André Platzer

Lecture Notes on Foundations of Cyber-Physical Systems

15-424/624/824 Foundations of Cyber-Physical Systems

Chapter 18
Axioms & Uniform Substitutions

Synposis This chapter explores a succinct approach for soundly implementing rigorous reasoning for hybrid systems. Unlike previous chapters, this chapter is not concerned with identifying new reasoning principles for cyber-physical systems, but, rather, focuses on how they can best be implemented correctly. Uniform substitutions are identified as a simple concept based upon which differential dynamic logic proof systems can be implemented easily. Uniform substitutions uniformly instantiate predicate symbols by formulas. Because all reasoning can be reduced to finding the appropriate sequence of uniform substitutions, this makes it possible to implement theorem provers with a small soundness-critical core.

18.1 Introduction

The logic and reasoning principles for hybrid systems (Part I), differential equations (Part II), and hybrid games (Part III) identified in previous chapters are conducive to quite simple correctness arguments. Proof principles decouple the question of what a correct argument is from the question how to find it. Soundness even of the most big and complicated proofs directly follows from the soundness of each of the proof steps. Every proof step uses one of a small set of dL axioms and proof rules, which can each be proved sound individually quite easily. The transfer of soundness was already rooted in Definition 6.2 on p. 165, which defined a proof rule to be sound iff the validity of all premises implies the validity of the conclusion. For axioms, Definition 5.1 on p. 132 defined an axiom to be sound iff all its instances are valid. Since proofs only consist of axiom composed with proof rules, this implies that the conclusion of every (completed) proof is valid.

The remaining challenge for soundness of the proofs is to ensure that all axioms and proof rules are also implemented correctly in a theorem prover. The primary obstacle is that the reasoning principles identified so far were considered as *axiom schemata*, i.e. they stand for an infinite family of formulas of the same shape. That is easily said, but still needs some form of implementation. Moreover, a fair

number of the axiom schemata have soundness-critical side conditions that need to be respected to guarantee soundness. That these soundness-critical side conditions cannot be elided is most obvious in the vacuous axiom schema from Lemma 5.11:

$$\mathrm{V}\quad p \rightarrow [\alpha]p \qquad (FV(p) \cap BV(\alpha) = \emptyset)$$

Of course, every use of axiom schema V needs to ensure that the same formula p is used in the precondition and the postcondition. But without checking that no free variable of p is written to in the hybrid program α, it would be unsound to conclude that p always holds after running HP α if p was true initially. After all, if α changes a variable that p reads, its truth-value may change. It is only thanks to this side condition that the following invalid formula is not provable by axiom V:

$$x \geq 0 \rightarrow [x' = -5]\, x \geq 0$$

The differential equation solution axiom schema from Lemma 5.3 has even more complicated side conditions:

$$[']\quad [x' = f(x)]p(x) \leftrightarrow \forall t{\geq}0\, [x := y(t)]p(x) \qquad (y'(t) = f(y))$$

The soundness-critical side conditions for axiom schema $[']$ are:

1. The variable t needs to be fresh and cannot have occurred already, because it is supposed to represent the independent variable for time.
2. The function of time $y(\cdot)$ needs to solve the differential equation $y(t)' = f(y(t))$ and needs to be defined at all times that the quantifier for t quantifies over, because the continuous dynamics of the differential equation can only be equivalently replaced by a discrete assignment when $y(\cdot)$ is the correct solution of the differential equation.
3. The solution $y(\cdot)$ needs to solve the symbolic initial value condition $y(0) = x$ for the variable x, because we usually do not have a specific numerical initial value when using axiom schema $[']$.
4. The solution $y(\cdot)$ needs to cover all solutions parametrically, e.g., when the solution has different shapes for different choices of the initial value x.
5. The postcondition $p(x)$ cannot have differential symbol x' as a free variable, because it receives the value $f(x)$ after the differential equation but retains the initial value after the discrete assignment to x.[1]

A correct implementation of axiom schema $[']$, thus, amounts to an algorithm accepting every formula of this shape after checking all the required side-conditions. Fortunately, Part II already provided a substantially more elegant way of proving properties of differential equations by induction that also makes the solution axiom schema $[']$ superfluous [6], because it can be replaced by an appropriate differential cut to augment the evolution domain with the solution after a suitable differential ghost to shift the dynamics into the time domain $t' = 1$ (Chap. **??**). But the fact remains that axiom schemata have a tendency to require a somewhat unwieldy set of

[1] Of course, this is easily fixed by adding an assignment $x' := f(x)$ after the assignment to x.

side conditions that are soundness-critical and, thus, need to be enforced for every reasoning step. Compared to verification algorithms that do not even benefit from a similar logical foundation, it is still substantially easier to devise correct implementations of individual axiom schemata and then glue them together with correct implementations of proof rules. But this chapter will find a more straightforward way that is even easier to get correct.

The primary observation to make this happen comes from a shift in perspective that distinguishes between axioms and axiom schemata. An *axiom* is a single valid formula that is adopted as a basis for reasoning in a proof calculus. An *axiom schema* stands for an infinite family of formulas of the same shape (subject to the required side conditions) and, thus, needs to be implemented with an algorithm. Implementing an axiom is trivial, because an axiom is just a single formula in the object logic. The only downside is that the only formula that an axiom enables us to prove is literally that formula in verbatim, which we are rarely interested in proving.

Consequently, the missing element for a reasoning system that is based on axioms is a mechanism for instantiating them. Church's uniform substitutions [1] provide such a mechanism for first-order logic. They make it possible to instantiate predicate symbols by formulas and already check the required conditions to ensure that that instantiation is sound. Generalizing uniform substitutions from first-order logic to differential dynamic logic leads to the corresponding mechanism to implement flexible dL proving parsimoniously with uniform substitution as essentially the only proof rule [5, 6].

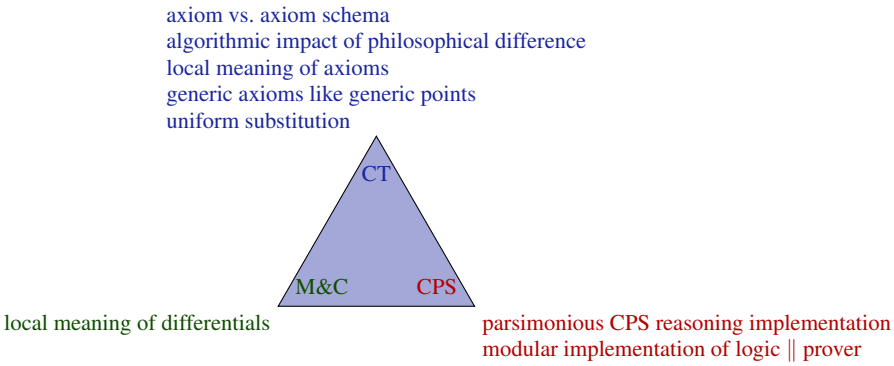
Differential dynamic logic provides sound reasoning principles, but uniform substitutions also make it easy to implement them correctly. Uniform substitutions are the secret for simple sound hybrid systems provers such as KeYmaera X [3]. This chapter has a major impact on the 1,700 lines of soundness-critical code in KeYmaera X compared to the 66,000 lines of soundness-critical code in its predecessor KeYmaera [7], which implements a schematic sequent calculus for dL [4].

The most important learning goals of this chapter are:

Modeling and Control: We will eventually see how the shift in perspective to axioms gives us an opportunity to reflect on the significance of the local meaning of differentials in hybrid systems.

Computational Thinking: This chapter investigates the relationship and fundamental difference of axioms versus axiom schemata. This philosophical distinction leads to a significant algorithmic impact on the style of implementing hybrid systems reasoning. This chapter also explores the local meaning of axioms, which is the axiomatic counterpart of how generic points are understood as a nondegenerate generalization of concrete points in algebraic geometry. The fundamental concept of uniform substitutions will be explored, which makes it possible to use axioms as if they were axiom schemata without the need for any additional mechanisms or side condition checking. This purely axiomatic reconsideration of the proof calculus for differential dynamic logic will lead to a new level of appreciation for what the axioms of differential dynamic logic already offered throughout this book without us noticing.

