Compile-Time Extensions to Hybrid ODEs

Yingfu Zeng
Rice University, Texas, USA
yz39@rice.edu

Ferenc Bartha
Rice University, Texas, USA
Ferenc.A.Bartha@rice.edu

Walid Taha
Halmstad University, Sweden
Walid.taha@hh.se

Reachability analysis for hybrid systems is an active area of development and has resulted in many promising prototype tools. Most of these tools allow users to express hybrid system as automata with a set of ordinary differential equations (ODEs) associated with each state, as well as rules for transitions between states. Significant effort goes into developing and verifying and correctly implementing those tools. As such, it is desirable to expand the scope of applicability tools of such as far as possible. With this goal, we show how compile-time transformations can be used to extend the basic hybrid ODE formalism traditionally supported in hybrid reachability tools such as SpaceEx or Flow*. The extension supports certain types of partial derivatives and equational constraints. These extensions allow users to express, among other things, the Euler-Lagrangian equation, and to capture practically relevant constraints that arise naturally in mechanical systems. Achieving this level of expressiveness requires using a binding time-analysis (BTA), program differentiation, symbolic Gaussian elimination, and abstract interpretation using interval analysis. Except for BTA, the other components are either readily available or can be easily added to most reachability tools. The paper therefore focuses on presenting both the declarative and algorithmic specifications for the BTA phase, and establishes the soundness of the algorithmic specifications with respect to the declarative one.

1 Introduction

Reachability analysis for hybrid systems [2] is an active area of development and has resulted in many promising prototype tools. Prominent examples of such tools include CHARON [3], HyTech [18], PHAVer [13], dReach [22], dReal [15], SpaceEx [14], and Flow*[7]. Most of these tools allow users to express hybrid systems as automata with a set of ordinary differential equations (ODEs) associated with each state, as well as rules for transitions between states. In particular, ODEs must be in the explicit form where the left hand side of an equality has to be the derivative of a state variable. Significant effort goes into verifying and correctly implementing those tools. As such, it is desirable to expand the scope of applicability tools of such as far as possible.

1.1 Contributions

With this goal, we present a systematic method to translate an expressive language with partial derivatives and equations to a standard language supporting ODEs, guards, and reset maps. The method can be used to extend reachability analysis tool such as SpaceEx or Flow*. An experimental implementation of the proposed technique is available in the freely available, open source Acumen language implementation [1]. Examples illustrating the use of these extension can be found in the directory `examples/04_Experimental/04_BTA`. Since both partial derivatives and equations are eliminated completely after the compile-time transformation, the user benefits from the added expressivity but the underlying tools do not need to change. The two extensions allow the user to express, among

E. Ábrahám and S. Bogomolov (Eds.): 3rd International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR 2017)
EPTCS 247, 2017, pp. 52–70, doi:10.4204/EPTCS.247.5

other things, the Euler-Lagrangian equation, and to capture practically relevant constraints that arise naturally in mechanical systems. Achieving this level of expressivity requires using a binding time-analysis (BTA) [20, 17, 8, 24], program differentiation, symbolic Gaussian elimination, and abstract interpretation using interval analysis. Except for BTA, the other components are either readily available or can be easily added. The technical part of the paper therefore focuses on presenting both the declarative and algorithmic specifications for the BTA phase, and establishes the soundness of the algorithmic with respect to the declarative.

After reviewing related work on compile-time extensions (Section 2), we introduce the syntax and type system for a core differential equation language (Section 3). Then, we present a declarative specification of binding-time analysis (BTA) and a big step semantics for specialization (Section 4), along with a formal proof of type safety (Theorem 1). We then present an algorithmic specification of the BTA that works by first generating a set of constraints and then attempting to solve them (Section 5), and we show that this algorithmic specification is faithful to the declarative BTA (Lemma 12) and always produces a unique minimum solution that maps as much of the code as possible to static if an assignment exists (Theorem 2). To illustrate the practical value of the formalism, we present two case studies that have been carried out using the implementation (Appendix A).

2 Related Work

Binding-time Analysis (BTA) is a static analysis traditionally supported in the offline partial evaluation of general purposes languages. It works by identifying a two-level structure in the program being analyzed, where the first level is a computation that can be done at "partial evaluation time" ("compile time" in our case), and the second level must be left as a "residual" that is executed at runtime. BTA has generally been studied for general purpose languages. In our setting, we study it in the context of Domain Specific Languages (DSLs) [19, 23, 30] intended for modeling hybrid systems. It should also be noted that our primary purpose is to use it for extending expressivity. Partial evaluation, in contrast, is only concerned with improving the runtime performance of programs. In what follows, we elaborate on these key points.

A key idea in the work we present in this paper is that there are powerful techniques from the programming languages community that can help make reachability tools more broadly applicable. To do this, this work uses two-level languages in a novel way. To put the existing related work in context, it is useful to consider several characteristics relating to the language considered and the transformation used, namely, whether the language is domain-specific (or general purpose), whether it supports equations, whether the transformation is done at compile-time (or runtime), whether the tool performs let-insertion (to avoid code duplication), whether the language is statically typed, and whether the tool provide accurate source level error reporting. The systems that we will consider are partial evaluation systems for C, namely, C-mix [16] and Tempo [10]; template instantiation mechanisms, namely, C++ Templates [11] and Template Haskell [27]; multi-stage programming languages, namely, MetaOCaml [11] and LMS [25]; the Verilog Preprocessor [26]; and the hybridization technique [4].

Table 1 provides an overview of how these different systems related to these key properties. The main observations from the table are as follows. Almost all tools are compile-time (except MetaOCaml), and almost all are statically checked (except C++ Templates). A key feature of static checking is that it facilitates accurate source-level reporting. That is primary reason for choosing an approach based on BTA or some type of static analysis. Compile-time program specializers, such as C-Mix and Tempo focus on automatically specializing a program through a well understood set of transformations to produce a program that is faster than the original one. There are no fundamental reasons why specialization (and

Table 1: Comparison of Compile-time Transformation

	Static checking	Source level reporting	Compile-time	Domain-specific	Let insertion	Equations
C-Mix	Yes	-	Yes	No	Yes	No
Tempo	Yes	Yes	Yes	No	Yes	No
C++ Template	No	No	Yes	No	-	No
Template Haskell	Yes	Yes	Yes	No	-	No
MetaOcaml	Yes	Yes	No	No	No	No
LMS	Yes	-	No	No	Yes	No
Verilog preprocessor	Yes	Yes	Yes	Yes	-	No
Hybridization	Yes	-	Yes	Yes	-	No
This paper	Yes	Yes	Yes	Yes	Yes	Yes

two-level languages) need to be limited to general purpose languages. In fact, as this paper shows, they can be quite useful as they can be used to increase expressivity. Let-insertion was invented in the partial evaluation community, and is adopted by automated tools by LMS (but not be explicit tools like MetaOCaml). It is quite critical when there are significant compile-time computations, as is the case when we are trying to eliminate non-trivial constructs like partial derivatives or performing substantial manipulations to turn equations into formulae. However, none of these works address the question of supporting equations, that is, allowing the user to write constraints in equational form, and then translating them directly into "formula" form for directed evaluation. Our work is comparable to that of HyST [5], which is a tool that aims to facilitate interchange of models between different tools. This way, HyST facilitates sharing of models and comparing solvers. In contrast, our work explores another dimension for reuse, namely, how these tools can be extended to support a more flexible and expressive modeling formalism.

3 A Differential Equation Language (Acumen-17)

Fig. 1 introduces the syntax and type system for a core differential equation language called Acumen-17. We use the following notational conventions:

- Writing $\langle e_j \rangle^{j \in 1...m}$ denotes a vector $\langle e_1, e_2, ..., e_m \rangle$. We will occasionally omit the superscript $j \in \{1...m\}$ and write $\langle e_j \rangle$ when the range of j is clear from context.
- Writing $\{e_j\}^{j \in 1...m}$ denotes a set $\{e_1, e_2, ..., e_m\}$, and we write $A \uplus B$ for $A \cup B$ when we require that $A \cap B = \emptyset$.

The set Names is a finite countable set of names, and we use n to denote elements of this set. We use i to range over natural numbers, q to range over rationals, and t to range over booleans.

Similarly, we introduce the natural number i drawn from the set of natural numbers $\mathbb{N}$, rational q from rational number set $\mathbb{Q}$ and lastly boolean values t from $\{\mathsf{true}, \mathsf{false}\}$, denoted by $\mathbb{B}$. Variables are either a name n or a name followed by a number of primes ($'$). Type terms represent naturals, reals, Booleans, and products, respectively. A type environment is a partial function from variables to type terms. We treat environments as graphs of functions or as functions.

