RICE UNIVERSITY

Making Hybrid Systems Easier to Model, Simulate, and Visualize

by

Yingfu Zeng

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Walid Taha, Director
Professor of Computer Science
Halmstad University

Robert Cartwright, Chairman
Professor of Computer Science

Marcia O'Malley
Professor of Mechanical Engineering

T. S. Eugene Ng
Professor of Computer Science

Houston, Texas
January, 2019
ABSTRACT

Making Hybrid Systems Easier to Model, Simulate, and Visualize

by

Yingfu Zeng

Specifying the behavior desired of hybrid systems requires analytical modeling of physical phenomena. Similarly, testing them requires simulation of continuous systems. While numerous tools support the later stages of developing simulation codes, there is still a large gap between the behavior of analytical models and corresponding simulation codes, impeding the development of novel hybrid systems. Support for partial derivatives, in particular, is severely limited in present mainstream modeling and simulation languages and in reachability analysis tools for hybrid systems. Current languages and tools do not provide such language construct, forcing the modeler to manually transform the model or resort to crude approximation of the problematic semantics.

In this thesis, we demonstrate that compile-time transformations can improve hybrid system formalisms by supporting partial derivatives and equational constraints. These improvements enable the user 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 expressivity requires using a binding time-analysis (BTA), program differentiation, symbolic Gaussian elimination, and abstract interpretation using interval analysis. We give an operational semantics for the specialization process along with a declarative and algorithmic specifications
of the binding-time analysis. A type safety theorem is given to show the correctness of the semantics for specialization. The declarative specification of binding-time analysis is used to prove soundness with respect to the specialization process. We also provide an open-source implementation demonstrating our approach.
Acknowledgments

Many people have contributed to Acumen, its theory and the applications described in this thesis. The design of the language and its implementation was led by Walid Taha. Initial work on the user interface and the traditional interpreter was implemented by Paul Brauner. Initial work on the optimized interpreter was done by Kevin Atkinson. The current traditional and enclosure interpreter, 3D visualization functionality and underlying libraries were developed by myself, Adam Duracz, Ferenc A. Barth, and Fei Xu.

On a more personal note, there are several people I want to thank, without whom I could not have finished this dissertation. First of all, I am extremely grateful to my adviser Walid Taha for his guidance and countless discussions. His deep insights helped at various stages of my research. I also remain indebted for his understanding and support during the most difficult times. I would wish to express my gratitude to Robert Cartwright for extended discussions and leading me into the field of programming language in general. I would like to thank Marcia O’malley, Roland Philippsen and Aaron Ames for their guidance, valuable suggestions and expertise in robotics that had significantly contributed to this thesis.

My warmest thanks also go to Eugenio Moggi and Neil Jones for their valuable feedbacks that helped formalizing the semantics of Acumen’s partial evaluator.

The members of the EMG group have contributed immensely to my personal and professional time at Halmstad. The group has been a source of friendship as well as good advice and collaboration. I shall be missing our 3 pm EMG meetings. Special thanks go to Adam Duracz, Ferenc A. Barth and Jawad Masood for their collaboration and contribution in various projects related to this thesis.
Yuanjia Bao, Fei Xu, Maytheewat Aramrattana, Wenjie Jiang, Chad Rose and Hamim Zafar are valuable friends who have always wished well and I have enjoyed the time that I spent with them.

Words can not express the feelings I have for my parents for their constant unconditional support.

This work was supported by the US National Science Foundation award CPS-1136099, the ELLIIT project, and Swedish Knowledge Foundation (KK) projects FAR-EIS and AstaMoCA.
Contents

Abstract ii
Acknowledgments iv
List of Illustrations ix
List of Tables xi

1 Introduction 1
  1.1 Background .................................................. 1
  1.2 Thesis ......................................................... 2
  1.3 Contribution .................................................. 2
  1.4 List of Related Publications ................................. 3

I Modeling 6

2 Background 7
  2.1 Introduction .................................................. 7
  2.2 Acumen: A Small, Experimental Language for Hybrid Modeling ... 10
  2.3 Related Work .................................................. 14

3 Basic Aspects of Cyber-physical Systems 17
  3.1 Geometry and Visual Form ................................. 17
  3.2 Particle Dynamics and Impacts ........................... 22
  3.3 Composite Models .......................................... 27
  3.4 Control ........................................................ 29
  3.5 Discretization and Quantization .......................... 33
# 4 Case Studies