CPS Skills: We identify techniques for a parsimonious straightforward implementation of CPS reasoning. These techniques enable a modular implementation of the logic and the prover mostly independently in parallel, which reduces complexity and makes it easier to advance the reasoning techniques.



18.2 Axioms versus Axiom Schemata

Recall the axiom [∪] for hybrid programs with a nondeterministic choice $\alpha \cup \beta$ from Lemma 5.1 on p. 130:

$$[\cup] \quad [\alpha \cup \beta]P \leftrightarrow [\alpha]P \wedge [\beta]P \tag{18.1}$$

The innocent way of reading (18.1) is as an axiom schema [∪]. An axiom schema is meant to stand for the infinite family of formulas that have the shape of that axiom schema, so α, β are schema variables or placeholders for arbitrary HPs and P is a placeholder for an arbitrary dL formula. The left-hand side of axiom schema [∪] applies for any dL formula of the form $[\alpha \cup \beta]P$, so to any box modality of any HP that begins with a nondeterministic choice as the top-level operator and has any HPs as subprograms and any dL formula as a postcondition. For example, the left-hand side of axiom schema [∪] fits to dL formula $[x := x+1 \cup x' = x^2]x \geq 0$, which implies that the axiom schema [′] directly justifies the following equivalence:

$$[x := x+1 \cup x' = x^2]x \geq 0 \leftrightarrow [x := x+1]x \geq 0 \wedge [x' = x^2]x \geq 0 \tag{18.2}$$

Of course, this is not the only dL formula that needs to be recognized to be of the shape that axiom schema [∪] indicates. Here are a few more:

$$[x' = x^2 \cup x := x+1]\, x \geq 0 \leftrightarrow [x' = x^2]\, x \geq 0 \wedge [x := x+1]\, x \geq 0$$
$$[x' = 5 \cup x' = -x]\, x^2 \geq 5 \leftrightarrow [x' = 5]\, x^2 \geq 5 \wedge [x' = -x]\, x^2 \geq 5$$
$$[v := v+1; x' = v \cup x' = 2]\, x \geq 5 \leftrightarrow [v := v+1; x' = v]\, x \geq 5 \wedge [x' = 2]\, x \geq 5$$

A direct implementation of axiom schema $[\cup]$ consists of an algorithm that takes a dL formula as an input and decides whether that formula is of the form of schema $[\cup]$. Of course, it is crucially important that literally the same postcondition is used for all three modalities of axiom schema $[\cup]$. And it is important that the same HP is used in the left part of the nondeterministic choice $\alpha \cup \beta$ and in the first modality $[\alpha]$ on the right-hand side, and that the same HP is used in the right part of $\alpha \cup \beta$ that is also used in the second modality $[\beta]$ on the right-hand side.[2] Axiom schema $[\cup]$ does not even have any side conditions yet, but it already comes with a few tedious conditions to check (if implementing it an imperative programming language) or match correctly (in a functional programming language with pattern matching).

A more conscious way of reading (18.1) is as an axiom $[\cup]$ that literally only refers to one dL formula:

$$[\alpha \cup \beta]P \leftrightarrow [\alpha]P \wedge [\beta]P \tag{18.3}$$

Of course, we will still have to make sure that (18.3) actually is a syntactically well-formed dL formula, which it is not presently. The only formula that such an axiom $[\cup]$ ever proves is (18.3). That alone is not so useful, but an axiom is easily implemented, just by copying the dL formula (18.3) from axiom $[\cup]$ into the prover.

Alonzo Church's seminal observation is that the only operation it takes to make more use of an axiom is to provide a uniform substitution mechanism that replace parts of formulas with other formulas [1]. The trick is to identify when such a replacement is sound. Of course, Church did not know about differential dynamic logic yet, so he settled for first-order logic. But with a sufficiently generalized notion of uniform substitutions for differential dynamic logic [6], we can prove the dL formula (18.3) from axiom $[\cup]$ and then use a uniform substitution to prove (18.2) from (18.3). All this takes is the uniform substitution that substitutes $x := x+1$ for α and substitutes $x' = x^2$ for β and simultaneously also substitutes $x \geq 0$ for P uniformly everywhere in (18.3).

Now, the one crucial missing piece is a precise definition of this uniform substitution mechanism. The other crucial element is a precise understanding whether and what the uniform substitution mechanism needs to check to ensure that all its replacements preserved soundness. And the final missing element is the question what precise form the syntactic expressions α, β, and P take in (18.1) if it is to be taken literally as an axiom. Then the same process needs to be repeated with an axiomatic reinterpretation of all other dL axioms to find out how they can all be read as axioms instead of as significantly more complicated axiom schemata.

[2] If the formula that is used in place of P has a modality, then the textual description of the places where these occur is, of course, slightly more complex, but they are still in the same places of the expression tree corresponding to the formula.

Admittedly, on a sheet of paper, it is more convenient to work with axiom schemata, because we are now already so well trained to pay attention to make no incorrect reasoning steps. But for precision purposes in a formal verification tool it is substantially easier to work with axioms instead, because the uniform substitution mechanism only needs to be understood and implemented once and because the axioms can be implemented by copy&paste. And even when working on a sheet of paper it may be easier to just remember a single uniform substitution mechanism instead of a diverse list of side conditions.

18.3 What Axioms Want

If the axiom of nondeterministic choice $[\cup]$ is internalized as an axiom, not as an axiom schema, then what syntactic elements of differential dynamic logic do the parts of its formula (18.1) correspond to? Suddenly, α and β need to be concrete HPs in the syntax of dL as opposed to schematic variables or placeholders for concrete HPs. Likewise the postcondition P needs to be a concrete dL formula. In fact, revisiting the differential dynamic logic axiom schemata from Chap. 5, there are three different cases of postconditions:

$[:=]$ $[x:=e]p(x) \leftrightarrow p(e)$

$[\cup]$ $[\alpha\cup\beta]P \leftrightarrow [\alpha]P \wedge [\beta]P$

V $p \rightarrow [\alpha]p \quad (FV(p)\cap BV(\alpha)=\emptyset)$

The postcondition p of the vacuous axiom schema V cannot have any variable free that is bound by the HP α. But anything that is not written to by HP α can still be mentioned in p, which is the whole point of this axiom compared to Gödel's generalization proof rule G. In comparison, the postcondition $p(x)$ of the assignment axiom schema $[:=]$ should be allowed to mention variable x despite the fact that it is written to in the HP $x:=e$. That is why the postcondition $p(x)$ mentions x explicitly. The postcondition on the left-hand side of axiom schema $[:=]$ can have the argument x free in the same place that the formula $p(e)$ on the right-hand side of that axiom schema has e. Its postcondition $p(x)$ can still mention other free variables besides x, because no other variable is written to in the discrete assignment $x:=e$. The postcondition P of the axiom schema of nondeterministic choice $[\cup]$, instead, can have any free variables without reservation, because the axiom is correct whether or not the HPs $\alpha\cup\beta$, α, or β modify the values of free variables of P.

Predicate Symbols

Predicate symbols explain all three cases of postconditions with one joint mechanism. The postcondition p in axiom V has a predicate symbol p with 0 arguments, so it has no special permission to have its truth-value depend on any particular free

variables. The postcondition $p(x)$ in axiom [:=] has a predicate symbol p with variable x as its only argument, so its truth-value can depend on the value of x. When reading the postcondition P in axiom [∪] as $p(\bar{x})$ for a predicate symbol p that receives the vector $\bar{x}$ of all variables as argument, so its truth-value can depend on the values of all variables, then all cases of postconditions are covered by corresponding predicate symbols that only differ in the number of arguments.