Syntax

$$n \in \text{Names}, \quad i \in \mathbb{N}, \quad q \in \mathbb{Q} \quad \text{and} \quad t \in \mathbb{B}$$

Constant	k	::=	$i \mid q \mid t$
Variable	x	::=	$n \mid x'$
Type	τ	::=	$\mathsf{nat} \mid \mathsf{bool} \mid \mathsf{real} \mid \prod_{j \in 1..m} \tau_j$
Type Environment	Γ	=	$\{x_j : \tau_j\}^{j \in 1...m}$ such that whenever $x_i = x_j$ then $i = j$
Function	f	::=	$+ \mid - \mid \times \mid / \mid \hat{} \mid \&\& \mid \mid\mid \mid > \mid >= \mid == \mid != \mid \sin \mid \cos$
Expression	e	::=	$k \mid x \mid \langle e_j \rangle^{j \in 1...m} \mid e_1(e_2) \mid f(e) \mid \frac{d}{dt} e \mid \frac{\partial}{\partial e_2} e_1$
Equation	s	::=	$x = e \mid e_1 = e_2,\ e_1 \not\equiv x \mid x^+ = e \mid \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \mid \forall n \in e.\ s \mid \{s_j\}^{j \in 1...m}$

$\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma \vdash i : \mathsf{nat}} \quad \frac{}{\Gamma \vdash q : \mathsf{real}} \quad \frac{}{\Gamma \vdash t : \mathsf{bool}} \quad \frac{n : \tau \in \Gamma}{\Gamma \vdash n : \tau} \quad \frac{x' : \mathsf{real} \in \Gamma \quad x : \mathsf{real} \in \Gamma}{\Gamma \vdash x' : \mathsf{real}} \quad \frac{\Gamma \vdash e_j : \tau_j}{\Gamma \vdash \langle e_j \rangle^{j \in 1...m} : \prod_{j \in 1..m} \tau_j}$$

$$\frac{\Gamma \vdash e_1 : \prod_{j \in 1..m} \tau_j \quad \Gamma \vdash i : \mathsf{nat} \quad i < m}{\Gamma \vdash e_1(i) : \tau_i} \quad \frac{\Gamma \vdash e_1 : \prod_{j \in 1..m} \tau \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash e_1(e_2) : \tau} \quad \frac{\Gamma \vdash e : \prod_{j \in 1..m_f} \tau_{f,j} \quad f : \prod_{j \in 1..m_f} \tau_{f,j} \to \tau}{\Gamma \vdash f(e) : \tau}$$

$$\frac{\Gamma \vdash e : \mathsf{real}}{\Gamma \vdash \frac{d}{dt} e : \mathsf{real}} \quad \frac{\Gamma \vdash e_1 : \mathsf{real} \quad \Gamma \vdash e_2 : \mathsf{real}}{\Gamma \vdash \frac{\partial}{\partial e_2} e_1 : \mathsf{real}}$$

$\boxed{\Gamma \vdash s}$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 = e_2} \quad \frac{\Gamma \vdash x : \tau \quad \Gamma \vdash e : \tau}{\Gamma \vdash x^+ = e} \quad \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2} \quad \frac{\Gamma \vdash e : \prod_{j \in 1..m} \tau \quad \Gamma, n : \tau \vdash s}{\Gamma \vdash \forall n \in e\ .\ s} \quad \frac{\Gamma \vdash s_j}{\Gamma \vdash \{s_j\}^{j \in 1...m}}$$

Figure 1: Syntax and Type system for Acumen-17

Function names f are drawn from a fixed set containing basic operators. Expressions include constants, variables, vector expressions, vector indexing, function application, time derivatives, and partial derivatives. Derivatives can be applied to both expressions and variables. The time derivative on a variable, for example x'', has special status, in that it can both be used in expressions to mean the value of the derivative at a given time and can also be equated to a value. When there is a constraint that equates a time derivative of a variable to a value, the effect is that integration is used to compute the value of the variable itself. In principle, in an equational language, if a symbolic expression for the variable is known, the derivative variable can be determined from that expression. In practice, it is generally rare that a closed form expression for the result of a simulation is known. Instead, it is more common to have the value of the derivative known, and then numerical integration is used to compute the value of the variable itself. The partial derivative ($\frac{\partial}{\partial e_2} e_1$) is an operator that takes two expressions and returns the result of the first expression differentiated with respect to the second expression. The ASCII-based syntax is `expr'[expr]`. For two scalars, the result is simply the first expression partially differentiated with the second one. If one expression is a scalar, and the other a vector, the operator is applied component-wise. Allowing arbitrary expressions e_2 instead of just variables in partial derivatives, allows us to express things like the Euler-Lagrange equation directly.

The first type of equations is a continuous equation. In processing such equations, we distinguish between two cases, one where the left hand side is a variable, and the other when the left hand side is not a variable. This will be used to illustrate that the formalism is able to accommodate languages where equations need not be directed. The second type of equation is the discrete assignment. "x is reset to e". Discrete assignments are essential for modeling hybrid systems, where instantaneous changes of a value (resets) can occur in juxtaposition to continuous dynamics. The third type of equations is a conditional

equation, which allows us to express the choice between which of two equations holds depending on the boolean condition given as an expression. The fourth kind of equations is a universal quantification, and it provides a concise way of describing the dynamics of a system that has a family of state variables. The variable introduced by this construct may only be an unprimed name. The last construct is a set of equations $\{s_j\}^{j\in 1\ldots m}$.

3.1 Type System

A Acumen-17expression e has type τ under environment Γ when the judgement $\Gamma \vdash e : \tau$ is derivable according to the rules presented in Fig. 1. The rules for natural, real, and boolean constants are straightforward. The rule for unprimed names is simple environment lookup. The rule for primed variables, however, requires that both primed and unprimed variables have type real. The rule for vector construction is also straightforward. Vector indexing is a bit more interesting, as it treats the case when the index is a literal as a special case, allowing elements to have different types. This makes it possible to use vectors both for tuples and for (homogeneous) vectors. Function applications assume that we have a function n_f that determines the arity of the function f, and a function $\tau_{f,j}$ that determines the type of the jth argument to the function. Partial derivatives have straightforward rules. The rules for equations are straightforward. Finally, environment extension of environment Γ_1 with the binding $x : \tau$, written $\Gamma_1, x : \tau$ is an environment Γ_2 defined as follows:

$$\Gamma_2(y) = \begin{cases} \tau & \text{if } y = x, \\ \Gamma_1(y) & \textit{otherwise}. \end{cases}$$

3.2 Example: A Lagrangian Model

For a variety of technical reasons, researchers working on novel robotic systems tend to make extensive use of the Lagrangian method. It is especially useful in the case when the system being described has more than one state variable. Then modeling using Lagrange employs families of equations, which are written as one equation but really represent a collection of different equations derived by instantiating certain indices. Figure 2(a) provides the Acumen-17 model of a second order nonlinear system shown in Figure 3. It consists of a pendulum hanging from a mass, which is attached through a spring to a wall.

As the system has two degrees of freedom, x and θ, the example introduces a vector of state variables q. The Euler-Lagrange equation that appears at the end of the example is expressed by the family of equations. In Figure 2(a), the $\forall$ quantifier is used to introduce the index variable for a family of equations. In the ASCII-based syntax, the keyword `foreach` represent this quantifier. The intent is to express as concisely and as close to what would typically appear in a mechanics textbook:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \quad \text{where} \quad \begin{array}{l} q = (x, \theta), \\ i \in \{1, 2\}. \end{array}$$

This notation generally has a syntactic interpretation, that is, the *name* contained in the ith element of the vector is looked up. In other words, this family of equations literally represents the following two

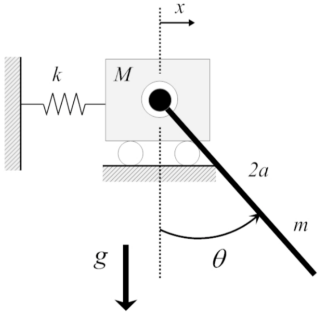


Figure 3: A Pendulum Spring-Mass system.

Figure 2: Compiling Pendulum/Mass Example

(a) Latex-style Acumen-17 Source for Pendulum/Mass Example

$$q = (x,\theta), a = 1, m = 2, M = 5, g = 9.8,$$
$$k = 2, I = \tfrac{4}{3}ma^2, L = T - V,$$
$$T = \tfrac{1}{2}(M+m)\dot{x}^2 + ma\dot{x}\dot{\theta}\cos(\theta) + \tfrac{2}{3}ma^2\dot{\theta}^2$$
$$V = \tfrac{1}{2}kx^2 + mga(1-\cos(\theta)),$$
$$\forall i \in \{1...|q|)\}, \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial \dot{q}_i}\right) - \tfrac{\partial L}{\partial q_i} = 0$$

(b) After Binding-Time Analysis (BTA)

$$q = (x,\theta), a = 1, ...I = \tfrac{4}{3}ma^2$$
$$\forall i \in \{1...|q|)\}, \tfrac{d}{dt}\left(\tfrac{\partial L}{\partial\, \dot{q}_i}\right) - \tfrac{\partial L}{\partial q_i} = 0$$

(c) After Specialization (Implicit ODEs)

$$q = (x,\theta), a = 1, ...I = \tfrac{8}{3}$$
$$2\cos(\theta)\ddot{\theta} - 2\sin(\theta)\dot{\theta}^2 + 7\ddot{x} + 2x = 0$$
$$\tfrac{98}{5}\sin(\theta) + 2\cos(\theta)\ddot{x} + \tfrac{8}{3}\ddot{\theta} = 0$$

(d) After Symbolic Gaussian Elimination (Explicit ODEs)

$$A = \sin(\theta), B = \cos(\theta),$$
$$\ddot{x} = \tfrac{2(A\dot{\theta}^2 - x) - B\ddot{\theta}}{7}$$
$$\ddot{\theta} = \tfrac{\frac{-686}{5}A - 4B(A\dot{\theta}^2 - x)}{\frac{56}{3} - 4B^2}$$

equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = 0 \quad \text{and} \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0$$

The offline partial evaluation strategy enables us to support family of equations and partial differentiation by utilizing the two most important components, namely the binding-time analysis (BTA) and specialization. A successful BTA annotates the model with instructions for performing certain part of the computation early and other part for later processing. The annotated model for the pendulum/spring example is illustrated in Fig. 2(b). In this illustration, computations that remain for further processing are shaded grey, whereas computations can be performed immediately in the next specialization phase appear in a white background. The value of a is marked known or static as it is a value, and the BTA also annotates variable I known for that both m and a are known variables. A more interesting case is the indexing operator $q(i)$. Although the state variable vector q being marked unknown, in fact we need to solve for the values of state variables x and θ in the simulation, we can still perform this operation statically for the reason that the size of q and the index variable i are known.

