(\mathsf{freshen\_var}\ V\ x) = \omega'(x)$. Additionally, if $\#$ appears in α, then it must also be the case that $\nu'(\mathsf{freshen\_var}\ V\ \#) = \omega'(\#)$.*

Although the statement of this lemma is somewhat complicated, the proof itself follows straightforwardly by induction on ($\mathsf{freshen\_program}\ V\ \alpha$). The only potentially nonobvious fact is that because $\mathsf{freshen\_var}\ V$ is injective, if $\mathsf{freshen\_program}\ V\ \alpha$ takes a form that can alter or read $\mathsf{freshen\_var}\ V\ x$, then α must be able to alter or read x in an analogous way. As an example, if ($\mathsf{freshen\_program}\ V\ \alpha$) is $y := *$, then there must be a unique $x \in V$ such that $y = \mathsf{freshen\_var}\ V\ x$. From this and the definition of $\mathsf{freshen\_program}$, α must equal $x := *$, and the statement of Lemma 3.3 immediately follows.

3.2 Translation

Unlike the truth of dL formulas, which depends only on state, the truth of Controller Aware dL formulas depends on both the state ω and the interpretation I that the formula is evaluated in. Because of this, it is impossible to write function that, given a Controller Aware dL formula P, outputs a dL formula Q such that for all interpretations I and all states ω, $\omega \in I[\![P]\!]$ iff $\omega \in [\![Q]\!]$.

To see this, consider the formula $P = [\text{measure } x]x \geq 0$. If the interpretation I is such that $I_{measure}\ x\ \omega$ returns a positive range for all ω, then this formula is valid in I[12]. However, if the interpretation I is such that $I_{measure}\ x\ \omega$ can return a range that includes negative numbers, then this formula is not valid in I. So if we attempted to translate $[\text{measure } x]x \geq 0$ into dL, if we produced a valid formula, we would not have a sound translation with respect to the interpretations that can return negative ranges, and if we produced an invalid formula, we would not have a sound translation with respect to the interpretations that return strictly positive ranges.

Although it is impossible to write a translation from Controller Aware dL to dL that respects all interpretations, there can still be a meaningful translation from Controller Aware dL to dL. If we fix an interpretation I, which is analogous to fixing particular hardware constraints, then we can write a function that takes in a Controller Aware dL formula P and outputs a dL formula Q such that $I[\![P]\!] = [\![Q]\!]$. In order to do this, we need to translate not only formulas, but also terms and hybrid programs. Translations from Controller Aware dL to dL for each of these is included below:

[12] A Controller Aware dL formula P is valid in an interpretation I iff $I[\![P]\!] = \mathscr{S}$.

Translation of Terms: translate_term takes in a Controller Aware dL term and outputs a dL term. It is intended that for all Controller Aware dL terms e and all states ω, $\omega[\![e]\!]$ in Controller Aware dL semantics is equal to $\omega[\![\text{translate\_term } e]\!]$ in dL semantics. translate_term is defined inductively below:

1. translate_term $x = x$
2. translate_term $c = c$
3. translate_term $(e_1 + e_2) =$ (translate_term e_1) + (translate_term e_2)
4. translate_term $(e_1 * e_2) =$ (translate_term e_1) $*$ (translate_term e_2)
5. translate_term $\sim x = \sim x$[13]

Translation of Formulas: translate_formula takes in an interpretation I and a Controller Aware dL formula and outputs a dL formula. It is intended that for all Controller Aware dL formulas P, $I[\![P]\!] = [\![\text{translate\_formula } I\ P]\!]$. translate_formula is defined inductively below:

1. translate_formula I $(e_1 = e_2) =$ (translate_term e_1) = (translate_term e_2)
2. translate_formula I $(e_1 \leq e_2) =$ (translate_term e_1) $\leq$ (translate_term e_2)
3. translate_formula I $(\neg P) = \neg$(translate_formula I P)
4. translate_formula I $(P \wedge Q) =$ (translate_formula I P) $\wedge$ (translate_formula I Q)
5. translate_formula I $(P \vee Q) =$ (translate_formula I P) $\vee$ (translate_formula I Q)
6. translate_formula I $(P \rightarrow Q) =$ (translate_formula I P) $\rightarrow$ (translate_formula I Q)
7. translate_formula I $(\forall x P) = \forall x$(translate_formula I P)
8. translate_formula I $(\exists x P) = \exists x$(translate_formula I P)
9. translate_formula I $(\langle\alpha\rangle P) =$ $\langle$translate_program I $V(\langle\alpha\rangle P)$ $\alpha\rangle$(translate_formula I P) where $V(\langle\alpha\rangle P)$ is the set of all variables that appear[14] in $\langle\alpha\rangle P$
10. translate_formula I $([\alpha]P) =$ [translate_program I $V([\alpha]P)$ α](translate_formula I P) where $V([\alpha]P)$ is the set of all variables that appear in $[\alpha]P$

Translation of Hybrid Programs: translate_program takes in an interpretation I, a finite set of variables $V \subseteq \mathscr{V}$, and a Controller Aware dL hybrid program, and outputs a dL hybrid program. It is intended that if V contains all variables that appear in α, then the following holds:

- If $(\omega, \omega') \in I[\![\alpha]\!]$ and ν agrees with ω on all variables in V, then there must exist a ν' such that $(\nu, \nu') \in [\![\text{translate\_program } I\ V\ \alpha]\!]$ and ν' agrees with ω' on all variables in V.
- If $(\nu, \nu') \in [\![\text{translate\_program } I\ V\ \alpha]\!]$ and ω agrees with ν on all variables in V, then there must exist an ω' such that $(\omega, \omega') \in I[\![\alpha]\!]$ and ω' agrees with ν' on all variables in V.

[13] The $\sim x$ that appears on the left is the syntactic construct ~ applied to the variable x, while the $\sim x$ that appears on the right is the approximation variable in $\mathscr{V}^{\sim}$ that corresponds to x.

[14] To formally define "the set of all variables that appear in $\langle\alpha\rangle P$", we extend the definition of $V(\langle\alpha\rangle P)$ given in section 5.6.6 of *Logical Foundations for Cyber-Physical Systems*[4]. For syntactic constructs unique to Controller Aware dL, we define the set of variables that appear as follows: The set of variables that appear in $\sim x$ is $\{x, \sim x\}$, where $\sim x$ is the approximation variable in $\mathscr{V}^{\sim}$ that corresponds to x. The set of variables that appear in "measure x" is $V(\alpha) \cup V(\beta) \cup \{x, \sim x\}$ where α and β are the dL programs that define $I_{measure}\ x$. Finally, the set of variables that appear in "set x e" is $V(\alpha) \cup V(\beta) \cup V(e) \cup \{x\}$ where α and β are the programs that define $I_{control}\ x$ Note that, in this context, # does not count as a variable, and therefore, does not appear in $V(\alpha)$ or $V(\beta)$.

translate_program is defined inductively below:

1. translate_program $I\ V\ (x := e) = (x :=$ translate_term $e)$
2. translate_program $I\ V\ (x := *) = (x := *)$
3. translate_program $I\ V\ (?P) = ?($translate_formula $I\ P)$
4. translate_program $I\ V\ (x' = e\ \&\ P) = (x' =$ (translate_term e) & (translate_formula $I\ P$))
5. translate_program $I\ V\ (\alpha \cup \beta) =$ (translate_program $I\ V\ \alpha) \cup$ (translate_program $I\ V\ \beta$)
6. translate_program $I\ V\ (\alpha; \beta) =$ (translate_program $I\ V\ \alpha$); (translate_program $I\ V\ \beta$)
7. translate_program $I\ V\ (\alpha^*) =$ (translate_program $I\ V\ \alpha)^*$
8. translate_program $I\ V$ (measure x) =
   γ; freshen_program $(V \cup \{y, z\})\ \alpha$; $y :=$ freshen_var $(V \cup \{y, z\})\ {\sim}x$;
   γ; freshen_program $(V \cup \{y, z\})\ \beta$; $z :=$ freshen_var $(V \cup \{y, z\})\ {\sim}x$;
   ${\sim}x := *$; $?(y \leq {\sim}x \wedge {\sim}x \leq z)$ with the following definitions:
   - γ is the hybrid program that, for each $x \in V$, assigns freshen_var $(V \cup \{y, z\})\ x$ the value of x via a sequential composition of hybrid programs of the form:
     freshen_var $(V \cup \{y, z\})\ x := x$. The fact that V is finite guarantees that γ is definable.
   - α and β are the dL hybrid programs that define $I_{measure}\ x$.
   - y and z are distinct fresh variables that do not appear in V.
9. translate_program $I\ V$ (set $x\ e$) =
   γ; (freshen_var $(V \cup \{y, z\})\ \#) :=$ translate_term e; freshen_program $(V \cup \{y, z\})\ \alpha$;
   $y :=$ freshen_var $(V \cup \{y, z\})\ x$;
   γ; (freshen_var $(V \cup \{y, z\})\ \#) :=$ translate_term e; freshen_program $(V \cup \{y, z\})\ \beta$;
   $z :=$ freshen_var $(V \cup \{y, z\})\ x$;
   $x := *; ?(y \leq x \wedge x \leq z)$ with the same definitions for γ, α, β, y, and z as above (except that α and β are the programs that define $I_{control}\ x$ rather than $I_{measure}\ x$).

3.3 Soundness

Although we now have a translation from Controller Aware dL to dL, such a translation is worthless if it does not honor the semantics of both logics. In particular, in order to use our translation from Controller Aware dL to dL as a means of verifying Controller Aware dL formulas, it is essential that a formula is valid in Controller Aware dL if and only if its translated formula is valid in dL. Towards this end, we prove the following three theorems.

Theorem 3.4. *For all Controller Aware dL terms e and all states $\omega \in \mathscr{S}$, $\omega[\![e]\!]$ in Controller Aware dL semantics is equal to $\omega[\![$*translate_term $e]\!]$ in dL semantics.*

Theorem 3.5. *For all interpretations I and Controller Aware dL formulas P, $I[\![P]\!] = [\![$*translate_formula $I\ P]\!]$.*

Theorem 3.6. *For all interpretations I, finite sets of variables $V \subseteq \mathscr{V}$, and Controller Aware dL hybrid programs α, if V contains all variables that appear in α, then the following holds:*

- *If $(\omega, \omega') \in I[\![\alpha]\!]$ and ν agrees with ω on all variables in V, then there must exist a ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \alpha]\!]$ and ν' agrees with ω' on all variables in V.*
- *If $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \alpha]\!]$ and ω agrees with ν on all variables in V, then there must exist an ω' such that $(\omega, \omega') \in I[\![\alpha]\!]$ and ω' agrees with ν' on all variables in V.*

Each of these theorems is proven in our appendix by simultaneous induction on the term, formula, or hybrid program that is being translated. Note that Theorem 3.5 directly entails that for all interpretations I and Controller Aware dL formulas P, P is valid in the interpretation I if and only if translate_formula I P is valid in dL. This means that Controller Aware dL formulas can be verified by translating them with translate_formula and verifying the result with tools like KeYmaera X.

4 Related Work and Future Work

4.1 Related Work

There are numerous works in the literature that are adjacent to my project. "Differential Dynamic Logic for Hybrid Systems" [6] and "Differential Game Logic," [7] provide formal definitions for the syntax and semantics of dL and dGL respectively. But unlike my project, which focuses particularly on creating an extension to aid usability, these papers emphasize proving soundness and completeness results for their respective logics. "The KeYmaera X Proof IDE: Concepts on Usability in Hybrid Systems Theorem Proving" [3] shared my project's emphasis on usability, but focused on how the KeYmaera X IDE can aid verification. In particular, the paper appears to focus on how an IDE can support users that intend to write and verify dL formulas, rather than on how dL itself might be augmented for usability purposes.

"ModelPlex: verified runtime validation of verified cyber-physical system models" [2] is a paper that addresses fidelity concerns between CPS models and implementations. But in my understanding, it addresses a different set of fidelity concerns, particularly ensuring that a CPS's implementation does what the model says it should so that verification results obtained for the model apply to the actual system. "Dynamic doxastic differential dynamic logic for belief-aware cyber-physical systems" [1] is perhaps the most similar work to my own, in terms of the issue being targeted, but it takes the very different approach of adding belief modalities to dL for the sake of reasoning about what a controller might believe about the world around it.

4.2 Future Work

The most obvious opportunity for extending this work lies in implementing a transpiler from Controller Aware dL to dL. Controller Aware dL is designed to increase usability and decrease the conceptual overhead of modeling and honoring hardware constraints. We believe that by isolating the definitions of hardware constraints to the definitions of interpretations, and injecting said hardware constraints into the main body of code only through simple syntactic constructs, Controller Aware dL makes it easier to define and enforce hardware constraints correctly. However, until a transpiler for Controller Aware dL is actually implemented, such beliefs can only be speculative. To properly establish the benefit of Controller Aware dL over dL itself, user studies using a transpiler from Controller Aware dL to dL to enable the verification of Controller Aware dL programs would be extremely useful.

References

[1] João G. Martins, André Platzer, and João Leite. "Dynamic Doxastic Differential Dynamic Logic for Belief-Aware Cyber-Physical Systems". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 428–445. DOI: 10.1007/978-3-030-29026-9_24. URL: https://doi.org/10.1007/978-3-030-29026-9_24.

[2] Stefan Mitsch and André Platzer. "ModelPlex: verified runtime validation of verified cyber-physical system models". In: *Formal Methods in System Design* 49.1-2 (Feb. 2016), pp. 33–74. DOI: 10.1007/s10703-016-0241-z. URL: https://doi.org/10.1007/s10703-016-0241-z.

[3] Stefan Mitsch and André Platzer. "The KeYmaera X Proof IDE - Concepts on Usability in Hybrid Systems Theorem Proving". In: 240 (Jan. 2017), pp. 67–81. DOI: 10.4204/eptcs.240.5. URL: https://doi.org/10.4204/eptcs.240.5.

[4] Andr Platzer. *Logical Foundations of Cyber-Physical Systems*. 1st. Springer Publishing Company, Incorporated, 2018. ISBN: 3319635875.

[5] André Platzer. "A Complete Uniform Substitution Calculus for Differential Dynamic Logic". In: *Journal of Automated Reasoning* 59.2 (Aug. 2016), pp. 219–265. DOI: 10.1007/s10817-016-9385-1. URL: https://doi.org/10.1007/s10817-016-9385-1.

[6] André Platzer. "Differential Dynamic Logic for Hybrid Systems". In: 41.2 (Aug. 2008), pp. 143–189. DOI: 10.1007/s10817-008-9103-8. URL: https://doi.org/10.1007/s10817-008-9103-8.

[7] André Platzer. "Differential Game Logic". In: 17.1 (Dec. 2015), pp. 1–51. DOI: 10.1145/2817824. URL: https://doi.org/10.1145/2817824.

A Appendix

Here, we include the full proof of Theorems 3.4, 3.5, and 3.6.

Theorem 3.4. *For all Controller Aware dL terms e and all states $\omega \in \mathscr{S}$, $\omega[\![e]\!]$ in Controller Aware dL semantics is equal to $\omega[\![\mathsf{translate\_term}\ e]\!]$ in dL semantics.*

Theorem 3.5. *For all interpretations I and Controller Aware dL formulas P, $I[\![P]\!] = [\![\mathsf{translate\_formula}\ I\ P]\!]$.*

Theorem 3.6. *For all interpretations I, finite sets of variables $V \subseteq \mathscr{V}$, and Controller Aware dL hybrid programs α, if V contains all variables that appear in α, then the following holds:*

- *If $(\omega, \omega') \in I[\![\alpha]\!]$ and ν agrees with ω on all variables in V, then there must exist a ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \alpha]\!]$ and ν' agrees with ω' on all variables in V.*
- *If $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \alpha]\!]$ and ω agrees with ν on all variables in V, then there must exist an ω' such that $(\omega, \omega') \in I[\![\alpha]\!]$ and ω' agrees with ν' on all variables in V.*

Each of these theorems is proven by simultaneous induction on the term, formula, or hybrid program that is being translated. All of the cases are included below:

Term Cases:

Case $e = x$: $\mathsf{translate\_term}\ x = x$, and $\omega[\![x]\!]$ is defined as $\omega(x)$ in both Controller Aware dL semantics and dL semantics.