It is conceptually easier to read the axioms [:=],[∪],V as axioms, so concrete dL formulas, with predicate symbols as postconditions instead of placeholders for formulas. The concrete dL formula $p(x)$ from axiom [:=] literally tells us that its truth-value depends on variable x and apparently nothing else. The formula p from axiom V directly indicates that its truth-value does not depend on the values of any variables. And the case $p(\bar{x})$, which is how we read P in axiom [∪], indicates that its truth-value may depend on the values of all variables. We no longer need to keep in mind what other dL formulas the respective postconditions might stand for, but we see the concrete dL formula explicitly.

Separately, we can then worry about what formulas are acceptable as drop-in replacements for predicate symbols. We can find out which replacements are fine once and for all, and independently of the particular axiom at hand. This separation of concerns is liberating because it enables us to understand the soundness of an axiom via the validity of its dL formula independently from the soundness of the mechanism that generalizes and replaces syntactic elements of the axioms with other concrete dL expressions. For example, the concrete instance (18.2) can be obtained from the concrete dL formula (18.3) of axiom [∪] by the uniform substitution:

$$\sigma = \{\alpha \mapsto x{:=}x+1,\ \beta \mapsto x' = x^2,\ P \mapsto x \geq 0\}$$

This substitution σ substitutes HP $x:=x+1$ for α and HP $x' = x^2$ for β and dL formula $x \geq 0$ for P alias $p(\bar{x})$. Of course, this will require us to better understand the substitution process itself and the rôle of the HPs α and β. But let us first stay on the topic of how to interpret predicate symbols.

Unlike a formula like $x^2 > 5$, which comes with a fixed interpretation of when it is *true*, namely exactly when the square of the value of x exceeds 5, a predicate symbol p does not have a fixed meaning, but is subject to our interpretation. That is what makes it a symbol, because it stands for something. Certainly, a predicate symbol can take different truth-values depending on its argument. So for example, depending on the value of its argument e, the formula $p(e)$ in axiom [:=] will be *true* or *false*. But if two terms e and $\tilde{e}$ evaluate to the same real value, then $p(e)$ and $p(\tilde{e})$ will, of course, either both be *true* everywhere consistently, or both be *false* consistently. Likewise, the predicate symbol p with 0 arguments in axiom V may be either *true* or *false*. But since it does not take any arguments at all, its truth-value does not depend on the values of any variables, so is independent of the state, and will either be *true* consistently everywhere or *false* consistently everywhere. Indeed, if the assumption p of axiom V holds then the arity 0 predicate symbol p is *true*, which makes it *true* everywhere, even after running HP α, because its truth-value

visibly does not depend on the values of any variables. If, instead p is *false*, then the assumption of axiom V is not met, so its implication is trivially *true*.

Function Symbols

Predicate symbols capture the different cases of formulas in dL axioms. Similarly, in the assignment axiom $[:=]$, the term e needs to be a concrete dL term, but one that could take on any value, because that is what a schema variable placeholder in the corresponding axiom schema $[:=]$ would have been able to do. A function symbol with 0 arguments plays that rôle, because function symbols could evaluate to any real value. Just like predicate symbols could evaluate to any truth-value.

The following concrete dL formula can, then, be used as assignment axiom $[:=]$

$$[x:=c()]p(x) \leftrightarrow p(c()) \tag{18.4}$$

with predicate symbol p of arity 1 and function symbol $c()$ of arity 0. For example, the concrete instance

$$[x:=x^2-1]x \geq 0 \leftrightarrow x^2-1 \geq 0 \tag{18.5}$$

can be obtained from (18.4) by the uniform substitution:

$$\sigma = \{c() \mapsto x^2-1,\ p(\cdot) \mapsto (\cdot \geq 0)\} \tag{18.6}$$

This substitution σ substitutes the term x^2-1 for the arity 0 function symbol $c()$ and substitutes the greater-or-equal zero comparison formula for the arity 1 predicate symbol p. To indicate that every occurrence of the predicate symbol p of any argument is affected and substituted with the corresponding ≥ 0 comparison, the substitution substitutes $p(\cdot)$ with a dL formula in which the dot $\cdot$ marks where the argument goes in the resulting dL formula. So for any argument e, the formula $p(e)$ will be replaced with $\sigma(e) \geq 0$. Of course, the substitution σ will also need to be applied to the argument e of $p(e)$, not just to the predicate symbol p, which is why $\sigma(e) \geq 0$ is substituted for $p(e)$ and not just $e \geq 0$. The result of applying the substitution σ to e is denoted $\sigma(e)$ and will be defined properly later.

Program Constant Symbols

Finally, we return to the rôle that the HPs α and β play in axiom $[\cup]$. On the one hand, both need to be concrete HPs for axiom $[\cup]$ to become a concrete dL formula. On the other hand, neither α nor β have a concrete specific behavior, because the axiom $[\cup]$ works for whatever HPs α and β do. Consequently, the HP we use for α and β in axiom $[\cup]$ are what we call *program constant symbols* and can have any arbitrary behavior. Just like predicate symbols do not have a fixed interpretation but might be *true* of any argument, and just like function symbols f do not have

a fixed interpretation but might have any real value as a function of the argument's value, so do program constant symbols not have a fixed interpretation but might have any arbitrary behavior. Depending on its interpretation, a program constant symbol could possibly transition from any initial state to any final state, because its behavior is not described explicitly as it would be in the case of a differential equation or a discrete assignment.

18.4 Differential Dynamic Logic with Interpretations

After having realized what syntactic elements dL axioms need so that they can be faithfully represented as concrete axioms instead of axiom schemata, the first thing we do is officially add those elements into the syntax of differential dynamic logic [6]. Of course, we could have added them right away when introducing hybrid programs in Chap. 3 and differential dynamic logic in Chap. 4, but that would have been a distraction, because we did not need them until now.

18.4.1 Syntax

Differential dynamic logic dL is as usual, except that function symbols, predicate symbols, and program constant symbols are added. Function symbols are usually written f,g,h, predicate symbols p,q,r, and program constant symbols are written a,b,c. Each function and predicate symbol expects a fixed number of terms as arguments, called *arity*. When f is a function symbol f of arity n, then $f(e_1,\ldots,e_n)$ is now also allowed as a term for any terms $e_1,\ldots,e_n$. Likewise, when p is a predicate symbol p of arity n, then $p(e_1,\ldots,e_n)$ is now a formula for any terms $e_1,\ldots,e_n$. But $f(e_1,\ldots,e_{n-1})$ is not a term, because the function symbol f of arity n has not even received sufficiently many arguments. When we have a function that can, say, add two numbers that we pass in as arguments, then we cannot just call this function with 1 argument or with 7 arguments, but need to provide exactly 2.

Function symbols are essentially a more liberal generalization of builtin term operators such as $+$, which has arity 2, is written infix as e_1+e_2 instead of as $+(e_1,e_2)$, and always means addition. Function symbols can have a different number of arguments, but also always expects exactly the same number of arguments as indicated by its arity. Function symbols of arity 0 are also called *constant symbols*, because their value does not depend on any arguments. The use of a function symbol c of arity 0 is sometimes written as $c()$ with empty parentheses for emphasis. In fact, we already allowed rational numbers as constant symbols of arity 0 when originally defining terms. The meaning of a rational number constant is, of course, also fixed. The meaning of the rational number constant 1 is always 1 and the meaning of the rational number constant $1/2$ is always the real number 0.5.

By contrast, function symbols are more general, because they are actually meant as symbols. That is, they do not have a fixed meaning once and for all times, but are symbolic, so their meaning is subject to interpretation. Similarly, predicate symbols are symbols so their meaning depends on our interpretation, and likewise for program constant symbols.

Definition 18.1 (Terms). A *term* e is defined by augmenting the grammar from Definition 2.2 on p. 40 with the following case (where $e_1,\dots,e_n$ are terms and f is a function symbol of arity n):

$$e ::= f(e_1,\dots,e_n) \mid \dots$$

Definition 18.2 (Hybrid program). *Hybrid programs* are defined by augmenting the grammar from Definition 3.1 on p. 71 with the following case (where a is any program constant symbol):

$$\alpha,\beta ::= a \mid \dots$$

Definition 18.3 (dL formula). The *formulas of differential dynamic logic* (dL) are defined by augmenting the grammar from Definition 4.1 on p. 97 with the following case (where $e_1,\dots,e_n$ are terms and p is a predicate symbol of arity n):

$$P ::= p(e_1,\dots,e_n)$$

For emphasis, we might call the resulting logic *differential dynamic logic with interpretations*, but continue to just call it dL, because we just neglected to consider these extensions until now, since they were not necessary for our understanding yet.