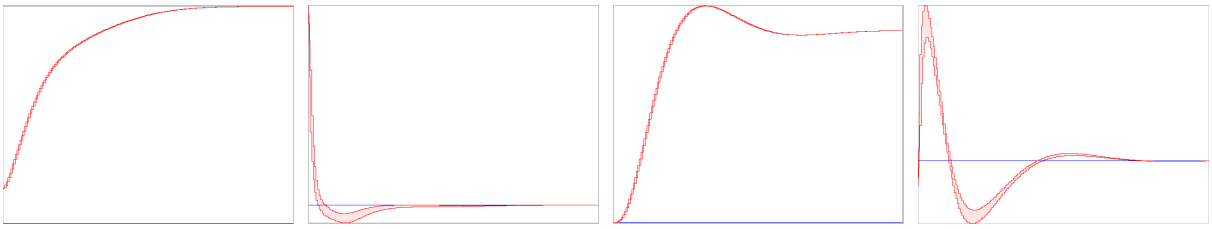
The step which performs the work that a BTA schedules is called specialization. The result of specializing our running example is presented in Fig. 2(c). Computing the value of I is simple rational arithmetic. The instantiation of a family of equations is essentially a type of iteration, which also replaces q_i by x and θ by vector lookup. In the same time, symbolic time and partial differentiation are performed using the chain rule. Solving multiple implicit ODEs to explicit form equations is achieved using an analog of symbolic Gaussian elimination. For our running example, the result of this step is presented in Fig. 2(d). Abstract interpretation with interval analysis is used to ensure the pivot expression

is non zero. To control the system, for example, stabilizing the position and the angular displacement, one can add two PD controllers. The modification to the original model in Fig. 2(a) are as follows:

$$ux = 100 * (2 - x) + 30 * (0 - \dot{x}), ut = 100 * (\pi - \theta) + 40 * (0 - \dot{\theta}),$$

$$u = (ut, ux), \quad \forall i \in \{1...|q|)\}, \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = u(i)$$

The Acumen implementation supports an enclosure simulation semantics that produces rigorous over-approximations (guaranteed upper and lower bounds) for all simulations [12]. Previously, this implementation only supported a formalism that worked with hybrid ODEs. With the work we presented here, this implementation can now handle models such as the pendulum spring mass model presented above. The plot of controlled system are as follows:



(a) θ (b) $\ddot{\theta}$ (c) x (d) $\ddot{x}$

3.3 A Cam and Follower Example

We further demonstrate the expressiveness of the proposed language using the following two case studies. Transforming rotational motion into any other motions is often conveniently accomplished by means of a *cam mechanism*. A *cam* is defined as a machine element having a curved outline, which by its rotation motion, gives a predetermined motion to another element, which is often called *follower*. Fig .4(a) shows such a *cam mechanism*, the curved outline of cam r is a function of rotational angle θ, defined as below:

$$r = (1.5 - \frac{cos\theta}{2}) * (1 + \frac{cos(2\theta)}{5})$$

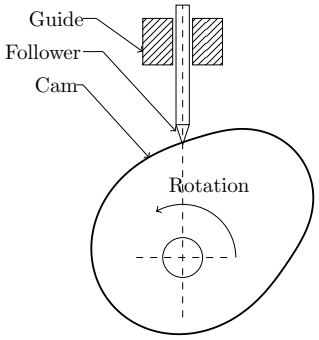
In the study of various aspects of the follower motion, the velocity and acceleration of the follower are needed. To get the correct form, the modeler usually has to manually derive the partial derivatives. Fig. 8 in the Appendix shows the mathematical model and the corresponding Acumen-17program. Clearly, supporting partial derivatives in the language greatly simplifies the modeling task, and can save the modeler much tedious and error-prone work.

3.4 A Compass Gait Biped Example

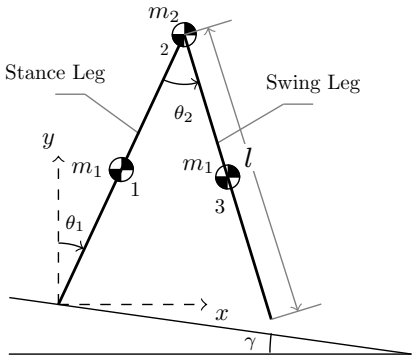
The Compass gait biped model [9, 29] is a two dimensional unactuated rigid body system placed on a downward surface inclined at a fixed angle γ from the horizontal plane. A diagram of the model is shown in Fig .4(c) with its physical parameters. The configuration of this two-link mechanism can be described by the generalized coordinates $q = [\theta_1, \theta_2]$, where θ_1 is the angle from the vertical line to the *stance leg* and θ_2 is the angle between two legs. It is a hybrid model featuring two phases. At the start of each

Figure 4: Two Case Studies

(a) Cross section of A Cam and Follower



(b) A Compass Gait Biped [6]



step, the system is governed by its continuous dynamics until the *swing leg* hits the ground. The discrete event can be modeled as an inelastic collision conserving angular momentum. The stance and swing legs switch instantaneously during the collision and go into the next step after.

3.4.1 Continuous Dynamics and Discrete Event

The continuous phase of this system can be modeled using the same Lagrange method shown earlier. Let point (x_i, y_i) denote the position of centralized masses shown in Fig .4(b), form which its easy to define the kinetic and potential energy of the system. Applying the same Lagrange equation shown in Equation 1 with $q = (\theta_1, \theta_2)$, we have the dynamic equations of the system during the swing phase. The perpendicular distance from the walking surface to the tip of the *swing leg* is given by

$$guard = lsin\gamma(sin\theta_1 + sin(\theta_2 - \theta_1))$$

Where γ is the slope of the ground. Impact occurs when the tip of the swing leg hits the walking surface in a downward direction, which can be describe as follows: $guard \leq 0 \wedge \dot{guard} < 0$. Using conservation of angular momentum [6], the explicit solution of post impact velocities can be determined. Fig. 8 in the Appendix shows the mathematical model and the full Acumen-17model. This example shows the proposed formalism can support a direct mapping from mathematical model to simulation code for a hybrid system model with complex dynamics.

4 BTA and Specialization for Acumen-17

This section presents a declarative specification of the binding-time analysis (BTA) and specialization process for Acumen-17, and proves the correctness (type-safety) of the BTA with respect to the specialization process.

$$
\begin{array}{lllll}
\text{Binding Time} & b & ::= & \mathsf{S} \mid \mathsf{D} \\
\text{Expression} & e^b & ::= & k^b \mid x^b \mid (\langle e_j^{b_j} \rangle^{j\in 1..m})^b \mid e_1^b(e_2^b)^b \mid f(e^b)^b \mid (\frac{d}{dt}e^b)^b \mid (\frac{\partial}{\partial e_2^{b_2}} e_1^{b_1})^b \\
\text{Equation} & s^b & ::= & (x^b = e^b)^b \mid (x^{+b} = e^b)^b \mid (e_1^b = e_2^b)^b,\ e_1 \not\equiv x \\
 & & & \mid (\mathsf{if}\ e^b\ \mathsf{then}\ s_1^b\ \mathsf{else}\ s_2^b)^b \mid (\forall n^b \in e^b.\ s^b)^b \mid (\{s_j^{b_j}\}^{j\in 1..m})^b \\
\text{Binding Time type} & \tau^b & ::= & \mathsf{nat}^b \mid \mathsf{bool}^b \mid \mathsf{real}^b \mid (\prod_{j\in 1..m} \tau_j^{b_j})^b \\
\text{Binding Time Environment} & \Gamma^b & = & \{x_j : \tau_j{}^{b_j}\}^{j\in 1..m} \text{ and } x_i = x_j \implies i = j
\end{array}
$$

$$
\frac{}{\Gamma^b \vdash i^b : \mathsf{nat}^b} \quad \frac{}{\Gamma^b \vdash q^b : \mathsf{real}^b} \quad \frac{}{\Gamma^b \vdash t^b : \mathsf{bool}^b} \quad \frac{n^b : \tau^b \in \Gamma^b}{\Gamma^b \vdash n^b : \tau^b} \quad \frac{x'^b : \mathsf{real}^b \in \Gamma^b \quad x^{\mathsf{D}} : \mathsf{real}^{\mathsf{D}} \in \Gamma^b}{\Gamma^b \vdash x'^b : \mathsf{real}^b}
$$

$$
\frac{\Gamma^b \vdash e_j^{b_j} : \tau_j^{b_j} \quad b = \sqcup b_j}{\Gamma^b \vdash (\langle e_j^{b_j} \rangle^{j\in 1..m})^b : (\prod_{j\in 1..m} \tau_j^{b_j})^b} \quad \frac{\Gamma^b \vdash e_1^b : (\prod_{j\in 1..m} \tau_j^{b_j})^b \quad \Gamma^b \vdash i^{\mathsf{S}} : \mathsf{nat}^{\mathsf{S}} \quad i < m}{\Gamma^b \vdash e_1^b(i^{\mathsf{S}})^{b_i} : \tau_i^{b_i}} \quad \frac{\Gamma^b \vdash e_1^{b_1} : (\prod_{j\in 1..m} \tau^{b_1})^{b_1} \quad \Gamma^b \vdash e_2^{b_2} : \mathsf{nat}^{b_2} \quad b = b_1 \sqcup b_2}{\Gamma^b \vdash e_1^{b_1}(e_2^{b_2})^b : \tau^b}
$$

$$
\frac{f : \prod_{j\in 1..m_f} \tau_{f,j} \to \tau \quad \Gamma^b \vdash e^b : (\prod_{j\in 1..m_f} \tau_{f,j}^{b_j})^b}{\Gamma^b \vdash f(e^b)^b : \tau^b} \quad \frac{\Gamma^b \vdash e^b : \mathsf{real}^b}{\Gamma^b \vdash (\frac{d}{dt}(e^b))^b : \mathsf{real}^b} \quad \frac{\Gamma^b \vdash e_1 : \mathsf{real}^{b_1} \quad \Gamma^b \vdash e_2 : \mathsf{real}^{b_2} \quad b = b_1 \sqcup b_2}{\Gamma^b \vdash (\frac{\partial}{\partial e_2^{b_2}} e_1^{b_1})^b : \mathsf{real}^b}
$$

$$
\frac{x^b : \tau^b \in \Gamma^b \quad \Gamma^b \vdash e^b : \tau^b}{\Gamma^b \vdash (x^b = e^b)^b} \quad \frac{\Gamma^b \vdash : x : \tau^{b_1} \quad \Gamma^b \vdash e : \tau^{b_2}}{\Gamma^b \vdash (x^{b_1+} = e^{b_2})^{b_2}} \quad \frac{\Gamma^b \vdash e_1^{b_1} : \tau^{b_1} \quad \Gamma^b \vdash e_2^{b_2} : \tau^{b_2} \quad b = b_1 \sqcup b_2}{\Gamma^b \vdash (e_1^{b_1} = e_2^{b_2})^b}
$$

$$
\frac{\Gamma^b \vdash e : \mathsf{bool}^b \quad \Gamma^b \vdash : s_1^{b_1} \quad \Gamma^b \vdash s_2^{b_2} \quad b_3 = b \sqcup b_1 \sqcup b_2}{\Gamma^b \vdash (\mathsf{if}\ e^b\ \mathsf{then}\ s_1^{b_1}\ \mathsf{else}\ s_2^{b_2})^{b_3}} \quad \frac{\Gamma^b \vdash e : (\prod_{j\in 1..m} \tau^{b_j})^{b_1} \quad \Gamma^b, n : \tau^{b_1} \vdash s^{b_2}}{\Gamma^b \vdash (\forall n \in e^{b_1}.\ s^{b_2})^{b_2}} \quad \frac{\Gamma^b \vdash s_j^{b_j} \quad b = \sqcup b_j}{\Gamma^b \vdash (\{s_j^{b_j}\}^{j\in 1..m})^b}
$$