Case $e = c$: $\mathsf{translate\_term}\ c = c$, and $\omega[\![c]\!]$ is defined as c in both Controller Aware dL semantics and dL semantics.

Case $e = e_1 + e_2$: $\mathsf{translate\_term}\ (e_1 + e_2) = \mathsf{translate\_term}\ e_1 + \mathsf{translate\_term}\ e_2$. By the inductive hypothesis, $\omega[\![e_1]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ e_1]\!]$ in dL semantics, and likewise, $\omega[\![e_2]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ e_2]\!]$ in dL semantics. Since $\omega[\![e_1 + e_2]\!] = \omega[\![e_1]\!] + \omega[\![e_2]\!]$ in Controller Aware dL semantics, this implies that $\omega[\![e_1 + e_2]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ e_1]\!] + \omega[\![\mathsf{translate\_term}\ e_2]\!]$ in dL semantics. Since $\omega[\![\mathsf{translate\_term}\ e_1]\!] + \omega[\![\mathsf{translate\_term}\ e_2]\!] = \omega[\![(\mathsf{translate\_term}\ e_1) + (\mathsf{translate\_term}\ e_2)]\!]$ in dL semantics, and since $(\mathsf{translate\_term}\ e_1) + (\mathsf{translate\_term}\ e_2) = \mathsf{translate\_term}\ (e_1 + e_2)$ by the definition of $\mathsf{translate\_term}$, this implies that $\omega[\![e_1 + e_2]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ (e_1 + e_2)]\!]$ in dL semantics, as desired.

Case $e = e_1 * e_2$: This case proceeds identically to the previous case with $+$ replaced by $*$.

Case $e = \sim x$: $\mathsf{translate\_term}\ \sim x = \sim x$ where the $\sim x$ on the left is the syntactic construct $\sim$ and the variable x, and the $\sim x$ on the right is the approximation variable in $\mathscr{V}^{\sim}$ associated with x. By the semantics of Controller Aware dL, $\omega[\![\sim x]\!]$ (where $\sim x$ is the syntactic construct) $= \omega(\sim x)$ (where $\sim x$ is the approximation variable), and by the semantics of dL, $\omega[\![\sim x]\!]$ (where $\sim x$ is the approximation variable) $= \omega(\sim x)$. Therefore, $\omega[\![\sim x]\!]$ resolves to the same value in both Controller Aware dL and ordinary dL, as desired.

Formula Cases:

Case $P = (e_1 = e_2)$: $\mathsf{translate\_formula}\ I\ (e_1 = e_2) = (\mathsf{translate\_term}\ e_1) = (\mathsf{translate\_term}\ e_2)$. Let $\omega \in \mathscr{S}$ be an arbitrary state. By the inductive hypothesis, $\omega[\![e_1]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ e_1]\!]$ in dL semantics, and likewise, $\omega[\![e_2]\!]$ in Controller Aware dL semantics equals $\omega[\![\mathsf{translate\_term}\ e_2]\!]$ in dL semantics. Therefore, $\omega[\![e_1]\!] = \omega[\![e_2]\!]$ in Controller Aware dL semantics if and only if $\omega[\![\mathsf{translate\_term}\ e_1]\!] = \omega[\![\mathsf{translate\_term}\ e_2]\!]$ in dL semantics. This means that $\omega \in I[\![e_1 = e_2]\!]$ if and only if $\omega \in [\![(\mathsf{translate\_term}\ e_1) = (\mathsf{translate\_term}\ e_2)]\!]$, as desired.

Case $P = (e_1 \leq e_2)$**:** This case proceeds identically to the previous case with $=$ replaced by $\leq$.

Case $P = \neg Q$**:** translate_formula I $(\neg Q) = \neg$(translate_formula I Q). By the inductive hypothesis, $I[\![Q]\!] = [\![$translate_formula I $Q]\!]$. Therefore, $(I[\![Q]\!])^C = [\![$translate_formula I $Q]\!]^C$, as desired.

Case $P = Q \wedge R$**:** translate_formula I $(Q \wedge R) =$ (translate_formula I Q) $\wedge$ (translate_formula I R). By the inductive hypothesis, $I[\![Q]\!] = [\![$translate_formula I $Q]\!]$ and $I[\![R]\!] = [\![$translate_formula I $R]\!]$. Therefore, $I[\![Q]\!] \cap I[\![R]\!] = [\![$translate_formula I $Q]\!] \cap [\![$translate_formula I $R]\!]$, as desired.

Case $P = Q \vee R$**:** This case proceeds identically to the previous case $\wedge$ replaced by $\vee$ and $\cap$ replaced by $\cup$.

Case $P = Q \rightarrow R$**:** This case directly mirrors the two previous cases.

Case $P = \forall x Q$**:** translate_formula I $(\forall x Q) = \forall x$(translate_formula I Q). Let $\omega \in \mathscr{S}$ be an arbitrary state. For all $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x, the inductive hypothesis yields that $\omega' \in I[\![Q]\!]$ if and only if $\omega' \in [\![$translate_formula I $Q]\!]$. This entails that all $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x are in $I[\![Q]\!]$ if and only if all $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x are in $[\![$translate_formula I $Q]\!]$. By the semantics of universal quantification in Controller Aware dL and dL, this means $\omega \in I[\![\forall x Q]\!]$ if and only if $\omega \in [\![\forall x($translate_formula I $Q)]\!]$, as desired.

Case $P = \exists x Q$**:** translate_formula I $(\exists x Q) = \exists x$(translate_formula I Q). Let $\omega \in \mathscr{S}$ be an arbitrary state. For all $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x, the inductive hypothesis yields that $\omega' \in I[\![Q]\!]$ if and only if $\omega' \in [\![$translate_formula I $Q]\!]$. This entails that some $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x is in $I[\![Q]\!]$ if and only if some $\omega' \in \mathscr{S}$ such that $\omega' = \omega$ except at x is in $[\![$translate_formula I $Q]\!]$. By the semantics of existential quantification in Controller Aware dL and dL, this means $\omega \in I[\![\exists x Q]\!]$ if and only if $\omega \in [\![\exists x($translate_formula I $Q)]\!]$, as desired.

Case $P = \langle\alpha\rangle Q$**:** translate_formula I $(\langle\alpha\rangle Q) = \langle$translate_program I $V(\langle\alpha\rangle Q)$ $\alpha\rangle$(translate_formula I Q). Let $\omega \in \mathscr{S}$ be an arbitrary state. Breaking form from the other proofs thus far, we prove that $I[\![\langle\alpha\rangle Q]\!] = [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$ by separately showing that $\omega \in I[\![\langle\alpha\rangle Q]\!]$ implies $\omega \in [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$ and $\omega \in [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$ implies $\omega \in I[\![\langle\alpha\rangle Q]\!]$.

- To show that $\omega \in I[\![\langle\alpha\rangle Q]\!]$ implies $\omega \in [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$, assume $\omega \in I[\![\langle\alpha\rangle]\!]$. By the semantics of the diamond operator in Controller Aware dL, this means that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ and $\omega' \in I[\![Q]\!]$. Note that $V(\langle\alpha\rangle Q)$ contains all variables that appear in α. Then, by the inductive hypothesis (the first bullet of Theorem 3.6), we have that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![$translate_program I $V(\langle\alpha\rangle Q)$ $\alpha]\!]$ and for all $x \in V(\langle\alpha\rangle Q)$, $\omega'(x) = \omega''(x)$.