This extension of the syntax of dL makes it possible to phrase all axioms we saw before as axioms with concrete dL formulas (instead of as axiom schemata that represent their infinitely many instances subject to side conditions). For example, the axiom schemata we considered as motivating examples above turn into:

[:=] $[x:=c()]p(x) \leftrightarrow p(c())$

[∪] $[a\cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$

V $p \rightarrow [a]p$

18.4.2 Semantics

The semantics of function symbols, predicate symbols, and program constant symbols is actually easy, but it comes with a twist compared to all other definitions of semantics we saw anywhere else in this textbook. The whole point of function symbols, predicate symbols, and program constant symbols is that they are symbolic, so they do not come with a fixed interpretation. Consequently, unlike for the binary $+$

operator, which always means addition, the semantics of a term mentioning a function symbol f of arity 2 depends on how we interpret the symbol f, which may be addition or multiplication or any other reasonable function from two reals to a real.

In order to be able to evaluate to a real number any term in any state, we fix an *interpretation* I which assigns a (sufficiently smooth[3]) n-ary function $I(f):\mathbb{R}^n\to\mathbb{R}$ to every function symbol f of arity n. Given such an interpretation I, we can easily evaluate every term in any state ω just by looking up in I the corresponding function $I(f)$ for every function symbol f in the term.

Definition 18.4 (Semantics of terms). The *value of term* e in state $\omega\in\mathscr{S}$ for interpretation I is a real number denoted $\omega[\![e]\!]$ and is defined by augmenting Definition 2.4 with the following case:

$$\omega[\![f(e_1,\ldots,e_n)]\!] = I(f)\big(\omega[\![e_1]\!],\ldots,\omega[\![e_n]\!]\big) \text{ if } f \text{ is a function symbol of arity } n$$

That is, in state ω, a function symbol application evaluates to the result of the function $I(f)$ applied to the real values $\omega[\![e_i]\!]$ that the respective arguments e_i evaluate to in the state ω.

As predicate symbols have no fixed interpretation either, the interpretation I also assigns an n-ary relation $I(p)\subseteq\mathbb{R}^n$ to every predicate symbol p of arity n. With such an interpretation, the set of states in which a formula is true can be defined easily.

Definition 18.5 (dL semantics). The *semantics* of a dL formula P for interpretation I is the set of states $[\![P]\!]\subseteq\mathscr{S}$ in which P is true, and is defined by augmenting Definition 4.2 with the following case:

12. $[\![p(e_1,\ldots,e_n)]\!] = \big\{\omega : \big(\omega[\![e_1]\!],\ldots,\omega[\![e_n]\!]\big)\in I(p)\big\}$
    That is, a predicate symbol application is true in the set of states ω in which the arguments e_i evaluate to a tuple of real numbers that is in the relation $I(p)$.

Finally, the interpretation I also assigns a reachability relation $I(a)\subseteq\mathscr{S}\times\mathscr{S}$ to every program constant symbol a.

Definition 18.6 (Transition semantics of HPs). Each HP α is interpreted semantically as a binary reachability relation $[\![\alpha]\!]\subseteq\mathscr{S}\times\mathscr{S}$ over states for each interpretation ω, and is defined by augmenting Definition 3.2 with the following case:

7. $[\![a]\!] = I(a)$
   That is, the reachability relation for program constant symbol a is an arbitrary state transition relation determined by the interpretation I.

[3] Functions that are continuously differentiable are smooth enough for our purposes.

18.5 Uniform Substitution

A *uniform substitution* σ substitutes function symbols with terms, predicate symbols with formulas, and program constant symbols with hybrid programs, and it does so uniformly, e.g. it uses the same HP as replacement for program constant symbol b in all places.[4] The result of applying the uniform substitution σ to dL formula ϕ is denoted $\sigma(\phi)$ and will be defined in Sect. 18.5.3. Similarly, $\sigma(\theta)$ denotes the result of applying the uniform substitution σ to term θ and $\sigma(\alpha)$ denotes the result of applying the uniform substitution σ to HP α.

The substitution σ defines a term σf as replacement for each arity 0 function symbol f. The substitution σ also defines a dL formula σp as a replacement for each arity 0 predicate symbol p. It also defines a hybrid program σa for each program constant symbol a. Applying the substitution σ will replace every occurrence of program constant symbol a uniformly by the HP σa and every occurrence of arity 0 function symbol f by σf and every occurrence of arity 0 predicate symbol p by the corresponding replacement σp. For function and predicate symbols with arguments, the reserved function symbol $\cdot$ is used as a placeholder to indicate where the argument goes. For an arity 1 function symbol f, the substitution defines a term whose occurrences of function symbol $\cdot$ indicate where the argument to f is placed. For an arity 1 predicate symbol p, the substitution defines a dL formula whose occurrences of function symbol $\cdot$ indicate where the argument to p goes.

The notation for describing a uniform substitution σ that substitutes term e_1 for arity 1 function symbol f and substitutes term e_2 for arity 0 function symbol c, and that substitutes dL formula ϕ_1 for arity 1 predicate symbol p and substitutes dL formula ϕ_2 for arity 0 predicate symbol q and substitutes hybrid program α for program constant symbol a is:

$$\sigma = \{f(\cdot) \mapsto e_1, c \mapsto e_2, p(\cdot) \mapsto \phi_1, q \mapsto \phi_2, a \mapsto \alpha\} \tag{18.7}$$

The occurrences of reserved arity 0 function symbol $\cdot$ in the term e_1 and the formula ϕ_1, respectively, indicate where the arguments to f and to p, respectively, go in the replacement. We have already seen examples of uniform substitutions in Sect. 18.3. The uniform substitution σ in (18.7) replaces arity 1 function symbol f and predicate symbol p, arity 0 function symbol c and predicate symbol q, and program constant symbol a but leaves all other symbols alone. The *domain* of substitution σ is the set of all symbols it replaces, so $\{f, c, p, q, a\}$ for (18.7).

18.5.1 Uniform Substitution Rule

Church's uniform substitution proof rule US says that the result $\sigma(\phi)$ of applying a uniform substitution σ to a valid formula ϕ is valid, too. Its generalization to differ-

[4] Replacing the same program constant symbol b with different HPs in different places would be very illogical and break all structure there ever was.

ential dynamic logic is sound as well [6]. The intuition is that, if a formula ϕ is valid, so true in all states with any interpretation of its predicate, function, and program constant symbols, then it is also valid after substituting concrete formulas in for its predicate symbols etc., because the predicate symbol very well could have been interpreted to have the same truth-value as its substitute formula. The tricky part is the correct handling of arguments to the predicate symbols and of variables in the replacements, because variables may have different values in different subformulas.

Theorem 18.1 (Uniform substitution). *The proof rule US is sound:*

$$\text{US}\ \frac{\phi}{\sigma(\phi)}$$

So if formula ϕ has a proof, then its uniform substitution instance $\sigma(\phi)$ has a proof, too, just by applying the uniform substitution proof rule US. *The uniform substitution mechanism checks that it does not introduce a free variable into a context in which it is bound in* $\sigma(\phi)$. If the uniform substitution σ applied to ϕ were to introduce a free variable x into a context in which x has been bound, then $\sigma(\phi)$ is not defined, because it *clashes*, and the proof rule US is not applicable to ϕ.

Before proceeding with an exact definition of the uniform substitution mechanism constructing $\sigma(\phi)$ in Sect. 18.5.3, we explore a number of representative examples to gain an intuition for the rôle of rule US in proving.