Figure 5: Binding Time Analysis for Acumen-17

4.1 BTA

BTA is the analysis performed in an offline partial evaluation system to determine, given some early or "static" inputs to a program, which of the program's computation can be done at an early stage [20]. Fig. 5 gives a declarative specification of the BTA. There are two binding times S and D, representing "static" and "dynamic" computations, respectively. Static is for compile-time computations that are done before the simulation starts, and dynamic is for computations done during the simulation proper. Expressions, equations, types, and type environments are all annotated with binding times.

The changes to the derivation rules are largely straightforward. Essentially, binding times are propagated with types. In addition, when multiple subexpressions occur, their binding times are combined using the least upper bound operator $\sqcup$ which returns static only if all arguments are static, otherwise returns dynamic. However, for vector indexing, when the index expression is static and the subexpressions dynamic, we will still perform the look up operation. Finally, the rule for primed variable requires the unprimed variable with the same name to be dynamic. And in the rule for vector indexing with a literal, where the literal is annotated as static, the binding time of the expression is the same as the corresponding entry.

4.2 Specialization

Fig. 9 in Appendix presents the big-step semantics for the specialization process. Values include constant with static annotation, dynamic expression and vector of values. Normal form equations are straightforward, with the absence of universal quantification equation. The first auxiliary function is used to compute static function application. The last two are for eliminating total and partial derivatives using the chain rule. Function *FV* returns free variables in an expression and function *LV* extracts the left hand side variable of a directed equation.

Relation $\hookrightarrow_e$ essentially specializes all subexpressions to values then combines them according to their binding times. Function application with static binding time returns the evaluation result of the corresponding function. Vector indexing with static index performs look up operation statically, even if the vector itself has dynamic binding time. Total and partial derivatives get eliminated statically using different rules depending on the what their subexpressions specialized to. The rules for relation $\hookrightarrow_s$ are similar. However, they all require that the equation to be specialized does not contain free variables that are defined in the equations following it. For directed equations, in addition to specializing the right hand side expression, the rule also substitutes the result into the rest of the equations. Both relations can also generate err terms, which will be propagated to top level for error reporting. For example, the static index may be specialized to a natural number that is bigger than the size of the vector. However, one type of error we do not catch is the case of partial derivative $\frac{\partial}{\partial e_2^{b_2}} e_1^{b_1}$, when $e_2^{b_2}$ can not be specialized to a variable x^{b_2}. It is analogous to the traditional division by zero error.

4.3 Type Safety

Definition 1. The erasure relation $|\cdot|$ for e^b, s^b and Γ^b is defined as follows:

$$|e^b| = e \quad |s^b| = s \quad |\Gamma^b|(x) = \tau \text{ if } \Gamma(x) = \tau$$

Lemma 1 (Erasure preserves typablity)**.** $\forall \Gamma^b, e^b, s^b$

$$\Gamma^b \vdash e^b : \tau^b \Rightarrow |\Gamma^b| \vdash |e| : \tau$$

$$\Gamma^b \vdash s^b \Rightarrow |\Gamma^b| \vdash |s|$$

Proof. By induction on the derivation of $\Gamma^b \vdash e^b : \tau^b$ and $\Gamma^b \vdash s^b$, respectively. □

Lemma 2. Substitution type preservation $\forall \Gamma^b$,
x, e, s, τ.

$$\begin{array}{l} \Gamma^b \vdash v^{b_1} : \tau^{b_1} \ \wedge \ \Gamma^b, x : \tau^{b_1} \vdash e : \tau^{b_2} \\ \qquad \Longrightarrow \ \Gamma^b \vdash e[x := v^{b_1}]^{b_2} : \tau^{b_2} \\ \Gamma^b \vdash v^{b_1} : \tau^{b_1} \ \wedge \ \Gamma^b, x : \tau^{b_1} \vdash s^{b_2} \\ \qquad \Longrightarrow \ \Gamma^b \vdash s[x := v^{b_1}]^{b_2} \end{array}$$

Proof. By induction on the derivation of $\Gamma^b \vdash e^b : \tau^b$ and $\Gamma^b \vdash s^b$ respectively. □

Lemma 3 (Type Preservation)**.** $\forall \Gamma, \Gamma^b, e, s, \tau$.

$$\Gamma^b \vdash e^b : \tau^b \wedge e^b \hookrightarrow_e v^b \implies \Gamma^b \vdash v^b : \tau$$

$$\Gamma^b \vdash s^b \wedge s^b \hookrightarrow_s w^b \implies \Gamma^b \vdash w^b$$

Lemma 4. Static value $\forall \Gamma^b, e$.

$$\Gamma^b \vdash v^{\mathsf{S}} : \mathsf{nat}^{\mathsf{S}} \implies |v^{\mathsf{S}}| = i$$

$$\Gamma^b \vdash v^{\mathsf{S}} : \mathsf{bool}^{\mathsf{S}} \implies |v^{\mathsf{S}}| = t$$

$$\Gamma^b \vdash v^{\mathsf{S}} : \mathsf{real}^{\mathsf{S}} \implies |v^{\mathsf{S}}| = q$$

Proof. When the binding time of a value v^b is static, by the definition, v can only be a constant or a vector of constants. And by typing rules in Fig. 5, we can prove the lemma above. □

Theorem 1. Type safety of specialization Let Γ^{D} denote $\{x_j : \tau^{\mathsf{D}}\}^{j \in 1 \ldots m}$ and $\forall \Gamma^{\mathsf{D}}, e, s, b, v^b, w^b$

$$\Gamma^{\mathsf{D}} \vdash e : \tau^b \wedge e^b \hookrightarrow_e r \implies r \neq \mathsf{err} \wedge \Gamma^{\mathsf{D}} \vdash r : \tau^b$$

$$\Gamma^{\mathsf{D}} \vdash s^b \wedge s^b \hookrightarrow_s r \implies r \neq \mathsf{err} \wedge \Gamma^{\mathsf{D}} \vdash r$$

Due to the page limits, the proof can be found in Section 1 of the supplementary document.

5 Algorithmic Specification

This section presents an algorithmic specification of the BTA. First we introduce the constraint type and the constraint generation function. We proceed by providing a normalization method that guarantees to find the unique *minimal solution*, if one exists. At last, we show that specification is faithful to the declarative BTA.

5.1 Generating Binding Time Constraints

Fig. 6 defines Label l, which can be root or a label indexed by a natural number. For example, let the label of equation $x = 1$ be l, and the label for x and e be l_1 and l_2 respectively. A Binding Time Expression B can be static, dynamic or a label. A constraint c is a partial order $\sqsubseteq$ between two binding time expressions. Constraint is satisfied in three cases $\mathsf{S} \sqsubseteq \mathsf{D}, \mathsf{S} \sqsubseteq \mathsf{S}$ and $\mathsf{D} \sqsubseteq \mathsf{D}$.

5.2 Control Flow and Control Scope

Because of conditional equations, a variable may have a different binding time depending on where it appears. For example:

```
x = 1, if t < 5 then y = x  else y' = x
```

The value of x is statically known for both branches but value of y is only static in the first branch. To handle the control scope issue, we build an auxiliary *global environment* while labeling the program.

Definition 2. A total map π : Variable $\rightarrow$ Label $\cup$ D is called *local environment*. □

Definition 3. A map $\rho : l \rightarrow \pi$ is called *global environment*. To look up the defining label of a variable x inside scope l_0, we first find the corresponding *local environment* π, then apply variable x in it. That is to say $\rho(l_0)(x)$, we abbreviate it to $\rho(l_0, x)$ in the rest of this section. □

Let $\Delta : s \to \rho$ be a construct function that takes a equation s and returns a *global environment* defined as follows, assuming the label of each equation is l, the scope is l_0 and starting global environment be ρ:

$$\Delta(l_0, \{s_j\}^{j\in 1\ldots m}, \rho) = \Delta(l_0, \{s_j\}^{j\in 2\ldots m}, \Delta(l_0, s_1, \rho))$$

$$\Delta(l_0, \{x = e\}_l, \rho) = \rho \uplus (l_0 \to \rho(l_0) \uplus (x \to l_1))$$

$$\Delta(l_0, \{\mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\}_l, \rho) = \rho \uplus \Delta(l_2, s_1, \rho) \uplus \Delta(l_3, s_2, \rho)$$

When constructing a *global environment*, the starting scope label l is root, and changed to the label of branches when inductively constructing inside a conditional equation. The example above has the following *global environment*:

$$\{\mathsf{root} \to \{x \to \mathsf{root}_{11}\}, \mathsf{root}_{22} \to \{x \to \mathsf{root}_{11}, y \to \mathsf{root}_{2211},\} \\ \mathsf{root}_{23} \to \{x \to \mathsf{root}_{11}, y' \to \mathsf{root}_{2311}\}\}$$

Definition 4. $C = \{c_j\}^{j\in 1\ldots m}$ is called a *constraint set*. The *labels* of C are defined as

$$Labels(C) = \{l \mid \exists B \text{ st } l \sqsubseteq B \in C \text{ or } B \sqsubseteq l \in C\}.$$