  Note that since $V(\langle\alpha\rangle Q)$ contains all variables that appear in Q, and since ω' and ω'' agree on all variables that appear in $V(\langle\alpha\rangle Q)$, ω' and ω'' must agree on all variables that appear in Q. By a straightforward induction on Q, this means that $\omega' \in I[\![Q]\!]$ if and only if $\omega'' \in I[\![Q]\!]$. Finally, by the inductive hypothesis, $\omega'' \in I[\![Q]\!]$ if and only if $\omega'' \in [\![$translate_formula I $Q]\!]$. Since $\omega' \in I[\![Q]\!]$, this entails that $\omega'' \in [\![$translate_formula I $Q]\!]$.

  Thus, we have shown that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![$translate_program I $V(\langle\alpha\rangle Q)$ $\alpha]\!]$ and $\omega'' \in [\![$translate_formula I $Q]\!]$. By the semantics of the diamond operator in dL, this means that $\omega \in [\![\langle$translate_program I $V(\langle\alpha\rangle Q)$ $\alpha\rangle$(translate_formula I $Q)]\!]$, as desired.

- To show that $\omega \in [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$ implies $\omega \in I[\![\langle\alpha\rangle Q]\!]$, assume $\omega \in [\![$translate_formula I $(\langle\alpha\rangle Q)]\!]$. By the semantics of the diamond operator in dL, this means that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![$translate_program I $V(\langle\alpha\rangle Q)$ $\alpha]\!]$ and $\omega'' \in [\![$translate_formula I $Q]\!]$. Note that $V(\langle\alpha\rangle Q)$ contains all variables that appear in α. Then, by the inductive hypothesis (the second

bullet of Theorem 3.6), we have that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ and for all $x \in V(\langle\alpha\rangle Q)$, $\omega'(x) = \omega''(x)$.

Note that since $V(\langle\alpha\rangle Q)$ contains all variables that appear in Q, and since ω' and ω'' agree on all variables that appear in $V(\langle\alpha\rangle Q)$, ω' and ω'' must agree on all variables that appear in Q. By a straightforward induction on Q, this means that $\omega' \in I[\![Q]\!]$ if and only if $\omega'' \in I[\![Q]\!]$. Finally, by the inductive hypothesis, $\omega'' \in I[\![Q]\!]$ if and only if $\omega'' \in [\![\mathsf{translate\_formula}\ I\ Q]\!]$. Since $\omega'' \in [\![\mathsf{translate\_formula}\ I\ Q]\!]$, this entails that $\omega' \in I[\![Q]\!]$.

Thus, we have shown that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ and $\omega' \in I[\![Q]\!]$. By the semantics of the diamond operator in Controller Aware dL, this means that $\omega \in I[\![\langle\alpha\rangle Q]\!]$, as desired.

Case $P = [\alpha]Q$**:** $\mathsf{translate\_formula}\ I\ ([\alpha]Q) = [\mathsf{translate\_program}\ I\ V([\alpha]Q)\ \alpha](\mathsf{translate\_formula}\ I\ Q)$. Let $\omega \in \mathscr{S}$ be an arbitrary state. We prove that $I[\![[\alpha]Q]\!] = [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$ by separately showing that $\omega \notin I[\![[\alpha]Q]\!]$ implies $\omega \notin [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$ and $\omega \notin [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$ implies $\omega \notin I[\![[\alpha]Q]\!]$.

- To show that $\omega \notin I[\![[\alpha]Q]\!]$ implies $\omega \notin [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$, assume $\omega \notin I[\![[\alpha]Q]\!]$. By the semantics of the box operator in Controller Aware dL, this means that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ but $\omega' \notin I[\![Q]\!]$. Note that $V([\alpha]Q)$ contains all variables that appear in α. Then, by the inductive hypothesis (the first bullet of Theorem 3.6), we have that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![\mathsf{translate\_program}\ I\ V([\alpha]Q)\ \alpha]\!]$ and for all $x \in V([\alpha]Q)$, $\omega'(x) = \omega''(x)$.

  Note that since $V([\alpha]Q)$ contains all variables that appear in Q, and ω' and ω'' agree on all variables that appear in $V([\alpha]Q)$, ω' and ω'' must agree on all variables that appear in Q. By a straightforward induction on Q, this means that $\omega' \in I[\![Q]\!]$ if and only if $\omega'' \in I[\![Q]\!]$. Finally, by the inductive hypothesis, $\omega'' \in I[\![Q]\!]$ if and only if $\omega'' \in [\![\mathsf{translate\_program}\ I\ Q]\!]$. Since $\omega' \notin I[\![Q]\!]$, this entails that $\omega'' \notin [\![\mathsf{translate\_program}\ I\ Q]\!]$.

  Thus, we have shown that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![\mathsf{translate\_program}\ I\ V([\alpha]Q)\ \alpha]\!]$ but $\omega'' \notin [\![\mathsf{translate\_formula}\ I\ Q]\!]$. By the semantics of the box operator in dL, this means that $\omega \notin [\mathsf{translate\_program}\ I\ V([\alpha]Q)\ \alpha](\mathsf{translate\_formula}\ I\ Q)$, as desired.

- To show that $\omega \notin [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$ implies $\omega \notin I[\![[\alpha]Q]\!]$, assume $\omega \notin [\![\mathsf{translate\_formula}\ I\ ([\alpha]Q)]\!]$. By the semantics of the box operator in dL, this means that there exists an $\omega'' \in \mathscr{S}$ such that $(\omega, \omega'') \in [\![\mathsf{translate\_program}\ I\ V([\alpha]Q)\ \alpha]\!]$ but $\omega'' \notin [\![\mathsf{translate\_formula}\ I\ Q]\!]$. Note that $V([\alpha]Q)$ contains all variables that appear in α. Then, by the inductive hypothesis (the second bullet of Theorem 3.6), we have that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ and for all $x \in V([\alpha]Q)$, $\omega'(x) = \omega''(x)$.

  Note that since $V([\alpha]Q)$ contains all variables that appear in Q, and ω' and ω'' agree on all variables that appear in $V([\alpha]Q)$, ω' and ω'' must agree on all variables that appear in Q. By a straightforward induction on Q, this means that $\omega' \in I[\![Q]\!]$ if and only if $\omega'' \in I[\![Q]\!]$. Finally, by the inductive hypothesis, $\omega'' \in I[\![Q]\!]$ if and only if $\omega'' \in [\![\mathsf{translate\_program}\ I\ Q]\!]$. Since $\omega'' \notin [\![\mathsf{translate\_formula}\ I\ Q]\!]$, this entails that $\omega' \notin I[\![Q]\!]$.

  Thus, we have shown that there exists an $\omega' \in \mathscr{S}$ such that $(\omega, \omega') \in I[\![\alpha]\!]$ but $\omega' \notin I[\![Q]\!]$. By the semantics of the box operator in Controller Aware dL, this means that $\omega \notin I[\![[\alpha]Q]\!]$, as desired.

Hybrid Program Cases:

Case $\alpha = (x := e)$**:** translate_program $I\ V\ (x := e) = (x :=$ translate_term e$)$.

- Let $(\omega, \omega') \in I[\![x := e]\!]$ and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, ω' must equal ω except that $\omega'(x) = \omega[\![e]\!]$. Then, let ν' equal ν except that $\nu'(x) = \nu[\![$translate_term $e]\!]$. Note that by the semantics of dL, $(\nu, \nu') \in [\![x :=$ translate_term $e]\!]$.

  Since $\omega[\![e]\!] = \nu[\![e]\!]$, as e only contains variables which ω and ν agree on (namely, variables in V), and since $\nu[\![e]\!]$ in Controller Aware dL semantics equals $\nu[\![$translate_term $e]\!]$ in dL semantics (by the inductive hypothesis), it follows that that $\nu'(x) = \omega'(x)$. Additionally, for any variable $y \in V$ such that $y \neq x$, $\nu'(y) = \nu(y) = \omega(y) = \omega'(y)$. Therefore, for all variables in $z \in V$, $\nu'(z) = \omega'(z)$, regardless of whether $z = x$. This fact, in conjunction with the fact that $(\nu, \nu') \in [\![x :=$ translate_term $e]\!]$, yields the desired result.