The formula $(\neg\neg p) \leftrightarrow p$, for example, is valid (in classical logic). When we pick any dL formula ψ, then it will also be valid the formula that results from $(\neg\neg p) \leftrightarrow p$ by uniformly substituting all occurrences of arity 0 predicate symbol p with this formula ψ. For example, the uniform substitution $\sigma = \{p \mapsto [x' = x^2]\, x \geq 0\}$ proves:

$$\text{US}\frac{(\neg\neg p) \leftrightarrow p}{(\neg\neg [x' = x^2]\, x \geq 0) \leftrightarrow [x' = x^2]\, x \geq 0}$$

Any other formula could have been used as a replacement for p (consistently everywhere) as well and rule US would have proved the result from $(\neg\neg p) \leftrightarrow p$.

Substitutions are more subtle when working, e.g., from the formula $(\forall x\, p) \leftrightarrow p$. This formula expresses for an arity 0 predicate symbol p that p is true for all x if and only if p is true in the current state, which makes apparent sense, because the arity 0 predicate symbol p quite visibly does not mention any variables that its truth-value would depend on. In fact, it is precisely this absence of the mention of x that the validity of the formula $(\forall x\, p) \leftrightarrow p$ depends on. We cannot possibly soundly replace p with $x \geq 0$, because that would lead to

$$\text{clash}\lightning\frac{(\forall x\, p) \leftrightarrow p}{\forall x\, (x \geq 0) \leftrightarrow x \geq 0}$$

which is unsound, because not all values of x are nonnegative (left) just because the present value of x is nonnegative (right) in the current state. Indeed, the uniform

substitution mechanism will clash when applying $\sigma = \{p \mapsto x \geq 0\}$ to $(\forall x\, p) \leftrightarrow p$, because σ would introduce the free variable x in the replacement for p into a context $\forall x\, p$ in which x refers to a bound variable, such that the variable x in the replacements for the two occurrences of p would possibly refer to two different values. In fact, the requirement that the replacement for p does not have x as a free variable is quite consistent with our original reason why the premise $(\forall x\, p) \leftrightarrow p$ was valid at all.

Variables other than x can be mentioned free in the replacement for p, though, because they are not bound anywhere where p occurs. For example, the uniform substitution $\sigma = \{p \mapsto y \geq 0\}$ enables rule US to prove:

$$\text{US}\frac{(\forall x\, p) \leftrightarrow p}{\forall x\,(y \geq 0) \leftrightarrow y \geq 0}$$

18.5.2 Examples

The primary, but not the only, use case of the uniform substitution proof rule US is that it makes it possible to instantiate axioms with specific dL formulas. The following examples will, thus, have an axiom as premise, which proves in the dL calculus just by mentioning its name. The primary focus will be on demonstrating how uniform substitutions work, when they clash, and why that is soundness-critical.

How Uniform Substitutions Handle Arguments

Rule US proves, for example, (18.5) from (18.4) with uniform substitution (18.6):

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2-1]x \geq 0 \leftrightarrow x^2-1 \geq 0}$$

Intuitively, this uniform substitution replaces all occurrences of function symbol $c()$ with x^2-1 while also replacing all occurrences of predicate symbol p with a greater-or-equal-to-zero comparison. Of course, in addition to substituting $(\cdot \geq 0)$ for $p(\cdot)$, the uniform substitution is also used on all arguments e of p in any subformula $p(e)$. So σ uniformly replaces every occurrence of $p(e)$ with $\sigma(e) \geq 0$. In particular, σ replaces $p(x)$ with $x \geq 0$ but $p(c())$ with $x^2-1 \geq 0$.

The uniform substitution $\sigma = \{c() \mapsto x^2-1,\ p(\cdot) \mapsto (\cdot \geq x)\}$, instead, clashes for the same formula, because the replacement for $p(\cdot)$ would introduce the free variable x into a context $[x:=x^2-1]\_$ in which x is bound:

$$\text{clash}\lightning\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2-1]x \geq x \leftrightarrow x^2-1 \geq x} \tag{18.8}$$

It is crucial for soundness that this substitution clashes, because the premise is valid (axiom [:=]) but the conclusion is not, because the postcondition $x \geq x$ of the assignment is valid, but the right-hand side $x^2-1 \geq x$ is not. This makes sense, because all free occurrences of x in the postcondition are affected by the assignment $x := x^2-1$, so the substitution $\{p(\cdot) \mapsto (\cdot \geq x)\}$ did not select all occurrences of x for the $\cdot$ placeholder. In contrast, the uniform substitution $\sigma = \{c() \mapsto x^2-1,\ p(\cdot) \mapsto (\cdot \geq \cdot)\}$ gives the perfectly acceptable result:

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2-1]x \geq x \leftrightarrow x^2-1 \geq x^2-1}$$

Likewise, the uniform substitution $\sigma = \{c() \mapsto x^2-1,\ p(\cdot) \mapsto (2(\cdot) \geq \cdot)\}$ results in:

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2-1]2x \geq x \leftrightarrow 2(x^2-1) \geq x^2-1}$$

In comparison, the uniform substitution $\sigma = \{c() \mapsto x^2-1,\ p(\cdot) \mapsto (\cdot \geq y)\}$ is acceptable, because, even if the replacement for $p(\cdot)$ introduces the free variable y, it only introduces it into the context $[x:=x^2-1]\_$ in which y is not bound either:

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2-1]x \geq y \leftrightarrow x^2-1 \geq y}$$

Observe how the explicit argument x in the subformula $p(x)$ of the premise makes it possible for the substitute $x \geq y$ to mention x instead of placeholder $\cdot$ in its replacement $(\cdot \geq y)$. But as (18.8) demonstrated, even such a mention of x as an argument does not give license to use variable x anywhere else in the replacements. Of course, the argument x in $p(x)$ only indicates an explicit license for a possible dependence on x, not that $p(x)$ has to depend on x. For example, this uniform substitution $\sigma = \{c() \mapsto 2x+1,\ p(\cdot) \mapsto (y^2 \geq y)\}$ does not use the $\cdot$ argument placeholder:

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=2x+1]y^2 \geq y \leftrightarrow y^2 \geq y}$$

Uniform substitutions can also have predicates in which the argument placeholder $\cdot$ appears in more deeply nested positions. For example, the uniform substitution $\sigma = \{c() \mapsto x^2,\ p(\cdot) \mapsto [(y:=\cdot+y)^*](\cdot \geq y)\}$ is acceptable, because it does not introduce any free variables into a context into which they are bound:

$$\text{US}\frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=x^2][(y:=x+y)^*](x \geq y) \leftrightarrow [(y:=x^2+y)^*](x^2 \geq y)}$$

How Uniform Substitutions Handle Constant Predicate Symbols

But without the original formula mentioning x as an argument in $p(x)$, the uniform substitution cannot use x in a context in which x is bound. For example, this uniform substitution $\sigma = \{a \mapsto x' = 5,\ p \mapsto (x \leq 5)\}$ clashes, because the replacement for p introduces the free variable x into the context $[x' = 5]\_$ in which x is bound, which is the context that results from applying σ to $[a]p$:

$$\text{clash}\lightning \frac{p \rightarrow [a]p}{x \leq 5 \rightarrow [x' = 5]x \leq 5}$$

It is crucial for soundness that this substitution clashes, because the premise is valid (axiom V), but the conclusion is not, because x does not stay below 5 forever when following the differential equation $x' = 5$. This is precisely what the side condition of axiom schema V from Lemma 5.11 would have prevented. In contrast, the uniform substitution $\sigma = \{a \mapsto x' = 5,\ p \mapsto (y \leq 5)\}$ works fine, because it only introduces free variable y into the resulting context $[x' = 5]\_$ in which y is not bound anyhow:

$$\text{US}\frac{p \rightarrow [a]p}{y \leq 5 \rightarrow [x' = 5]y \leq 5}$$

The uniform substitution $\sigma = \{a \mapsto (v := v + 1; \{x' = v, v' = -b\}),\ p \mapsto (y \leq b)\}$ works fine, because its function and predicate symbols only introduce free variables y and b into a context in which they are possibly read but never written:

$$\text{US}\frac{p \rightarrow [a]p}{y \leq b \rightarrow [v := v + 1; \{x' = v, v' = -b\}]y \leq b}$$

Telling the respective good and bad cases of axiom instantiation attempts apart is what uniform substitutions achieve without having to provide any side conditions that are specific to the formula or axiom schema at hand. Uniform substitutions provide a uniform answer, once and for all, to the question which instantiations of formulas are sound because they preserve validity.