C is in *normal form* if for all $c \in C$, one of the following is true:

$$c \equiv \mathsf{S} \sqsubseteq l,\ c \equiv l \sqsubseteq \mathsf{D},\ c \equiv l \sqsubseteq \tilde{l}$$

and is of *error form* if $\mathsf{D} \sqsubseteq \mathsf{S} \ \in C$. The sets of C in normal form and of error are denoted by NF and $Error$, respectively. □

Fig. 6 defines function $[\![\cdot]\!]$ that takes e in scope l_0 and *global environment* ρ returns a constraint set. It generates a constraint $l \sqsubseteq \mathsf{S}$ for constant k. For an unprimed variable n, it generates a constraint between the occurrent label l and the definition label in the *global environment* $\rho(l_0, n)$. For variable with primes x', additional constraints between D and labels of all its lower derivatives in the *global environment* are added. For $\langle e_j \rangle$, $e_1(e_2)$,$f(\langle e_j \rangle)$,$\frac{d}{dt}(e)$ and $\frac{\partial}{\partial e_2} e_1$, we generate constraints between label of e and all its subexpressions e_i, that is $\bigcup_i \{l_i \sqsubseteq l\}$. Then inductively apply $[\![\cdot]\!]$ to all the subexpressions.
Function $[\![\cdot]\!]$ for equation is defined in a similar way. In the case of directed equation $n = e$ and $x' = e$, the binding time of corresponding definition label depends on right hand side expression e, captured by $l_1 \sqsubseteq \rho(l_0, n)$ and $l_1 \sqsubseteq \rho(l_0, x')$ respectively. For $x^+ = e$, $e_1 = e_2$ and $\{s_j\}$, the binding time depends on subexpressions and sub-constraints. In the case for if e then s_1 else s_2, it changes the scope label from l_0 to l_2 and l_3 when inductively apply $[\![\cdot]\!]$ to s_1 and s_2. In the case of $\forall n \in e,\ s$, it adds a new mapping from binding variable into the *global environment* ρ when inductively generate constraint from s.

Definition 5. A map σ: Label $\to$ Binding time is called a *substitution*, and σ is the identity function on labels not in domain of σ. As Label is finite, $dom_\sigma = \{l_1, ..., l_n\}$. Thus, σ may be described by $[l_1 \mapsto \sigma(l_1), \ldots, l_n \mapsto \sigma(l_n)]$.
Given two substitutions σ and $\tilde{\sigma}$, the *extension* of σ with $\tilde{\sigma}$ is defined as $\sigma \rhd \tilde{\sigma}$: $dom_\sigma \cup dom_{\tilde{\sigma}} \to$ Binding time such that

$$\sigma \rhd \tilde{\sigma}(l) \quad = \begin{cases} \sigma(l) & \text{if } l \in dom_\sigma, \\ \tilde{\sigma}(l) & \text{otherwise.} \end{cases}$$

Syntax

$$\begin{array}{rrcl} \text{Label} & l & := & \mathsf{root} \mid l_i \text{ where } i \in \mathbb{N} \\ \text{Binding Time Expression} & B & := & b \mid l \\ \text{Constraint} & c & := & B \sqsubseteq B \\ \text{Satisfied Constraint} & \vdash c & := & \vdash \mathsf{S} \sqsubseteq \mathsf{D} \mid \vdash \mathsf{S} \sqsubseteq \mathsf{S} \mid \vdash \mathsf{D} \sqsubseteq \mathsf{D} \end{array}$$

Constraint Generation Function

Input e	Output $[\![e]\!]_{l_0,l,\rho}$
k	$\{l \sqsubseteq \mathsf{S}\}$
n	$\{\rho(l_0,n) \sqsubseteq l,\ l \sqsubseteq \rho(l_0,n)\}$
x'	$[\![x]\!]_{l_0,l_1,\rho} \cup \{\rho(l_0,x') \sqsubseteq l, \mathsf{D} \sqsubseteq l_1\}$
$\langle e_j \rangle^{j \in 1\ldots m}$	$\bigcup_{j\in 1\ldots m}[\![e_j]\!]_{l_0,l_j,\rho} \bigcup_{j\in 1\ldots m}\{l_j \sqsubseteq l\}$
$e_1(e_2)$	$[\![e_1]\!]_{l_0,l_1,\rho} \cup [\![e_2]\!]_{l_0,l_2,\rho}$
	$\cup\{l_1 \sqsubseteq l,\ l_2 \sqsubseteq l\}$
$f(e)$	$[\![e]\!]_{l_0,l_1,\rho} \cup \{l_1 \sqsubseteq l\}$
$\frac{d}{dt}e$	$[\![e]\!]_{l_0,l_1,\rho} \cup \{l_1 \sqsubseteq l\}$
$\frac{\partial}{\partial e_2}e_1$	$[\![e_1]\!]_{l_0,l_1,\rho} \cup [\![e_2]\!]_{l_0,l_2,\rho}$
	$\cup\{l_1 \sqsubseteq l,\ l_2 \sqsubseteq l\}$

Input s	Output $[\![s]\!]_{l_0,l,\rho}$
$x = e$	$[\![x]\!]_{l_0,l_1,\rho} \cup [\![e]\!]_{l_0,l_2,\rho}$
	$\cup\{l_1 \sqsubseteq l,\ l_2 \sqsubseteq l,\ l_2 \sqsubseteq l_1\}$
$x^+ = e$	$[\![e]\!]_{l_0,l_2,\rho} \cup \{l_2 \sqsubseteq l\}$
$e_1 = e_2, e_1 \neq x$	$[\![e_1]\!]_{l_0,l_1,\rho} \cup [\![e_2]\!]_{l_0,l_2,\rho}$
	$\cup\{l_1 \sqsubseteq l,\ l_2 \sqsubseteq l\}$
if e then s_1	$[\![e]\!]_{l_0,l_1,\rho} \cup [\![s_1]\!]_{l_2,l_2,\rho} \cup [\![s_2]\!]_{l_3,l_3,\rho}$
else s_2	$\cup\{l_1 \sqsubseteq l,\ l_2 \sqsubseteq l,\ l_3 \sqsubseteq l\}$
$\forall n \in e\ .\ s$	$[\![n]\!]_{l_0,l_1,\rho} \cup [\![s]\!]_{l_0,l_3,\rho[(l_0,n)\mapsto l_1]}$
	$\cup [\![e]\!]_{l_0,l_2,\rho} \cup \{l_2 \sqsubseteq l_1\} \cup \{l_3 \sqsubseteq l\}$
$\{s_j\}^{j\in 1\ldots m}$	$\bigcup_{j\in 1\ldots m}[\![s_j]\!]_{l_0,l_j,\rho} \bigcup_{j\in 1\ldots m}\{l_j \sqsubseteq l\}$

Figure 6: Constraint Generation

Definition 6. A substitution σ is a *solution* to C if for all $c \in C$ it holds:

$$c \equiv\ B_1 \sqsubseteq B_2\ \implies \vdash \sigma(B_1) \sqsubseteq \sigma(B_2)$$

We denote this by $\sigma \vdash C$. The substitution σ_c is a *minimum solution* to C, denoted by $\sigma_c \vdash_{min} C$, when

- $\sigma_c \vdash C$,
- $dom_{\sigma_c} = Labels(C)$,
- $\sigma \vdash C, l \in Labels(C)\ \implies \vdash \sigma_c(l) \sqsubseteq \sigma(l)$. □

Lemma 5 (Uniqueness of minimal solution)**.**

$$\forall C, \sigma_1, \sigma_2. \qquad \sigma_1, \sigma_2 \vdash_{min} C\ \implies\ \sigma_1 = \sigma_2.$$

Proof. We readily have that $dom_{\sigma_1} = Labels(C) = dom_{\sigma_2}$. As both $\vdash \sigma_1(l) \sqsubseteq \sigma_2(l)$ and $\vdash \sigma_2(l) \sqsubseteq \sigma_1(l)$ hold for all $l \in Labels(C)$, the equality $\sigma_1(l) = \sigma_2(l)$ follows from definition of *Satisfied Constraint*. □

Lemma 6 (Existence of solutions)**.**

$$C \in NF\ \implies\ \exists! \sigma_c.\ \sigma_c \vdash_{min} C.$$

Proof. Consider $C \in NF$. Define a substitution σ_c as $[l_1 \mapsto \mathsf{S}, \ldots, l_n \mapsto \mathsf{S}]$, where $Labels(C) = \{l_1, \ldots, l_n\}$. As σ_c clearly solves all constraints of the form $l \sqsubseteq \tilde{l}, \mathsf{S} \sqsubseteq l$ *or* $\mathsf{D} \sqsubseteq l$ with $l, \tilde{l} \in NF$, it is a solution $\sigma_c \vdash C$. The minimality follows from $dom_{\sigma_c} = Labels(C)$ and from the fact that both $\vdash \mathsf{S} \sqsubseteq \mathsf{S}$ and $\vdash \mathsf{S} \sqsubseteq \mathsf{D}$. □

5.3 Normal Form and Normalization

Lemma 6 shows how the *minimum solution* for a normal form constraint set can be found. This section presents a set of rewrite rules that transform any constraint set to the corresponding normal form, thus making the solution easy to find. In Fig. 7, a set of normalizing rewrite rules are shown.

Redex		*Result*	
$C \in \mathscr{C}$		σ	$\tilde{C}$
a) $C_0 \cup \{\mathsf{S} \sqsubseteq \mathsf{S}\}$	$\rightarrow$	$[\,]$	C_0
b) $C_0 \cup \{\mathsf{S} \sqsubseteq \mathsf{D}\}$	$\rightarrow$	$[\,]$	C_0
c) $C_0 \cup \{\mathsf{D} \sqsubseteq \mathsf{D}\}$	$\rightarrow$	$[\,]$	C_0
d) $C_0 \cup \{l \sqsubseteq \mathsf{S}\}$	$\rightarrow$	$[l \mapsto \mathsf{S}]$	$[l \mapsto \mathsf{S}](C_0)$
e) $C_0 \cup \{\mathsf{D} \sqsubseteq l\}$	$\rightarrow$	$[l \mapsto \mathsf{D}]$	$[l \mapsto \mathsf{D}](C_0)$

Figure 7: Constraints Normalization ($\rightarrow$)

Definition 7. For a $C \in \mathscr{C}$ the *application* of a normalization rewrite rule from Fig. 7 returns a constraint set $\tilde{C}$ and a substitution σ. We denote this by $C \rightarrow^{\sigma} \tilde{C}$. Exhaustive application is denoted by $C \rightarrow^{\star\ \sigma} \tilde{C}_k$. □

Lemma 7 (Termination)**.** For all C, $\tilde{C}$, and σ, whenever $C \rightarrow^{\sigma} \tilde{C}$ then $|\tilde{C}| < |C|$.