- Let $(\nu, \nu') \in [\![x :=$ translate_term $e]\!]$ and let ω agree with ν on all variables in V. By the semantics of dL, ν' must equal ν except that $\nu'(x) = \nu[\![$translate_term $e]\!]$. Then, let ω' equal ω except that $\omega'(x) = \omega[\![e]\!]$. Note that by the semantics of Controller Aware dL, $(\omega, \omega') \in I[\![x := e]\!]$.

  Since $\omega[\![e]\!] = \nu[\![e]\!]$, as e only contains variables which ω and ν agree on (namely, variables in V), and since $\nu[\![e]\!]$ in Controller Aware dL semantics equals $\nu[\![$translate_term $e]\!]$ in dL semantics (by the inductive hypothesis), it follows that that $\omega'(x) = \nu'(x)$. Additionally, for any variable $y \in V$ such that $y \neq x$, $\nu'(y) = \nu(y) = \omega(y) = \omega'(y)$. Therefore, for all variables in $z \in V$, $\nu'(z) = \omega'(z)$, regardless of whether $z = x$. This fact, in conjunction with the fact that $(\omega, \omega') \in I[\![x := e]\!]$, yields the desired result.

Case $\alpha = (x := *)$**:** translate_program $I\ V\ (x := *) = (x := *)$.

- Let $(\omega, \omega') \in I[\![x := *]\!]$, and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, ω' must equal ω except at x. Then, let ν' equal ν except that $\nu'(x) = \omega'(x)$. Note that by the semantics of dL, $(\nu, \nu') \in [\![x := *]\!]$.

  For all $y \in V$, either $y = x$ or $y \neq x$. If $y = x$, then $\omega'(y) = \nu'(y)$ by definition. If $y \neq x$, then $\omega'(y) = \omega(y) = \nu(y) = \nu'(y)$. Therefore, ν' and ω' agree on all variables in V. This fact, in conjunction with the fact that $(\nu, \nu') \in [\![x := *]\!]$, yields the desired result.

- Let $(\nu, \nu') \in [\![x := *]\!]$ and let ω agree with ν on all variables in V. By the semantics of dL, ν' must equal ν except at x. Then, let ω' equal ω except that $\omega'(x) = \nu'(x)$. Note that by the semantics of Controller Aware dL, $(\omega, \omega') \in I[\![x := *]\!]$.

  For all $y \in V$, either $y = x$ or $y \neq x$. If $y = x$, then $\omega'(y) = \nu'(y)$ by definition. If $y \neq x$, then $\omega'(y) = \omega(y) = \nu(y) = \nu'(y)$. Therefore, ω' and ν' agree on all variables in V. This fact, in conjunction with the fact that $(\omega, \omega') \in I[\![x := *]\!]$, yields the desired result.

Case $\alpha = ?P$**:** translate_program $I\ V\ (?P) = ?($translate_formula $I\ P)$

- Let $(\omega, \omega') \in I[\![?P]\!]$, and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, $\omega' = \omega$ and $\omega \in I[\![P]\!]$. By the inductive hypothesis, $\nu \in I[\![P]\!]$ if and only if $\nu \in [\![$translate_formula $I\ P]\!]$. Additionally, since ν agrees with ω on all variables in V, which includes all variables in P, it can be shown by a straightforward induction on P that $\omega \in I[\![P]\!]$ if and only if $\nu \in I[\![P]\!]$. Since $\omega \in I[\![P]\!]$, these facts entail that $\nu \in [\![$translate_formula $I\ P]\!]$. Then, by the semantics of dL, $(\nu, \nu) \in [\![?($translate_formula $I\ P)]\!]$. Since ν agrees with ω on all variables in V, this yields the desired result.

- Let $(\nu, \nu') \in [\![?(\mathsf{translate\_formula}\ I\ P)]\!]$, and let ω agree with ν on all variables in V. By the semantics of dL, $\nu' = \nu$ and $\nu \in [\![\mathsf{translate\_formula}\ I\ P]\!]$. By the inductive hypothesis, $\nu \in I[\![P]\!]$ if and only if $\nu \in [\![\mathsf{translate\_formula}\ I\ P]\!]$. So $\nu \in I[\![P]\!]$. Additionally, since ν agrees with ω on all variables in V, which includes all variables in P, it can be shown by a straightforward induction on P that $\omega \in I[\![P]\!]$ if and only if $\nu \in I[\![P]\!]$. Since $\nu \in I[\![P]\!]$, this means that $\omega \in I[\![P]\!]$. Then, by the semantics of Controller Aware dL, $(\omega, \omega) \in I[\![?P]\!]$. Since ω agrees with ν on all variables in V, this yields the desired result.

Case $\alpha = (x' = e\ \&\ P)$**:**
$\mathsf{translate\_program}\ I\ V\ (x' = e\ \&\ P) = (x' = (\mathsf{translate\_term}\ e)\ \&\ (\mathsf{translate\_formula}\ I\ P))$.

- Let $(\omega, \omega') \in I[\![x' = e\ \&\ P]\!]$, and let ν agree with ω on all variables in V. Then, by the semantics of Controller Aware dL, $\omega = \varphi(0)$ and $\omega' = \varphi(r)$ for some solution $\varphi : [0, r] \to \mathscr{S}$ of some duration r satisfying $I, \varphi \vDash x' = e\ \&\ P$, meaning that at all times $0 \leq z \leq r$, $\varphi(z) \in I[\![x' = e \wedge P]\!]$ with $\varphi(z)(x')$ defined as $(d\varphi(t)(x)/dt)(z)$ and $\varphi(z) = \varphi(0)$ except at x and x'.

  Let $\varphi' : [0, r] \to \mathscr{S}$ be a solution defined as follows. For all $z \in [0, r]$, let $\varphi'(z)$ be the state that agrees with $\varphi(z)$ on all variables in V, and agrees with ν on all variables not in V. We wish to show that $(\nu, \varphi'(r)) \in [\![\mathsf{translate\_program}\ I\ V\ (x' = e\ \&\ P)]\!]$ and $\varphi'(r)$ agrees with ω' on all variables in V. These two statements entail the desired result.

  To show that $(\nu, \varphi'(r)) \in [\![\mathsf{translate\_program}\ I\ V\ (x' = e\ \&\ P)]\!]$, the semantics of dL require that $\varphi'(0) = \nu$ and at all times $0 \leq z \leq r$, $\varphi'(z) \in [\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!]$. To see that $\varphi'(0) = \nu$, let y be an arbitrary variable in $\mathscr{V}$. If $y \notin V$, then $\varphi'(0)(y) = \nu(y)$ by definition. If $y \in V$, then $\varphi'(0)(y) = \varphi(0)(y) = \omega(y)$. Since ν and ω agree on all variables in V, this means that $\varphi'(0)(y) = \nu(y)$. So for all $y \in \mathscr{V}$, $\varphi'(0)(y) = \nu(y)$, meaning $\varphi'(0) = \nu$ as desired.

  To see that at all times, $0 \leq z \leq r$, $\varphi'(z) \in [\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!]$, note that by the inductive hypothesis, $[\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!] = I[\![x' = e \wedge P]\!]$[15]. Additionally, since $\varphi(z)$ agrees with $\varphi'(z)$ on all variables in V, including all variables that appear in $(x' = e \wedge P)$, $\varphi'(z) \in I[\![x' = e \wedge P]\!]$ if and only if $\varphi(z) \in I[\![x' = e \wedge P]\!]$. These facts, along with the fact that $\varphi(z) \in I[\![x' = e \wedge P]\!]$, yield that $\varphi'(z) \in [\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!]$, as desired.

  So all that is left is to show that $\varphi'(r)$ agrees with ω' on all variables in V. Note that $\omega' = \varphi(r)$, which, by definition, agrees with $\varphi'(r)$ on all variables in V. So it immediately follows that $\varphi'(r)$ agrees with ω' on all variables in V, as desired.

- Let $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ (x' = e\ \&\ P)]\!]$, and let ω agree with ν on all variables in V. Then, by the semantics of dL, $\nu = \varphi'(0)$ and $\nu' = \varphi'(r)$ for some solution $\varphi' : [0, r] \to \mathscr{S}$ of some duration r satisfying $I, \varphi \vDash x' = (\mathsf{translate\_term}\ e)\ \&\ (\mathsf{translate\_formula}\ I\ P)$.