How Uniform Substitutions Handle Program Constant Symbols

When working from the assignment axiom $[:=]$ with its postcondition $p(x)$ or from the vacuous axiom V with its postcondition p, the uniform substitution rule US needs to check for capture of other variables, which is important for soundness. When working from the nondeterministic choice axiom $[\cup]$ with its postcondition $p(\bar{x})$, instead, then this postcondition has explicit permission to mention all variables $\bar{x}$, such that any dL formula can be accepted as replacement. The uniform substitution $\sigma = \{a \mapsto v := -cv,\ b \mapsto x'' = -g,\ p(\bar{x}) \mapsto 2gx \leq 2gH - v^2\}$ yields:

$$\text{US}\frac{[a\cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x})\wedge[b]p(\bar{x})}{[v:=\text{-}cv\cup x''=\text{-}g]2gx\leq 2gH-v^2 \leftrightarrow [v:=\text{-}cv]2gx\leq 2gH-v^2\wedge[x''=\text{-}g]2gx\leq 2gH-v^2}$$

As usual, $x''=-g$ is short for $\{x'=v,v'=-g\}$.

18.5.3 Uniform Substitution Application

A uniform substitution can replace any number of such function, predicate, or program constant symbols simultaneously. The notation $\sigma f(\cdot)$ denotes the replacement for $f(\cdot)$ according to σ, i.e. the value $\sigma f(\cdot)$ of function σ at $f(\cdot)$. By contrast, $\sigma(\phi)$ denotes the result of applying σ to ϕ which we defined now (likewise for $\sigma(\theta)$ and $\sigma(\alpha)$). The notation $f\in\sigma$ signifies that σ replaces function symbol f, i.e. $\sigma f(\cdot)\neq f(\cdot)$, so f is in the domain of σ. Likewise, the notation $p\in\sigma$ signifies that σ replaces predicate symbol p, and correspondingly $a\in\sigma$ means that σ replaces program constant symbol a.

$\sigma(x)$	$=x$	for variable $x\in\mathscr{V}$
$\sigma(f(e))$	$=(\sigma(f))(\sigma(e))\overset{\text{def}}{=}\{\cdot\mapsto\sigma(e)\}(\sigma f(\cdot))$	for function symbol $f\in\sigma$
$\sigma(g(e))$	$=g(\sigma(e))$	for function symbol $g\notin\sigma$
$\sigma(e+\tilde{e})$	$=\sigma(e)+\sigma(\tilde{e})$	
$\sigma(e\cdot\tilde{e})$	$=\sigma(e)\cdot\sigma(\tilde{e})$	
$\sigma((e)')$	$=(\sigma(e))'$	if σ is $\mathscr{V}$-admissible for e
$\sigma(e\geq\tilde{e})$	$\equiv\sigma(e)\geq\sigma(\tilde{e})$	likewise for $>,=,<,\leq$
$\sigma(p(e))$	$\equiv(\sigma(p))(\sigma(e))\overset{\text{def}}{\equiv}\{\cdot\mapsto\sigma(e)\}(\sigma p(\cdot))$	for predicate symbol $p\in\sigma$
$\sigma(q(e))$	$\equiv q(\sigma(e))$	for predicate symbol $q\notin\sigma$
$\sigma(\neg\phi)$	$\equiv\neg\sigma(\phi)$	
$\sigma(\phi\wedge\psi)$	$\equiv\sigma(\phi)\wedge\sigma(\psi)$	likewise for $\vee,\rightarrow,\leftrightarrow$
$\sigma(\forall x\,\phi)$	$\equiv\forall x\,\sigma(\phi)$	if σ is $\{x\}$-admissible for ϕ
$\sigma(\exists x\,\phi)$	$\equiv\exists x\,\sigma(\phi)$	if σ is $\{x\}$-admissible for ϕ
$\sigma([\alpha]\phi)$	$\equiv[\sigma(\alpha)]\sigma(\phi)$	if σ is $\mathrm{BV}(\sigma(\alpha))$-admissible for ϕ
$\sigma(\langle\alpha\rangle\phi)$	$\equiv\langle\sigma(\alpha)\rangle\sigma(\phi)$	if σ is $\mathrm{BV}(\sigma(\alpha))$-admissible for ϕ
$\sigma(a)$	$\equiv\sigma a$	for program constant symbol $a\in\sigma$
$\sigma(b)$	$\equiv b$	for program constant symbol $b\notin\sigma$
$\sigma(x:=e)$	$\equiv x:=\sigma(e)$	
$\sigma(x'=e\,\&\,Q)$	$\equiv x'=\sigma(e)\,\&\,\sigma(Q)$	if σ is $\{x,x'\}$-admissible for e,Q
$\sigma(?Q)$	$\equiv ?\sigma(Q)$	
$\sigma(\alpha\cup\beta)$	$\equiv\sigma(\alpha)\cup\sigma(\beta)$	
$\sigma(\alpha;\beta)$	$\equiv\sigma(\alpha);\sigma(\beta)$	if σ is $\mathrm{BV}(\sigma(\alpha))$-admissible for β
$\sigma(\alpha^*)$	$\equiv(\sigma(\alpha))^*$	if σ is $\mathrm{BV}(\sigma(\alpha))$-admissible for α

Fig. 18.1 Recursive application of uniform substitution σ

Figure 18.5.3 defines the result $\sigma(\phi)$ of applying to a dL formula ϕ the *uniform substitution* σ that uniformly replaces all occurrences of a function f by a term (instantiated with its respective argument of f) and all occurrences of a predicate p by

a formula (instantiated with its argument) as well as of a program constant symbol a by a program. Each case in Fig. 18.5.3 applies the uniform substitution recursively.[5] In each case, the uniform substitution application mechanism checks that the substitution is admissible for the bound variables of the operator, i.e. σ will not introduce free variables into the scope of an operator in which they are bound (which will be defined in Definition 18.7 below).

For example, for the case $\sigma(\forall x\,\phi)$, the bound variables that σ needs to be admissible for ϕ are $\{x\}$, because if σ were to introduce free variable x while forming $\sigma(\phi)$, then x would be incorrectly captured by quantifier $\forall x$. Suppose the substitution σ were to replace an arity 0 predicate symbol p that occurs in ϕ by the formula $x \geq 0$, then within the scope of the quantifier of $\forall x\,\phi$, this formula $x \geq 0$ suddenly refers to a different variable called x, namely the one bound by the universal quantifier $\forall x$ and no longer the free variable x. That is why such a uniform substitution is not defined, because it is not admissible. This is crucial for soundness, e.g., for the formula $p \leftrightarrow \forall x\,p$, because the substitution would otherwise replace p inconsistently with the same formula $x \geq 0$ but referring to different values of x in different places, since one of the newly introduced occurrences of x in the resulting $x \geq 0 \leftrightarrow \forall x(x \geq 0)$ is in the scope of a quantifier binding x. In the case of a modal formula $\sigma([\alpha]\phi)$, the bound variables that are taboo and cannot be introduced as free variables when forming the substituted postcondition $\sigma(\phi)$ are the bound variables $\mathrm{BV}(\sigma(\alpha))$ of the substituted HP $\sigma(\alpha)$. In the case of a differential equation $\sigma(x' = e\,\&\,Q)$, the bound variables $\{x, x'\}$ are taboo and cannot be introduced as free variables when forming $\sigma(e)$ or $\sigma(Q)$, because the differential equation changes the values of both.