Proof. By inspecting Fig. 7, it is obvious that every rule reduces the number of constraints by one. □

Lemma 8 (Solution preservation)**.**

$$C \rightarrow^{\sigma} \tilde{C} \ \wedge\ \tilde{\sigma} \vdash \tilde{C} \implies \sigma \rhd \tilde{\sigma} \vdash C.$$

Proof. By case analysis of the normalization rewrite rules. a), b), c): Since σ is the empty substitution, it is clear that $\sigma \rhd \tilde{\sigma} = \tilde{\sigma}$. As $Labels(C) = Labels(\tilde{C})$, it follows that $\sigma \rhd \tilde{\sigma} \vdash C$.
d), e): The constraint removed from C is solved by σ. Thus, $\tilde{\sigma} \vdash \sigma(C)$ follows from $\tilde{\sigma} \vdash \tilde{C}$. As $Labels(\sigma(C)) = Labels(\tilde{C})$ and $dom_{\sigma} \cap Labels(\tilde{C}) = \varnothing$, we obtain $\sigma \rhd \tilde{\sigma} \vdash C$. □

Lemma 9 (*NF* or *Error*)**.** For all C and $\tilde{C}$, whenever $C \rightarrow^{*\ \sigma} \tilde{C}$ then

$$\tilde{C} \in NF \ \vee\ \tilde{C} \in Error.$$

Proof. By definition of *exhaustive application*, no rewrite rule can apply to $\tilde{C}$, then only four types of constraints may appear in $\tilde{C}$. Namely $\mathsf{S} \sqsubseteq l$, $l \sqsubseteq \mathsf{D}$, $l \sqsubseteq \tilde{l}$ or $\mathsf{D} \sqsubseteq \mathsf{S}$. If $\mathsf{D} \sqsubseteq \mathsf{S}$ is present, then $\tilde{C} \in Error$, else $\tilde{C} \in NF$. □

Lemma 10 (Minimal solution preservation)**.**

$$C \rightarrow^{\sigma} \tilde{C} \ \wedge\ \sigma_{\tilde{c}} \vdash_{min} \tilde{C} \implies \sigma \rhd \sigma_{\tilde{c}} \vdash_{min} C.$$

Proof. Lemma 8 implies that $\sigma_c = \sigma \rhd \sigma_{\tilde{c}} \vdash C$. To show minimality first note that $dom_{\sigma_c} = dom_{\sigma} \cup Labels(\tilde{C}) = Labels(C)$ with the union being disjoint. Now assume that $l \in Labels(C)$, $\tilde{\sigma} \vdash C$ and consider the following two cases.
$l \in Labels(\tilde{C})$: Clearly $\tilde{\sigma} \vdash \tilde{C}$. As $\sigma_{\tilde{c}}$ is minimal, $\vdash \sigma_{\tilde{c}}(l) \sqsubseteq \tilde{\sigma}(l)$. Thus, from $\sigma_c(l) = \sigma_{\tilde{c}}(l)$ we get that $\vdash \sigma_c(l) \sqsubseteq \tilde{\sigma}(l)$.
$l \in dom_{\sigma}$: σ was obtained by applying rule e) or d). Thus, either $\sigma_c(l) = \sigma(l) = \mathsf{S}$ or $\sigma_c(l) = \sigma(l) = \mathsf{D} = \tilde{\sigma}(l)$. In both cases $\vdash \sigma_c(l) \sqsubseteq \tilde{\sigma}(l)$ holds. □

Theorem 2 (Unique minimal solution)**.**

$$C \rightarrow^{*\ \sigma} \tilde{C} \ \wedge\ \tilde{C} \in NF \implies \exists! \sigma_c.\ \sigma_c \vdash_{min} C.$$

Proof. We obtain $\sigma_{\tilde{c}}$ such that $\sigma_{\tilde{c}} \vdash_{min} \tilde{C}$ from Lemma 6. By definition we have that $\exists k \in \mathbb{N}$ and $\sigma_1, \ldots, \sigma_k$ such that $C \rightarrow^{\sigma_1} \tilde{C}_1 \rightarrow^{\sigma_2} \ldots \rightarrow^{\sigma_k} \tilde{C}_k = \tilde{C}$ and $\sigma = \sigma_1 \rhd \ldots \rhd \sigma_k$. Thus, by introducing $\tilde{C}_0 = C$, we get $\sigma_i \rhd \ldots \rhd \sigma_k \rhd \sigma_{\tilde{c}} \vdash_{min} \tilde{C}_{i-1}$ for all $1 \leq i \leq k$ from Lemma 10. Thus, $\sigma_c \vdash_{min} C$ with $\sigma_c = \sigma \rhd \sigma_{\tilde{c}}$. □

Combine Theorem 2 and Lemma 5, the *minimal solution* to any constraint set can be found as follows: first normalize the constraint set by applying constraint rewrite rules in figure 3, and then find the unique minimal solution to the normal form constraint set. The composition of the substitutions is the *minimal solution*.

5.4 Binding Time Analysis Correctness

Lemma 11 (Completeness)**.** Consider a binding time type environment such that $\Gamma^b(x_i) = \tau_i^{b_i}$, and $\rho(l_0, x_i) \in$ Labels. Then, $\forall e, s, b$:

- $\Gamma^b \vdash e : \tau^b \implies \exists! \sigma . \sigma \vdash_{min} [\![e]\!]_{l_0,l,\rho} \wedge \sigma(l) \sqsubseteq b$
- $\Gamma^b \vdash s^b \implies \exists! \sigma . \sigma \vdash_{min} [\![s]\!]_{l_0,l,\rho} \wedge \sigma(l) \sqsubseteq b$

Lemma 12 (Soundness)**.** Consider a typing environment such that $\Gamma(x_i) = \tau_i$, and $\rho(l_0, x_i) \in$ Labels. There exists a binding time environment such that $|\Gamma^b| = \Gamma$ and

- $\forall\, e.\ \Gamma \vdash e : \tau \wedge \sigma \vdash_{min} [\![e]\!]_{l_0,l,\rho} \implies \Gamma^b \vdash e : \tau^{\sigma(l)}$
- $\forall\, s.\ \Gamma \vdash s \wedge \sigma \vdash_{min} [\![s]\!]_{l_0,l,\rho} \implies \Gamma^b \vdash s^{\sigma(l)}$

The proofs for the two lemmas above are in Section 2 of the supplementary document.

6 Conclusions and Future work

In this paper we showed how the basic hybrid ODE formalism can be extended to support certain types of partial derivatives and equational constraints. The treatment is generic and so can be applied to any hybrid systems reachability analysis, and has been implemented in the context of the Acumen modeling language. Interesting future work includes investigation of more advanced type systems[28, 21] that can ensure that a dynamic value is a variable so that we can build a type system that is able to detect all possible run-time errors that can interfere with the elimination of partial derivatives.

References

[1] *Acumen implementation*. https://bitbucket.org/effective/acumen-dev/downloads.

[2] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger & Pei-Hsin Ho (1993): *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, doi:10.1007/3-540-57318-6_30.

[3] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar & Insup Lee (2000): *Modular Specification of Hybrid Systems in CHARON*. In: *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, HSCC '00, Springer-Verlag, doi:10.1007/3-540-46430-1_5.

[4] Stanley Bak, Sergiy Bogomolov, Thomas A Henzinger, Taylor T Johnson & Pradyot Prakash (2016): *Scalable static hybridization methods for analysis of nonlinear systems*. In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ACM, pp. 155–164, doi:10.1145/2883817.2883837.

[5] Stanley Bak, Sergiy Bogomolov & Taylor T Johnson (2015): *HYST: a source transformation and translation tool for hybrid automaton models*. In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ACM, pp. 128–133, doi:10.1145/2728606.2728630.

[6] Hsu Chen (2007): *Passive dynamic walking with knees: A point foot model*. Ph.D. thesis, Massachusetts Institute of Technology.

[7] Xin Chen, Erika Ábrahám & Sriram Sankaranarayanan (2013): *Flow*: An analyzer for non-linear hybrid systems*. In: *International Conference on Computer Aided Verification*, Springer, pp. 258–263, doi:10.1007/978-3-642-39799-8_18.

[8] Niels H. Christensen & Robert Glück (2004): *Offline Partial Evaluation Can Be As Accurate As Online Partial Evaluation*. *ACM Trans. Program. Lang. Syst.* 26(1), pp. 191–220, doi:10.1145/963778.963784.

[9] Steve Collins, Andy Ruina, Russ Tedrake & Martijn Wisse (2005): *Efficient bipedal robots based on passive-dynamic walkers*. *Science* 307(5712), pp. 1082–1085, doi:10.1126/science.1107799.

[10] Charles Consel, Julia L Lawall & Anne-Françoise Le Meur (2004): *A tour of Tempo: A program specializer for the C language*. *Science of Computer Programming* 52(1), pp. 341–370, doi:10.1016/j.scico.2004.03.011.

[11] Krzysztof Czarnecki, John T O'Donnell, Jörg Striegnitz & Walid Taha (2004): *DSL implementation in MetaOCaml, Template Haskell, and C++*. In: *Domain-Specific Program Generation*, Springer, pp. 51–72, doi:10.1007/978-3-540-25935-0_4.

[12] Adam Duracz (2016): *Rigorous Simulation: Its Theory and Applications*. Ph.D. thesis, Halmstad University Press.