  Let $\varphi : [0, r] \to \mathscr{S}$ be a solution defined as follows. For all $z \in [0, r]$, let $\varphi(z)$ be the state that agrees with $\varphi'(z)$ on all variables in V, and agrees with ω on all variables not in V. We wish to show that $(\omega, \varphi(r)) \in I[\![x' = e\ \&\ P]\!]$ and $\varphi(r)$ agrees with ν' on all variables in V. These two statements entail the desired result.

[15] Technically, $x' = e \wedge P$ is not a subformula of the current α. However, this inductive call is nonetheless permissible because it reduces the number of &'s in the term, formula, or program being inducted on, and no inductive call is ever made that increases the number of &'s in the term, formula, or program being inducted on. In this sense, although $x' = e \wedge P$ is not a subformula of the current α, we can view it as smaller than the current α, and therefore, appropriate to invoke the inductive hypothesis on.

To show that $(\omega, \varphi(r)) \in I[\![x' = e \;\&\; P]\!]$, the semantics of Controller Aware dL require that $\varphi(0) = \omega$ and at all times $0 \leq z \leq r$, $\varphi(z) \in I[\![x' = e \wedge P]\!]$. To see that $\varphi(0) = \omega$, let y be an arbitrary variable in $\mathscr{V}$. If $y \notin V$, then $\varphi(0)(y) = \omega(y)$ by definition. If $y \in V$, then $\varphi(0)(y) = \varphi'(0)(y) = \nu(y)$. Since ω agrees with ν on all variables in V, this means that $\varphi(0)(y) = \omega(y)$. So for all $y \in \mathscr{V}$, $\varphi(0)(y) = \omega(y)$, meaning $\varphi(0) = \omega$ as desired.

To see that at all times, $0 \leq z \leq r$, $\varphi(z) \in I[\![x' = e \wedge P]\!]$, note that by the inductive hypothesis, $I[\![x' = e \wedge P]\!] = [\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!]$. Additionally, since $\varphi(z)$ agrees with $\varphi'(z)$ on all variables in V, including all variables that appear in $(x' = e \wedge P)$, $\varphi'(z) \in I[\![x' = e \wedge P]\!]$ if and only if $\varphi(z) \in I[\![x' = e \wedge P]\!]$. These facts, along with the fact that $\varphi'(z) \in [\![x' = (\mathsf{translate\_term}\ e) \wedge (\mathsf{translate\_formula}\ I\ P)]\!]$, yield that $\varphi(z) \in I[\![x' = e \wedge P]\!]$, as desired.

So all that is left is to show that $\varphi(r)$ agrees with ν' on all variables in V. Note that $\nu' = \varphi'(r)$, which, by definition, agrees with $\varphi(r)$ on all variables in V. So it immediately follows that $\varphi(r)$ agrees with ν' on all variables in V, as desired.

Case $\alpha = \beta \cup \gamma$**:** $\mathsf{translate\_program}\ I\ V\ (\beta \cup \gamma) = (\mathsf{translate\_program}\ I\ V\ \beta) \cup (\mathsf{translate\_program}\ I\ V\ \gamma)$.

- Let $(\omega, \omega') \in I[\![\beta \cup \gamma]\!]$, and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, either $(\omega, \omega') \in I[\![\beta]\!]$ or $(\omega, \omega') \in I[\![\gamma]\!]$. Without loss of generality, let $(\omega, \omega') \in I[\![\beta]\!]$. Then, by the inductive hypothesis, there exists a ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \beta]\!]$ and ν' agrees with ω' on all variables in V. By the semantics of dL, this entails that $(\nu, \nu') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta) \cup (\mathsf{translate\_program}\ I\ V\ \gamma)]\!]$, as desired.
- Let $(\nu, \nu') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta) \cup (\mathsf{translate\_program}\ I\ V\ \gamma)]\!]$, and let ω agree with ν on all variables in V. By the semantics of dL, either $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \beta]\!]$ or $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \gamma]\!]$. Without loss of generality, let $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \beta]\!]$. Then, by the inductive hypothesis, there exists an ω' such that $(\omega, \omega') \in I[\![\beta]\!]$ and ω' agrees with ν' on all variables in V. By the semantics of Controller Aware dL, this entails that $(\omega, \omega') \in I[\![\beta \cup \gamma]\!]$, as desired.

Case $\alpha = \beta; \gamma$**:** $\mathsf{translate\_program}\ I\ V\ (\beta; \gamma) = (\mathsf{translate\_program}\ I\ V\ \beta); (\mathsf{translate\_program}\ I\ V\ \gamma)$

- Let $(\omega, \omega'') \in I[\![\beta; \gamma]\!]$, and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, there must exist a state ω' such that $(\omega, \omega') \in I[\![\alpha]\!]$ and $(\omega', \omega'') \in I[\![\beta]\!]$. By the inductive hypothesis, there exists a state ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \beta]\!]$ and ν' agrees with ω' on all variables in V. Then, by a subsequent application of the inductive hypothesis, since ν' agrees with ω' on all variables in V, there exists a state ν'' such that $(\nu', \nu'') \in [\![\mathsf{translate\_program}\ I\ V\ \gamma]\!]$ and ν'' agrees with ω'' on all variables in V. By the semantics of dL, this entails that $(\nu, \nu'') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta); (\mathsf{translate\_program}\ I\ V\ \gamma)]\!]$. Since ν'' agrees with ω'' on all variables in V, this yields the desired result.
- Let $(\nu, \nu'') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta); (\mathsf{translate\_program}\ I\ V\ \gamma)]\!]$, and let ω agree with ν on all variables in V. By the semantics of dL, there must exist a state ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ \beta]\!]$ and $(\nu', \nu'') \in [\![\mathsf{translate\_program}\ I\ V\ \gamma]\!]$. By the inductive hypothesis, there exists a state ω' such that $(\omega, \omega') \in I[\![\beta]\!]$ and ω' agrees with ν' on all variables in V. Then, by a subsequent application of the inductive hypothesis, since ω' agrees with ν' on all variables in V, there exists a state ω'' such that $(\omega', \omega'') \in I[\![\gamma]\!]$ and ω'' agrees with ν'' on all variables in V. By the semantics of Controller Aware dL, this entails that $(\omega, \omega'') \in I[\![\beta; \gamma]\!]$. Since ω'' agrees with ν'' on all variables in V, this yields the desired result.

Case $\alpha = \beta^*$**:** $\mathsf{translate\_program}\ I\ V\ (\beta^*) = (\mathsf{translate\_program}\ I\ V\ \beta)^*$.

- Let $(\omega, \omega') \in I[\![\beta^*]\!]$, and let ν agree with ω on all variables in V. By the semantics of Controller Aware dL, there must be some $n \in \mathbb{N}$ such that $(\omega, \omega') \in I[\![\beta^n]\!]$. Then, by the inductive hypothesis on β^n,[16] there exists a state ν' such that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ (\beta^n)]\!]$ and ν' agrees with ω' on all variables in V. Note that $\mathsf{translate\_program}\ I\ V\ (\beta^n) = (\mathsf{translate\_program}\ I\ V\ \beta)^n$.[17] Then, by the semantics of dL, it follows that $(\nu, \nu') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta)^*]\!]$. Since ν' agrees with ω' on all variables in V, this yields the desired result.
- Let $(\nu, \nu') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta)^*]\!]$, and let ω agree with ν on all variables in V. By the semantics of dL, there must be some $n \in \mathbb{N}$ such that $(\nu, \nu') \in [\![(\mathsf{translate\_program}\ I\ V\ \beta)^n]\!]$. Note that $(\mathsf{translate\_program}\ I\ V\ \beta)^n = \mathsf{translate\_program}\ I\ V\ (\beta^n)$, and therefore, $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ (\beta^n)]\!]$. Then, by the inductive hypothesis, there exists a state ω' such that $(\omega, \omega') \in I[\![\beta^n]\!]$ and ω' agrees with ν' on all variables in V. By the semantics of Controller Aware dL, this entails that $(\omega, \omega') \in I[\![\beta^*]\!]$. Since ω' agrees with ν' on all variables in V, this yields the desired result.