Arguments are put in for the placeholder $\cdot$ recursively by uniform substitution $\{\cdot \mapsto \sigma(\theta)\}$ in Fig. 18.5.3, which is defined since it replaces the placeholder function symbol $\cdot$ of arity 0 by the readily substituted argument $\sigma(\theta)$.

Definition 18.7 (Admissible uniform substitution). A uniform substitution σ is *U-admissible for formula ϕ* (or term θ or HP α, respectively) with respect to the variables $U \subseteq \mathcal{V}$ iff $\mathrm{FV}(\sigma|_{\Sigma(\phi)}) \cap U = \emptyset$, where $\sigma|_{\Sigma(\phi)}$ is the restriction of substitution σ that only replaces symbols that occur in ϕ, and $\mathrm{FV}(\sigma) = \bigcup_{f \in \sigma} \mathrm{FV}(\sigma f(\cdot)) \cup \bigcup_{p \in \sigma} \mathrm{FV}(\sigma p(\cdot))$ are the *free variables* that σ introduces for function or predicate symbols.

A uniform substitution σ is *admissible for* ϕ (or θ or α, respectively) iff the bound variables U of each operator of ϕ are not free in the substitution on its arguments, i.e. σ is U-admissible. These admissibility conditions are listed explicitly in Fig. 18.5.3, which defines the result $\sigma(\phi)$ of applying σ to ϕ. For each case in Fig. 18.5.3, the taboo set U whose U-admissibility is required of σ is exactly the set of variables that are bound by its top-level operator.

The substitution σ is said to *clash* and its result $\sigma(\phi)$ (or $\sigma(\theta)$ or $\sigma(\alpha)$) is not defined if σ is not admissible, in which case rule US is not applicable either.

[5] This makes the uniform substitution a *homomorphism*, because the substitution of an addition is the addition of substitutions: $\sigma(e + \tilde{e}) = \sigma(e) + \sigma(\tilde{e})$ and accordingly for all other operators.

Note that the free variables $\mathrm{FV}(\sigma)$ of a substitution σ are only defined as the union of the free variables of the replacements for its function symbols f and predicate symbols p, not the program constant symbols, because programs may already read the full state and change it to a new state. Likewise, replacements of predicate symbols $p(\bar{x})$ with all variables $\bar{x}$ as arguments are disregarded in the free variable determination, because they apparently already have explicit permission to depend on the values of all variables and, thus, do not introduce any new free variables.

Finally observe that σ is already U-admissible for formula ϕ if the sufficient condition $\mathrm{FV}(\sigma) \cap U = \emptyset$ holds. The only reason for Definition 18.7 to restrict the admissibility check to the restriction $\sigma|_{\Sigma(\phi)}$ of the substitution to the symbols that actually occur in the affected formula ϕ is that there is no need for the substitution to clash if σ introduces free variables for function or predicate symbols that do not even occur in ϕ. For example, $\sigma = \{p(\cdot) \mapsto (\cdot \leq y), q \mapsto (x \leq 5)\}$ is $\{x\}$-admissible for $\phi \overset{\text{def}}{=} (x > 2 \land p(y))$, because the dangerous predicate symbol q with its free variable x that would not be $\{x\}$-admissible does not even occur in ϕ, so the substitution is restricted to $\sigma|_{\Sigma(\phi)} = \{p(\cdot) \mapsto (\cdot \leq y)\}$ whose free variables are just y. Neither the original substitution σ nor its restriction $\sigma|_{\Sigma(\phi)}$ are $\{y\}$-admissible for ϕ, because both have y as a free variable. The original substitution σ also would not be $\{x\}$-admissible for $\psi \overset{\text{def}}{\equiv} (x > 2 \land p(y) \land q)$, because its replacement for the predicate symbol q that occurs in ψ has x as a free variable.

This uniform substitution $\sigma = \{a \mapsto x' = 5,\ p \mapsto (y \leq 5)\}$ succeeds:

$$\text{US}\frac{p \to [a]p}{y \leq 5 \to [x' = 5]\, y \leq 5}$$

It uses the uniform substitution mechanism in Fig. 18.5.3 and also $y \notin \mathrm{BV}(x' = 5)$:

$$\begin{aligned}&\sigma(p \to [a]p) \equiv \sigma(p) \to \sigma([a]p) \equiv \sigma(p) \to [\sigma(a)]\sigma(p)\\ \equiv\ &\sigma p \to [\sigma a]\sigma p \equiv y \leq 5 \to [x' = 5]\, y \leq 5\end{aligned}$$

In addition to the previous examples, we consider a few more. The uniform substitution $\sigma = \{p(\cdot) \mapsto (\cdot \geq 0),\ q \mapsto (y < 0)\}$ works fine, because it only introduces free variable y into the context $\forall x\_$ in which y is not bound:

$$\text{US}\frac{\forall x\,(p(x) \lor q) \leftrightarrow (\forall x\, p(x)) \lor q}{\forall x\,(x \geq 0 \lor y < 0) \leftrightarrow (\forall x\,(x \geq 0)) \lor y < 0}$$

The application of the uniform substitution mechanism according to Fig. 18.5.3 is straightforward:

$$\begin{aligned}&\sigma(\forall x\,(p(x) \lor q) \leftrightarrow (\forall x\, p(x)) \lor q) \equiv \sigma(\forall x\,(p(x) \lor q)) \leftrightarrow \sigma((\forall x\, p(x)) \lor q)\\ \equiv\ &\forall x\,(\sigma(p(x) \lor q)) \leftrightarrow \sigma(\forall x\, p(x)) \lor \sigma(q) \equiv \forall x\,(\sigma(p(x)) \lor \sigma(q)) \leftrightarrow \forall x\, \sigma(p(x)) \lor \sigma(q)\\ \equiv\ &\forall x\,(x \geq 0 \lor y < 0) \leftrightarrow (\forall x\,(x \geq 0)) \lor y < 0\end{aligned}$$

This substitution uses that x is not free in the replacement for q.

In contrast, the uniform substitution $\sigma = \{p(\cdot) \mapsto (\cdot \geq 0),\ q \mapsto (x < 0)\}$ clashes, because its replacement for q introduces free variable x into a context $\forall x\_$ in which it is bound:

$$\text{clash}\lightning \frac{\forall x\,(p(x) \lor q) \leftrightarrow (\forall x\, p(x)) \lor q}{\forall x\,(x \geq 0 \lor x < 0) \leftrightarrow (\forall x\,(x \geq 0)) \lor x < 0}$$

This is soundness-critical, because the left formula is valid (every number is either greater-or-equal or smaller than 0) but the right formula is not, because it is equivalent to $x < 0$, which imposes a condition on the present value of x. The uniform substitution application $\sigma(\forall x\,(p(x) \lor q))$ clashes, because σ is not $\{x\}$-admissible for $p(x) \lor q$ on account of the replacement $x < 0$ for q having free variable x which would be bound by the $\forall x$ quantifier. Of course, this makes sense, because a disjunction can only be pulled outside the scope of a quantifier if it does not use the quantified variable. That is precisely what the premise expresses. Indeed, the premise can be proved from other quantifier axioms.

Observe that it is crucial for soundness that even an occurrence of $p(x)$ in a context where x is bound does not permit mentioning free variable x in the replacement except in the places of the $\cdot$ placeholder. For example, uniform substitution $\sigma = \{c() \mapsto 0,\ p(\cdot) \mapsto (\cdot \geq x)\}$ clashes when used on the assignment axiom $[:=]$, because the replacement for $p(\cdot)$ would introduce the extra free variable x into a context $[x:=0]\_$ in which x is bound:

$$\text{clash}\lightning \frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=0]x \geq x \leftrightarrow 0 \geq x}$$

The premise is valid (axiom $[:=]$) but the conclusion is not, because the postcondition $x \geq x$ of the assignment is valid, but the right-hand side $0 \geq x$ is not. The reason is that the free variable x in the replacement $(\cdot \geq x)$ for $p(\cdot)$ would refer to the variable bound by $x:=0$ in the substitute for $p(x)$ but would refer to a free variable x in the substitute for $p(c())$.