[13] Goran Frehse (2005): *PHAVer: Algorithmic verification of hybrid systems past HyTech*. In: *International workshop on hybrid systems: computation and control*, Springer, pp. 258–273, doi:10.1007/s10009-007-0062-x.

[14] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang & Oded Maler (2011): *SpaceEx: Scalable verification of hybrid systems*. In: *International Conference on Computer Aided Verification*, Springer, pp. 379–395, doi:10.1007/978-3-642-22110-1_30.

[15] Sicun Gao, Soonho Kong & Edmund M Clarke (2013): *dReal: An SMT solver for nonlinear theories over the reals*. In: *International Conference on Automated Deduction*, Springer, pp. 208–214, doi:10.1007/978-3-642-38574-2_14.

[16] Arne J. Glenstrup, Henning Makholm & Jens P. Secher (1999): *C-MIX: Specialization of C Programs*. In: *Partial Evaluation - Practice and Theory, DIKU 1998 International Summer School*, Springer-Verlag, London, UK, UK, pp. 108–154, doi:10.1007/3-540-47018-2_4. Available at `http://dl.acm.org/citation.cfm?id=645795.665921`.

[17] Carsten K Gomard & Neil D Jones (1991): *A partial evaluator for the untyped lambda-calculus*. *Journal of functional programming* 1(01), pp. 21–69, doi:10.1017/S0956796800000058.

[18] Thomas A Henzinger, Pei-Hsin Ho & Howard Wong-Toi (1997): *HyTech: A model checker for hybrid systems*. In: *International Conference on Computer Aided Verification*, Springer, pp. 460–463, doi:10.1007/s100090050008.

[19] Paul Hudak (1998): *Domain Specific Languages*. In: *Handbook of Programming Languages, Vol. III: Little Languages and Tools*, chapter 3, MacMillan, Indianapolis, pp. 39–60, doi:10.1145/1925844.1922397.

[20] Neil D Jones, Peter Sestoft & Harald Søndergaard (1985): *An experiment in partial evaluation: the generation of a compiler generator*. In: *Rewriting techniques and applications*, Springer, pp. 124–140, doi:10.1007/3-540-15976-2_6.

[21] Ik-Soon Kim, Kwangkeun Yi & Cristiano Calcagno (2006): *A Polymorphic Modal Type System for Lisp-like Multi-staged Languages*. *SIGPLAN Not.* 41(1), pp. 257–268, doi:10.1145/1111320.1111060.

[22] Soonho Kong, Sicun Gao, Wei Chen & Edmund Clarke (2015): *dReach: δ-reachability analysis for hybrid systems*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 200–205, doi:10.1007/978-3-662-46681-0_15.

[23] Marjan Mernik, Jan Heering & Anthony M. Sloane (2005): *When and How to Develop Domain-specific Languages*. *ACM Comput. Surv.* 37(4), pp. 316–344, doi:10.1145/1118890.1118892.

[24] Eugenio Moggi (1997): *A categorical account of two-level languages*. *Electronic Notes in Theoretical Computer Science* 6, p. 272, doi:10.1016/S1571-0661(05)80155-0.

[25] Tiark Rompf & Martin Odersky (2010): *Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs*. In: *Acm Sigplan Notices*, 46, ACM, pp. 127–136, doi:10.1007/s10990-011-9072-1.

[26] Cherif Salama, Gregory Malecha, Walid Taha, Jim Grundy & John O'Leary (2009): *Static consistency checking for verilog wire interconnects: using dependent types to check the sanity of verilog descriptions*. In: *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, ACM, pp. 121–130, doi:10.1007/s10990-011-9072-1.

[27] Tim Sheard & Simon Peyton Jones (2002): *Template meta-programming for Haskell*. In: *Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*, ACM, pp. 1–16, doi:10.1145/636517.636528.

[28] Mark R Shinwell, Andrew M Pitts & Murdoch J Gabbay (2003): *FreshML: Programming with binders made simple*. In: *ACM SIGPLAN Notices*, 38, ACM, pp. 263–274, doi:10.1145/944746.944729.

[29] Ryan W. Sinnet & Aaron D. Ames (2009): *2D bipedal walking with knees and feet: A hybrid control approach*. In: *Conference on Decision and Control*, pp. 3200–3207, doi:10.1109/CDC.2009.5400503.

[30] Arvind K. Sujeeth, Kevin J. Brown, Hyoukjoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky & Kunle Olukotun (2014): *Delite: A Compiler Architecture for Performance-Oriented Embedded Domain-Specific Languages*. *ACM Trans. Embed. Comput. Syst.* 13(4s), pp. 134:1–134:25, doi:10.1145/2584665.

A Appendix

Cam and Follower

$x = (1.5 - \frac{cos(\frac{\pi}{2} - \theta))}{2}) * (1 + \frac{cos2(\frac{\pi}{2} - \theta)}{5})$	`x = (1.5-cos(pi/2-t)/2)*(1+cos(2*(pi/2-t)/5))`
$\ddot{\theta} = 1 \quad v = \frac{\partial}{\partial \theta} x\, \dot{\theta} \quad a = \dot{v}$	`t'' = 1, v = x'[t]*t', a = (v)'`

Compass Gait Biped

$q = [\theta_1, \theta_2]\ \ m_1 = 1\ \ m_2 = 2\ \ l = 1$	`q = (t1,t2), m1 = 1, l = 1, m2 =2,`
$\gamma = 0.044 \quad g = 9.8$	`r = 0.044, g = 9.8,`
$x_1 = \frac{1}{2} l sin\theta_1\ \ y_1 = \frac{1}{2} l cos\theta_1$	`x1 = l/2*sin(t1),y1 =l/2*cos(t1),`
$x_2 = l sin\theta_2\ \ y_2 = l cos\theta_2$	`x2 = l*sin(t2), y2 = l*cos(t2),`
$x_3 = x_2 + \frac{l}{2} sin(\theta_2 - \theta_1)\ \ y_3 = y_2 - \frac{l}{2} cos(\theta_2 - \theta_1)$	`x3=x2+l/2*sin(t2-t1),y3=y2-l/2*cos(t2-t1),`
$L = T - V \quad guard = l sin\gamma(sin\theta_1 + sin(\theta_2 - \theta_1))$	`L=T-V,guard=l*sin(r)*(sin(t1)+sin(t2-t1)),`
$T = \frac{1}{2} m_1 (\dot{x_1}^2 + \dot{y_1}^2 + \dot{x_3}^2 + \dot{y_3}^2)$	`T = 1/2*m1*((x1)'^2+(y1)'^2+(x3)'^2+(y3)'^2)`
$\quad + \frac{1}{2} m_2 (\dot{x_2}^2 + \dot{y_2}^2)$	`+ 1/2*m2*((x2)'^2 + (y2)'^2),`
$V = m_1 g(y_1 + y_3) + m_2 g y_2$	`V = m1*g*(y1+y3)+m2*g*y2,`
$H = H_1^{-1} \cdot H_2 \cdot [\dot{\theta_1}^-, \dot{\theta_2}^-]^T$	`H = inv(H1)*H2*trans((t1',t2')),`
$\forall i \in \{1...\|q\|)\}. \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q_i}} \right) - \frac{\partial L}{\partial q_i} = 0$	`foreach i in 0:length(q) - 1 do` `L'[(q(i))']' - L'[q(i)] = 0 ,`

$H_1 =$
$[m_1 l^2 (\frac{5}{4} - \frac{cos\theta_2^-}{2}) + m_2 l^2 \quad \frac{m_1}{4} l^2 (1 - 2cos\theta_2^-)$
$\frac{m_1}{2} l^2 cos\theta_2^- \quad \frac{m_1}{4} l^2]$
	`H1 =`
	`((m1*l^2*((5/4-cos(t2)/2)+m2*l^2),`
	`m1/4*l^2*(1-2*cos(t2))),`
	`( m1/2*l^2*cos(t2), m1/4*l^2)),`

$H_2 = [-\frac{m_1}{4} l^2 + (m_2 l^2 + m1 l^2) cos\theta_2^- \quad -\frac{m_1}{4} l^2$
$\frac{m_1}{4} l^2 \quad 0\,]$
	`H2 =`
	`((-m1/4*l^2+(m2*l^2+m1*l^2)*cos(t2),-m1/4*l^2),`
	`( m1/4*l^2, 0),`

if $guard < 0 \wedge \dot{guard} < 0$ then
$\theta_1^+ = \theta_2^- - \theta_1^- \quad \theta_2^+ = -\theta_2^-$
$\dot{\theta_1}^+ = H(0) \quad \dot{\theta_2}^+ = H(1)$
	`if guard <0 && (guard)' then`
	`t1 += t2 - t1, t2 += -t2,`
	`t1' += H(0), t2' += H(1) noelse`

Figure 8: Two Examples Problems in Mathematical Notation and in Acumen Syntax

Value $v^b ::= k^{\mathsf{S}} \mid e^{\mathsf{D}} \mid \langle v_j^{b_j} \rangle^{j \in 1 \ldots m}$

Normal Form Equation $w^b ::= (x^b = v^b)^b \mid (x^{+b} = v^b)^b \mid (v^b = v^b)^b \mid (\mathsf{if}\ v^b\ \mathsf{then}\ w^b\ \mathsf{else}\ w^b)^b \mid (\{w_j^{b_j}\}^{j \in 1 \ldots m})^b$

Function application $[\![f(\langle v_j^{\mathsf{S}} \rangle^{j \in 1 \ldots m})]\!] = v^{\mathsf{S}}$ such that $v^{\mathsf{S}} \equiv f(\langle v_j \rangle^{j \in 1 \ldots m})^{\mathsf{S}}$

Time derivative $D_f(\langle v_j^{\mathsf{D}} \rangle^{j \in 1 \ldots m}, \langle v_j'^{\mathsf{D}} \rangle^{j \in 1 \ldots m}) = v^{\mathsf{D}}$ such that $v^{\mathsf{D}} \equiv (\frac{d}{dt} f(\langle v_j^{\mathsf{D}} \rangle^{j \in 1 \ldots m}, \langle v_j'^{\mathsf{D}} \rangle^{j \in 1 \ldots m}))^{\mathsf{S}}$