Case $\alpha = \text{measure } x$**:**
$\mathsf{translate\_program}\ I\ V\ (\text{measure } x) =$
$\gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha;\ y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ {\sim}x;$
$\gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta;\ z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ {\sim}x;$
${\sim}x := *;\ ?(y \leq {\sim}x \wedge {\sim}x \leq z)$

- Let $(\omega, \omega') \in I[\![\text{measure } x]\!]$ and let ν agree with ω on all variables in V. Then, let ν_1 be the unique state such that $(\nu, \nu_1) \in [\![\gamma]\!]$. The fact that ν_1 exists and is unique is guaranteed by the semantics of assignment and sequential composition. From γ's definition, the following two facts follow:
  - ν agrees with ν_1 on all variables in $(V \cup \{y, z\})$.
  - For each $x \in V$, $\nu(x) = \nu_1(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$.

  Let $(a, b) = I_{measure}\ x\ \omega$. Since α and β define $I_{measure}\ x$, a must be the minimum value in the set $\{c \mid \omega''({\sim}x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$. Then, let ω'' be a state such that $(\omega, \omega'') \in [\![\alpha]\!]$ and $\omega''({\sim}x) = a$. Note that for each x that appears in α, $\omega(x) = \nu(x)$, as all variables that appear in α are in V[18]. Then, since for each $x \in V$, $\nu(x) = \nu_1(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$, it follows that for each variable x that appears in α, $\omega(x) = \nu_1(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$. From Lemma 3.2, this entails that there exists a ν_2 such that $(\nu_1, \nu_2) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha]\!]$, and for each variable x that appears in α, $\omega''(x) = \nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$. In particular, $\omega''({\sim}x) = \nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ {\sim}x)$, meaning $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ {\sim}x) = a$. Additionally, from Lemma 3.1, ν_2 agrees with ν_1 on all variables in $(V \cup \{y, z\})$. Since ω agrees with ν on all variables in V, and ν agrees with ν_1 on all variables in V, this entails that ω agrees with ν_2 on all variables that appear in V. So the important known properties about ν_2 are that:

[16] Technically, β^n is not a subformula of the current α. However, the inductive call is nonetheless permissible because it reduces the depth of the deepest nondeterministic loop in the term, formula or program, being inducted on. By depth of a nondeterministic loop, we mean the number of nondeterministic loops that a nondeterministic loop is inside of (e.g. α^* has depth 0 and the inner loop of $(\alpha^*)^*$ has depth 1). Since no inductive call is ever made that increases the depth of the deepest nondeterministic loop in the term, formula, or program being inducted on, our induction remains well-founded.

[17] This fact can be proven by a straightforward induction on n.

[18] Recall that in our extension of the definition of the set of variables that appear in a program, we state that the set of variables that appear in "measure x" is $V(\alpha) \cup V(\beta) \cup \{x, {\sim}x\}$, where α and β define $I_{measure}\ x$. It is for this reason that the set of variables that appear in α is a subset of V

- $(\nu_1, \nu_2) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha]\!]$.
- $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$ is the minimum value in the set $\{c \mid \omega''(\sim x) = c$ where $(\omega, \omega'') \in [\![\alpha]\!]\}$.
- ν_2 agrees with ω on all variables that appear in V.

Let $\nu_3 = \nu_2$ except that $\nu_3(y) = \nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$. By the semantics of the assignment operator in dL, $(\nu_2, \nu_3) \in [\![y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x]\!]$. Additionally, since $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$ is the minimum value in the set $\{c \mid \omega''(\sim x) = c$ where $(\omega, \omega'') \in [\![\alpha]\!]\}$, $\nu_3(y)$ is the minimum value in the set $\{c \mid \omega''(\sim x) = c$ where $(\omega, \omega'') \in [\![\alpha]\!]\}$.

Then, let ν_4 be the unique state such that $(\nu_3, \nu_4) \in [\![\gamma]\!]$. The fact that ν_4 exists and is unique is guaranteed by the semantics of assignment and sequential composition. From $\gamma's$ definition, and the facts known about ν_3, the following must hold about ν_4.

- ν_4 agrees with ν_3 on all variables in $(V \cup \{y, z\})$. Since ν_2 agrees with with ω on all variables that appear in V, and since ν_3 agrees with ν_2 on all variables except y (which is not in V), this means that ν_4 agrees with ω on all variables that appear in V.
- For each $x \in V$, $\nu_3(x) = \nu_4(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$. Since ν_3 agrees with ω on all values in V, this means that for each $x \in V$, $\omega(x) = \nu_4(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$.
- Since γ does not alter y, $\nu_4(y)$ is the minimum value in the set $\{c \mid \omega''(\sim x) = c$ where $(\omega, \omega'') \in [\![\alpha]\!]\}$.

Since $(a, b) = I_{measure}\ x\ \omega$, and since α and β define $I_{measure}\ x$, b must be the maximum value in the set $\{c \mid \omega'''(\sim x) = c$ where $(\omega, \omega''') \in [\![\beta]\!]\}$. Then let ω''' be a state such that $(\omega, \omega''') \in [\![\beta]\!]$ and $\omega'''(\sim x) = b$. By Lemma 3.2, there must exist a ν_5 such that $(\nu_4, \nu_5) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta]\!]$ and for each x that appears in β, $\omega'''(x) = \nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$. In particular, $\omega'''(\sim x) = \nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$, meaning $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) = b$. Additionally, from Lemma 3.1, ν_5 agrees with ν_4 on all variables in $(V \cup \{y, z\})$, meaning ν_5 agrees with ω on all variables in V and $\nu_5(y) = a$.

Let $\nu_6 = \nu_5$ except that $\nu_6(z) = \nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$. By the semantics of the assignment operator in dL, $(\nu_5, \nu_6) \in [\![z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x]\!]$. From this, and the facts known about ν_5, the following is known about ν_6:

- ν_6 agrees with ν_5 on y and all variables in V. This means that ν_6 agrees with ω on all variables in V and $\nu_6(y) = a$.
- $\nu_6(z) = \nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) = b$.
- By the semantics of sequential composition, $(\nu, \nu_6) \in$
  $[\![\gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha;\ y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;$
  $\gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta;\ z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x]\!]$

Let $\nu_7 = \nu_6$ except that $\nu_7(\sim x) = \omega'(\sim x)$. Recall that by the semantics of "measure x" in Controller Aware dL, and the fact that $(\omega, \omega') \in I[\![\text{measure } x]\!]$, $\omega' = \omega$ except that $\omega'(\sim x)$ is between a and b inclusive. I assert that $(\nu_6, \nu_7) \in [\![\sim x := *; ?(y \leq \sim x \land \sim x \leq z)]\!]$. By the semantics of nondeterministic assignment, sequential composition, and the test operator, this can be true if and only if $\nu_6(y) \leq \omega'(\sim x)$ and $\omega'(\sim x) \leq \nu_6(z)$. Since $\nu_6(y) = a$ and $\nu_6(z) = b$, this is equivalent to saying that $a \leq \omega'(\sim x) \leq b$. Since $\omega'(\sim x)$ must be between a and b inclusive, this is true, and so $(\nu_6, \nu_7) \in [\![\sim x := *; ?(y \leq \sim x \land \sim x \leq z)]\!]$. By the semantics of sequential composition, this implies that $(\nu, \nu_7) \in [\![\mathsf{translate\_program}\ I\ V\ (\text{measure } x)]\!]$.

Therefore, we have shown that there exists a state ν_7 such that $(\nu, \nu_7) \in [\![\mathsf{translate\_program}\ I\ V\ (\text{measure}\ x)]\!]$. Furthermore, since ν_7 agrees with ν_6 on all variables in V except $\sim x$, ν_7 agrees with ω on all variables in V except $\sim x$. Since $\omega' = \omega$ except at $\sim x$, this means that for all variables in V except $\sim x$, ω' and ν_7 agree. Finally, from the definition of ν_7, $\nu_7(\sim x) = \omega'(\sim x)$, and so ν_7 agrees with ω' on all variables in V, as desired.