Uniform substitutions need to pay attention when substituting in the argument. For example, $\sigma = \{c() \mapsto y^2,\ p(\cdot) \mapsto [(y:=\cdot+y)^*](\cdot \geq y)\}$ clashes when applied to the assignment axiom $[:=]$ while substituting the replacement y^2 for $c()$ for the argument placeholder $\cdot$ in the replacement for $p(c())$, since that would introduce free variable y into a context $[(y:=\cdot+y)^*](\cdot \geq y)$ where it is bound:

$$\text{clash}\lightning \frac{[x:=c()]p(x) \leftrightarrow p(c())}{[x:=y^2][(y:=x+y)^*](x \geq y) \leftrightarrow [(y:=y^2+y)^*](y^2 \geq y)}$$

This is, of course, crucial for soundness because the left loop always adds to y the same value in each round (square of initial value of y) while the right loop would always add to y the square of the most recent value of y.

18.5.4 Uniform Substitution Lemmas

18.5.5 Soundness

18.6 Static Semantics

Modular interface between prover and logic
- Recall semantic static semantics
- Augment analysis for new cases

18.6.1 Correctness

18.7 Axiomatic Proof Calculus for dL

A purely axiomatic formulation of the differential dynamic logic axiomatization [6] is shown in Fig. 18.2. The axioms listed in Fig. 18.2 are axioms, so concrete dL formulas, and not axiom schemata that stand for an infinite collection of formulas. Besides the cases we already discussed so far, these axioms are formed by using program constant symbols a and b as HPS and by using $p(\bar{x})$ as a concrete formula for the postconditions that have no admissibility requirement.

$[:=]$ $[x:=c()]p(x) \leftrightarrow p(c())$

$[?]$ $[?q]p \leftrightarrow (q \rightarrow p)$

$[\cup]$ $[a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x})$

$[;]$ $[a;b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x})$

$[^*]$ $[a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a][a^*]p(\bar{x})$

$\langle\cdot\rangle$ $\langle a\rangle p(\bar{x}) \leftrightarrow \neg[a]\neg p(\bar{x})$

K $[a](p(\bar{x}) \rightarrow q(\bar{x})) \rightarrow ([a]p(\bar{x}) \rightarrow [a]q(\bar{x}))$

I $[a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a^*](p(\bar{x}) \rightarrow [a]p(\bar{x}))$

V $p \rightarrow [a]p$

G $\dfrac{p(\bar{x})}{[a]p(\bar{x})}$

$\forall$ $\dfrac{p(x)}{\forall x\, p(x)}$

MP $\dfrac{p \rightarrow q \quad p}{q}$

Fig. 18.2 Differential dynamic logic axioms and proof rules

The only exception is the test axiom [?], which could have been phrased as either of the following two dL formulas:

$$[?q]p \leftrightarrow (q \rightarrow p) \tag{18.9}$$

$$[?q(\bar{x})]p(\bar{x}) \leftrightarrow (q(\bar{x}) \rightarrow p(\bar{x})) \tag{18.10}$$

It looks as if the second formulation (18.10) would be more flexible, because its explicit mention of the list of all variables in $p(\bar{x})$ and $q(\bar{x})$ make it obvious that the axiom can be instantiated with any arbitrary dL formulas for the test $?q(\bar{x})$ and postcondition $p(\bar{x})$. However, the first formulation (18.9) is sufficient, because any arbitrary dL formulas can be substituted in for the arity 0 predicate symbols p and q as well, because no variables are bound anywhere in (18.10), so its intersection with any arbitrary set of free variables of any substitution will always be empty.

Soundness of the axioms and proof rules in Fig. 18.2 follows from soundness of the corresponding axiom schemata and proof rule schemata in Chap. 5 and Chap. 7 from Part I of this textbook. The concrete axioms in Fig. 18.2 are instances of the previous axiom schemata, even if their soundness would have been easier to prove directly [6]. Implementing the axioms of Fig. 18.2 in a theorem prover is now straightforward, because each axiom is just a single concrete dL formula that the prover needs to remember. It can be shown that the uniform substitution proof rule US can prove all instances of these axioms that are required for completeness [6].

18.8 Differential Axioms

A purely axiomatic of the differential equation axioms and axioms for differentials of dL [6] is shown in Fig. 18.3. These axioms are special instances of the axiom schemata from Part II, which explains their soundness.

18.9 Formula Symbols Applied to Formulas

A predicate symbol p of arity n applies to n terms $e_1, \ldots, e_n$ to form the formula $p(e_1, \ldots, e_n)$. Even if we only really need them for one neat but optional purpose, we, nevertheless, also add a new kind of symbolic formula that takes another formula as an argument instead of taking terms as arguments. For systematic reasons, these symbols C that take formulas as arguments are called *quantifier symbols* $C(P)$.

18.10 Summary

See [6]

DW $[x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow [x' = f(x)\,\&\,q(x)](q(x) \to p(x))$

DI $\big([x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow [?q(x)]p(x)\big) \leftarrow (Q \to [x' = f(x)\,\&\,q(x))](p(x))')$

DC $\big([x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow [x' = f(x)\,\&\,q(x) \wedge r(x)]p(x)\big) \leftarrow [x' = f(x)\,\&\,q(x)]r(x)$

DE $[x' = f(x)\,\&\,q(x)]p(\bar{x}) \leftrightarrow [x' = f(x)\,\&\,q(x)][x' := f(x)]p(\bar{x})$

DG $[x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow \exists y\,[x' = f(x), y' = a(x)\cdot y + b(x)\,\&\,q(x)]p(x)$

DS $[x' = c()\,\&\,q(x)]p(x) \leftrightarrow \forall t{\geq}0\,\big((\forall 0{\leq}s{\leq}t\; q(x + c()s)) \to [x := x + c()t]p(x)\big)$

$+'$ $(f(\bar{x}) + g(\bar{x}))' = (f(\bar{x}))' + (g(\bar{x}))'$

$-'$ $(f(\bar{x}) - g(\bar{x}))' = (f(\bar{x}))' - (g(\bar{x}))'$

$\cdot'$ $(f(\bar{x}) \cdot g(\bar{x}))' = (f(\bar{x}))' \cdot g(\bar{x}) + f(\bar{x}) \cdot (g(\bar{x}))'$

$/'$ $(f(\bar{x})/g(\bar{x}))' = \big((f(\bar{x}))' \cdot g(\bar{x}) - f(\bar{x}) \cdot (g(\bar{x}))'\big)/g(\bar{x})^2$

c' $(c())' = 0$ (for numbers or constants $c()$)

x' $(x)' = x'$ (for variable $x \in \mathcal{V}$)

Fig. 18.3 Differential equation axioms and differential axioms

References

1. Church, A. *Introduction to Mathematical Logic, Volume I* (Princeton University Press, Princeton, NJ, 1956).
2. (eds Felty, A. & Middeldorp, A.) *International Conference on Automated Deduction, CADE'15, Berlin, Germany, Proceedings* **9195** (Springer, 2015).
3. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M. & Platzer, A. *KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems* in *CADE* (eds Felty, A. & Middeldorp, A.) **9195** (Springer, 2015), 527–538. doi:10.1007/978-3-319-21401-6_36.
4. Platzer, A. Differential Dynamic Logic for Hybrid Systems. *J. Autom. Reas.* **41,** 143–189 (2008).
5. Platzer, A. *A Uniform Substitution Calculus for Differential Dynamic Logic* in *CADE* (eds Felty, A. & Middeldorp, A.) **9195** (Springer, 2015), 467–481. doi:10.1007/978-3-319-21401-6_32.
6. Platzer, A. A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *J. Autom. Reas.* doi:10.1007/s10817-016-9385-1 (2016).
7. Platzer, A. & Quesel, J.-D. *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems.* in *IJCAR* (eds Armando, A., Baumgartner, P. & Dowek, G.) **5195** (Springer, 2008), 171–178. doi:10.1007/978-3-540-71070-7_15.