Partial derivative $P_f(\langle v_j^{\mathsf{D}} \rangle^{j \in 1 \ldots m}, \langle v_j'^{\mathsf{D}} \rangle^{j \in 1 \ldots m}, x^{\mathsf{D}}) = v^{\mathsf{D}}$ such that $v^{\mathsf{D}} \equiv \frac{\partial}{\partial x} f(\langle v_j^{\mathsf{D}} \rangle^{j \in 1 \ldots m}, \langle v_j'^{\mathsf{D}} \rangle^{j \in 1 \ldots m})^{\mathsf{D}}$

Free Variables

$$FV(x^b) = \{x\} \quad FV(\langle e_j{}^{b_j} \rangle^{j \in 1 \ldots m}) = \bigcup_j FV(e_j^{b_j}) \quad FV((f(\langle e_j^{b_j} \rangle^{j \in 1 \ldots m})^b)) = \bigcup_j FV(e_j^{b_j}) \quad LV(x = e) = \{x\}$$

$$FV(e_1^b(e_2^b)^b) = FV(e_1^b) \cup FV(e_2^b) \quad FV(\tfrac{d}{dt}(e^b)) = FV(e^b) \quad FV(\tfrac{\partial}{\partial e_2^b} e_1^b) = FV(e_1^b) \cup FV(e_2^b) \quad LV(\{s_j\}^{j \in 1 \ldots m}) = \bigcup_j LV(s_j)$$

$e^b \hookrightarrow_e v^b \cup \{\mathsf{err}\}$

$$\frac{}{k^b \hookrightarrow_e k^b} \quad \frac{e_j^{b_j} \hookrightarrow_e v_j^{b_j}}{(\langle e_j^{b_j} \rangle^{j \in 1 \ldots m})^b \hookrightarrow_e (\langle v_j^{b_j} \rangle^{j \in 1 \ldots m})^b} \quad \frac{e_1^b \hookrightarrow_e (\langle v_j^b \rangle^{j \in 1 \ldots m})^b \quad e_2^{\mathsf{S}} \hookrightarrow_e i^{\mathsf{S}}}{e_1^b(e_2^{\mathsf{S}})^b \hookrightarrow_e v_i^b} \quad \frac{e^{\mathsf{S}} \hookrightarrow_e (\langle v_j^{\mathsf{S}} \rangle^{j \in 1 \ldots m})^{\mathsf{S}}}{f(e^{\mathsf{S}})^{\mathsf{S}} \hookrightarrow_e [\![f(\langle v_j \rangle^{j \in 1 \ldots m})]\!]}$$

$$\frac{e_1^b \hookrightarrow_e v_1^b \quad e_2^{\mathsf{D}} \hookrightarrow_e v_2^{\mathsf{D}}}{e_1^b(e_2^{\mathsf{D}})^{\mathsf{D}} \hookrightarrow_e v_1^b(v_2^{\mathsf{D}})^{\mathsf{D}}} \quad \frac{}{x^{\mathsf{D}} \hookrightarrow_e x^{\mathsf{D}}} \quad \frac{e^{\mathsf{D}} \hookrightarrow_e v^{\mathsf{D}}}{f(e^{\mathsf{D}})^{\mathsf{D}} \hookrightarrow_e f(v^{\mathsf{D}})^{\mathsf{D}}} \quad \frac{e^{\mathsf{S}} \hookrightarrow_e q^{\mathsf{S}}}{\frac{d}{dt}(e^{\mathsf{S}})^{\mathsf{S}} \hookrightarrow_e 0^{\mathsf{S}}} \quad \frac{e^b \hookrightarrow_e x^b}{\frac{d}{dt}(e^b)^b \hookrightarrow_e x'^b} \quad \frac{e_1^b \hookrightarrow_e q_1^{\mathsf{D}} \quad e_2^{\mathsf{D}} \hookrightarrow_e x^{\mathsf{S}}}{(\frac{\partial}{\partial e_2^{\mathsf{D}}} e_1^b)^{\mathsf{S}} \hookrightarrow_e 0^{\mathsf{D}}}$$

$$\frac{e_1^{\mathsf{D}} \hookrightarrow_e x_1^{\mathsf{D}} \quad e_2^{\mathsf{D}} \hookrightarrow_e x_2^{\mathsf{D}} \quad i^{\mathsf{D}} = \begin{cases} 1^{\mathsf{D}} \text{ if } x_1 = x_2 \\ 0^{\mathsf{D}} \text{ otherwise} \end{cases}}{(\frac{\partial}{\partial e_2^{\mathsf{D}}} e_1^{\mathsf{D}})^{\mathsf{D}} \hookrightarrow_e i^{\mathsf{D}}} \quad \frac{e^{\mathsf{D}} \hookrightarrow_e f((\langle v_j^{b_j} \rangle^{j \in 1 \ldots m})^{\mathsf{D}})^{\mathsf{D}} \quad (\frac{d}{dt} v_j^{b_j})^{\mathsf{D}} \hookrightarrow_e v_j'^{b_j}}{\frac{d}{dt}(e^{\mathsf{D}})^{\mathsf{D}} \hookrightarrow_e D_f(\langle v_j^{b_j} \rangle^{\mathsf{D}}, \langle v_j'^{b_j} \rangle^{\mathsf{D}})^{\mathsf{D}}} \quad \frac{\begin{array}{c} e_1^{\mathsf{D}} \hookrightarrow_e f((\langle v_j^{b_j} \rangle^{j \in 1 \ldots m})^{\mathsf{D}})^{\mathsf{D}} \\ e_2^{\mathsf{D}} \hookrightarrow_e x^{\mathsf{D}} \quad (\frac{\partial}{\partial x^{\mathsf{D}}} v_j^{b_j})^{b_j} \hookrightarrow_e v_j'^{b_j} \end{array}}{\frac{\partial}{\partial e_2^{\mathsf{D}}} e_1^{\mathsf{D}} \hookrightarrow_e P_f(\langle v_j^{b_j} \rangle^{\mathsf{D}}, \langle v_j'^{b_j} \rangle^{\mathsf{D}}, x^{\mathsf{D}})^{\mathsf{D}}}$$

$s^b \hookrightarrow_s w^b \cup \{\mathsf{err}\}$

$$\frac{FV(e^b) \cap LV(s^{b_s}) = \emptyset \quad e^b \hookrightarrow_e v^b \quad s^{b_s}[x^b := v^b] \hookrightarrow_s w^{b_s}}{\{x^b = e^b\}^b \uplus s^{b_s} \hookrightarrow_s \{x^b = v^b\} \uplus w^{b_s}} \quad \frac{FV(e^b) \cap LV(s^{b_s}) = \emptyset \quad e^b \hookrightarrow_e v^b \quad s^{b_s} \hookrightarrow_s w^{b_s}}{\{x^{+b_1} = e^b\}^b \uplus s^{b_s} \hookrightarrow_s \{x^{+b_1} = v^b\}^b \uplus w^{b_s}}$$

$$\frac{\begin{array}{c} FV(e_1^{b_1}) \cup FV(e_2^{b_2}) \cap LV(s^{b_s}) = \emptyset \\ e_1^{b_1} \hookrightarrow_e v_1^{b_1} \quad e_2^{b_2} \hookrightarrow_e v_2^{b_2} \quad s^{b_s} \hookrightarrow_s w^{b_s} \end{array}}{\{e_1^{b_1} = e_2^{b_2}\}^b \uplus s^{b_s} \hookrightarrow_s \{v_1^{b_1} = v_2^{b_2}\}^b \uplus w^{b_s}} \quad \frac{\begin{array}{c} e^{\mathsf{S}} \hookrightarrow_e t_j^{\mathsf{S}} \quad s_j^{b_j} \hookrightarrow_s w_j^{b_j} \quad s^{b_s} \hookrightarrow_s w^{b_s} \\ FV(e^{\mathsf{S}}) \cap LV(s^{b_s}) = \emptyset \quad t_1^{\mathsf{S}} = \mathsf{true}^{\mathsf{S}} \quad t_2^{\mathsf{S}} = \mathsf{false}^{\mathsf{S}} \end{array}}{\{\mathsf{if}\ e^{\mathsf{S}}\ \mathsf{then}\ s_1^{b_1}\ \mathsf{else}\ s_2^{b_2}\}^b \uplus s^{b_s} \hookrightarrow_s w_j^{b_j} \uplus w^{b_s}}$$

$$\frac{\begin{array}{c} FV(e^{\mathsf{D}}) \cap LV(s^{b_s}) = \emptyset \quad e^{\mathsf{D}} \hookrightarrow_e v^{\mathsf{D}} \\ s_1^{b_1} \hookrightarrow_s w_1^{b_s} \quad s_2^{b_2} \hookrightarrow_s w_2^{b_s} \quad s^{b_s} \hookrightarrow_s w^{b_s} \end{array}}{\{\mathsf{if}\ e^{\mathsf{D}}\ \mathsf{then}\ s_1^{b_1}\ \mathsf{else}\ s_2^{b_2}\}^b \uplus s^{b_s} \hookrightarrow_s \{\mathsf{if}\ v^{\mathsf{D}}\ \mathsf{then}\ w_1^{b_s}\ \mathsf{else}\ w_2^{b_s}\}^b \uplus w^{b_s}}$$

$$\frac{\begin{array}{c} FV(e^b) \cap LV(s^{b_s}) = \emptyset \quad e^b \hookrightarrow_e (\langle v_j^b \rangle^{j \in 1 \ldots m})^b \\ (\{s_1^{b_1}[n := v_j^b]\}^{j \in 1..m})^{b_1} \hookrightarrow_s (\{w_j^{b_1}\}^{j \in 1..m})^{b_1} \\ s^{b_s} \hookrightarrow_s w^{b_s} \end{array}}{\{\forall n \in e^b\ s_1^{b_1}\}^{b_1} \uplus s^{b_s} \hookrightarrow_s (\{w_j^{b_1}\}^{j \in 1..n})^{b_1} \uplus w^{b_s}}$$

Figure 9: Specialization big-step semantics