- Let $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ (\text{measure}\ x)]\!]$, and let ω agree with ν on all variables in V. Then, let ν_1 be the unique state such that $(\nu, \nu_1) \in [\![\gamma]\!]$. The fact that ν_1 exists and is unique is guaranteed by the semantics of assignment and sequential composition. From γ's definition, the following two facts follow:
  - ν agrees with ν_1 on all variables in $(V \cup \{y, z\})$.
  - For each $x \in V$, $\nu(x) = \nu_1(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$.

  From the semantics of sequential composition, and the fact that $(\nu, \nu') \in [\![\mathsf{translate\_program}\ I\ V\ (\text{measure}\ x)]\!]$, there must be some state ν_2 such that $(\nu_1, \nu_2) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha]\!]$ and $(\nu_2, \nu') \in [\![y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x; \gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta;\ z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$. Since ν agrees with ω on all variables in V, it must be the case that for each $x \in V$, $\nu_1(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x) = \omega(x)$. Then, from Lemma 3.3, it follows that there exists an ω'' such that $(\omega, \omega'') \in I[\![\alpha]\!]$ and for each x that appears in α, $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x) = \omega''(x)$. In particular, $\omega''(\sim x) = \nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$.

  Since $(\omega, \omega'') \in I[\![\alpha]\!]$, it must be the case that $\omega''(\sim x)$ is greater than or equal to the minimum value of the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$. Therefore, if $I_{measure}\ x\ \omega = (a, b)$, $\omega''(\sim x) \geq a$. Since $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) = \omega''(\sim x)$, this entails that $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) \geq a$ as well. So the important known properties about ν_2 are that:
  - $(\nu_1, \nu_2) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \alpha]\!]$.
  - $(\nu_2, \nu') \in [\![y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta;\ z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$
  - $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$ is greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$.
  - ν_2 agrees with ω on all variables that appear in V. This is because ν_1 agrees with ν on all variables in V, ν agrees with ω on all variables in V, and ν_2 agrees with ν_1 on all variables in $(V \cup \{y, z\})$ from Lemma 3.1.

  Let $\nu_3 = \nu_2$ except that $\nu_3(y) = \nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$. By the semantics of the assignment operator in dL, $(\nu_2, \nu_3) \in [\![y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x]\!]$. Additionally, since $\nu_2(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$ is greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$, $\nu_3(y)$ is also greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$.

  Then, let ν_4 be the unique state such that $(\nu_3, \nu_4) \in [\![\gamma]\!]$. The fact that ν_4 exists and is unique is guaranteed by the semantics of assignment and sequential composition. From $\gamma's$ definition, and the facts known about ν_3, the following must hold about ν_4.
  - ν_4 agrees with ν_3 on all variables in $(V \cup \{y, z\})$. Since ν_2 agrees with with ω on all variables that appear in V, and since ν_3 agrees with ν_2 on all variables except y (which is not in V), this means that ν_4 agrees with ω on all variables that appear in V.

- For each $x \in V$, $\nu_3(x) = \nu_4(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$. Since ν_3 agrees with ω on all values in V, this means that for each $x \in V$, $\omega(x) = \nu_4(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x)$.
- Since γ does not alter y, $\nu_4(y)$ is greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$.

From the semantics of sequential composition, and the fact that $(\nu_2, \nu') \in [\![y := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \gamma;\ \mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta;\ z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$, there must be some state ν_5 such that $(\nu_4, \nu_5) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta]\!]$ and $(\nu_5, \nu') \in [\![z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$. Since ν_4 agrees with ω on all variables that appear in V, by Lemma 3.3, it follows that there exists an ω''' such that $(\omega, \omega''') \in I[\![\beta]\!]$ and for each x that appears in β, $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ x) = \omega'''(x)$. In particular, $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) = \omega'''(\sim x)$.

Since $(\omega, \omega''') \in I[\![\beta]\!]$, it must be the case that $\omega'''(\sim x)$ is less than or equal to the maximum value of the set $\{c \mid \omega'''(\sim x) = c \text{ where } (\omega, \omega''') \in [\![\beta]\!]\}$. Therefore, if $I_{measure}\ x\ \omega = (a, b)$, $\omega'''(\sim x) \leq b$. Since $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) = \omega'''(\sim x)$, this entails that $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x) \leq b$ as well. So the important known properties about ν_5 are that:

- $(\nu_4, \nu_5) \in [\![\mathsf{freshen\_program}\ (V \cup \{y, z\})\ \beta]\!]$.
- $(\nu_5, \nu') \in [\![z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$.
- $\nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$ is less than or equal to the maximum value in the set $\{c \mid \omega'''(\sim x) = c \text{ where } (\omega, \omega''') \in [\![\beta]\!]\}$.
- From Lemma 3.1, $\nu_5(y) = \nu_4(y)$. Therefore, $\nu_5(y)$ is greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$.
- ν_5 agrees with ω on all variables that appear in V. This follows from Lemma 3.1 and the fact that ν_4 agrees with ω on all variables that appear in V.

Let $\nu_6 = \nu_5$ except that $\nu_6(z) = \nu_5(\mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x)$. By the semantics of the assignment operator in dL, ν_6 is the unique state such that $(\nu_5, \nu_6) \in [\![z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x]\!]$. Then, from the fact that $(\nu_5, \nu') \in [\![z := \mathsf{freshen\_var}\ (V \cup \{y, z\})\ \sim x;\ \sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$ and the semantics of sequential composition, $(\nu_6, \nu') \in [\![\sim x := *;\ ?(y \leq \sim x \wedge \sim x \leq z)]\!]$. By the semantics of nondeterministic assignment, sequential composition, and the test operator, this entails that $\nu_6 = \nu'$ except at $\sim x$ where $\nu_6(y) \leq \nu'(\sim x) \leq \nu_6(z)$.

From the facts known about ν_5, ν_6, and the relationship between ν' and ν_6, the following is known about ν':

- ν' agrees with ω on all variables that appear in V except $\sim x$. This follows from the facts that ν' agrees with ν_6 on all variables that appear in V except $\sim x$, $\nu_6 = \nu_5$ except at z, and ν_5 agrees with ω on all variables that appear in V.
- $\nu'(\sim x)$ is greater than or equal to $\nu_6(y)$, which means it is greater than or equal to the minimum value in the set $\{c \mid \omega''(\sim x) = c \text{ where } (\omega, \omega'') \in [\![\alpha]\!]\}$. In other words, if $I_{measure}\ x\ \omega = (a, b)$, then $a \leq \nu'(\sim x)$.
- $\nu'(\sim x)$ is less than or equal to $\nu_6(z)$, which means it is less than or equal to the maximum value in the set $\{c \mid \omega'''(\sim x) = c \text{ where } (\omega, \omega''') \in [\![\beta]\!]\}$. In other words, if $I_{measure}\ x\ \omega = (a, b)$, then $\nu'(\sim x) \leq b$.

Consider $\omega' = \omega$ except that $\omega'(\sim x) = \nu'(\sim x)$. For all variables in V except $\sim x$, both ν' and ω' agree with ω, and therefore, agree with each other. Additionally, by definition, $\omega'(\sim x) = \nu'(\sim x)$, so it follows that for all variables in $v \in V$, ω' and ν' agree on v. Finally, since $a \leq \nu'(\sim x) \leq b$ where $(a, b) = I_{measure}\, x\, \omega$, by the definition of $I[\![\text{measure } x]\!]$, it follows that $(\omega, \omega') \in I[\![\text{measure } x]\!]$. With this, we have shown that there exists an ω' such that $(\omega, \omega') \in I[\![\text{measure } x]\!]$ and ω' agrees with ν' on all variables in V, as desired.

Case $\alpha = \text{set } x\ e$**:** This case directly mirrors the previous case.