4.1 Case Study I: Quadcopter ................................................. 36

4.1.1 Background ................................................................. 36

4.1.2 Reducing Model Complexity Through Control ...................... 37

4.1.3 Mathematical Model ....................................................... 38

4.1.4 Acumen Model for Quadcopter ........................................... 39

4.2 Lagrangian Modeling, and Why we Need it .............................. 40

4.3 Case Study II: The RiceWrist-S robot .................................... 46

4.3.1 Background ................................................................. 46

4.3.2 Analytical Model ............................................................ 47

4.4 Case Study III: Electromechanical Systems. .......................... 51

4.4.1 Background ................................................................. 51

4.4.2 Analytical Model ............................................................ 52

4.5 Case Study IV: A Compass Gait Biped .................................. 53

4.5.1 Continuous Dynamics ....................................................... 55

4.5.2 Discrete Events ............................................................. 55

# II Compile-Time Extensions to Hybrid ODEs

5 Introduction ........................................................................... 58

5.1 Contributions ................................................................. 58

5.2 Related Work .................................................................. 60

6 A Differential Equation Language ........................................... 64

6.1 Syntax and Type System ..................................................... 64

6.2 Example: A Lagrangian Model ............................................ 68

6.3 A Cam and Follower Example ............................................. 72

6.4 A Compass Gait Biped Example .......................................... 73
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1 Continuous Dynamics and Discrete Event</td>
<td>73</td>
</tr>
<tr>
<td>7 BTA and Specialization</td>
<td>75</td>
</tr>
<tr>
<td>7.1 Binding Time Analysis</td>
<td>75</td>
</tr>
<tr>
<td>7.2 Specialization</td>
<td>77</td>
</tr>
<tr>
<td>7.3 Type Safety</td>
<td>78</td>
</tr>
<tr>
<td>8 Algorithmic Specification</td>
<td>80</td>
</tr>
<tr>
<td>8.1 Generating Binding Time Constraints</td>
<td>80</td>
</tr>
<tr>
<td>8.2 Control Flow and Control Scope</td>
<td>80</td>
</tr>
<tr>
<td>8.3 Normal Form and Normalization</td>
<td>85</td>
</tr>
<tr>
<td>8.4 Binding Time Analysis Correctness</td>
<td>87</td>
</tr>
<tr>
<td>9 Conclusion and Future work</td>
<td>92</td>
</tr>
<tr>
<td>9.1 Summery of Results</td>
<td>92</td>
</tr>
<tr>
<td>9.2 Future Work</td>
<td>94</td>
</tr>
<tr>
<td>Bibliography</td>
<td>96</td>
</tr>
</tbody>
</table>
Illustrations

3.1 The 3D Output Generated for an Instance of the Model Sphere. . . . 18
3.2 The IDE of the Implementation with the Bouncing Ball Model and
Simulation Results. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.3 The Diagram of the Three-Mass/Two-Spring System. . . . . . . . . 28

4.1 Free Body Diagram of the Quadcopter. . . . . . . . . . . . . . . . . . 37
4.2 The Simulation Results of the Quadcopter Model with PID Control. . . 38
4.3 Diagram for Single and Double Pendulums. . . . . . . . . . . . . . . . 43
4.4 The RiceWrist-S, with Superimposed Axes of Rotation. . . . . . . 44
4.5 Diagram of the RiceWrist-S as a Gimbal . . . . . . . . . . . . . . . . . 46
4.6 A Simple Coupled Electromechanical System . . . . . . . . . . . . . . 53
4.7 A Compass Gait Biped [1]. . . . . . . . . . . . . . . . . . . . . . . . . 54

5.1 Syntax and Type system for Acumen . . . . . . . . . . . . . . . . . . 63

6.1 Compiling Pendulum/Mass Example . . . . . . . . . . . . . . . . . . . 68
6.2 A Pendulum Spring-Mass system. . . . . . . . . . . . . . . . . . . . . 69
6.3 Two Case Studies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

7.1 Binding Time Analysis for Acumen . . . . . . . . . . . . . . . . . . . 76

8.1 Constraint Generation . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
8.2 Constraints Normalization (→) ............................................. 85
8.1 Two Examples Problems in Mathematical Notation and in Acumen Syntax ................................................................. 89
8.1 Specialization Big-Step Semantics ........................................... 91
Tables

2.1 Comparison of Modeling and Simulation Tools  . . . . . . . . . . . . . .  11

4.1 Denavit Hartenberg Parameters  . . . . . . . . . . . . . . . . . . . .  48

5.1 Comparison of Compile-time Transformation  . . . . . . . . . . . .  61
Chapter 1

Introduction

This thesis explores a compile-time transformation approach for improving the modeling and simulation of hybrid systems. This chapter motivates the work, states its thesis and describes its contributions and origins.

1.1 Background

Many important products today have both continuous and discrete dynamics. Examples include a modern car, an aircraft, or a team of robots. A system exhibits both continuous and discrete dynamical behavior is called a hybrid system. Since the seminal work of Alur and Henzinger [2] over 20 years ago, there has been a lot of interest in hybrid systems. This includes both theoretical, applied, and commercial outcomes, the later class includes simulation tools such as Modelica [3], Simscape [4] and MapleSim [5]. Unfortunately, simulation tools generally do not provide a guarantee of the correctness of the simulation with respect to the model or how the more expressive features of the tools can be reduced to a core subset, such as the one used by verification tools. It is evident that the processing of such features requires symbolic algebra to perform operations such as differentiation, but it is not evident how this is done. Instead, there is a broad range of algebraic manipulations that can be used to improve the expressivity of a continuous modeling formalism, but it is far from clear what principles should guide the implementation of such methods. In
particular, it is tempting to consider the use of a generic symbolic algebra engine as a front-end to achieve this goal. What is surprising is that such a combination can prove highly problematic in practice.

1.2 Thesis

Our thesis is useful to view hybrid systems models as programs. In particular, if we view frameworks for defining hybrid systems as programming languages, and developing hybrid system models as programming, then developing an appropriate Integrated Development Environment (IDE) can greatly facilitate the modeling and simulation processes.

1.3 Contribution

This thesis consists of two major contributions. Part I of this dissertation illustrates what language features are needed to model hybrid systems and shows how modeling languages can be improved by integrating several features, most notably, partial derivatives, differentiation without duplication, and support for equations. These features do not appear to be addressed in a satisfactory manner in mainstream modeling and simulation tools. Part II presents a systematic method for translating such an expressive language with partial derivatives and equations to a core language supporting ODEs, guards, and reset maps.

1. Identifying a number of aspects that are representative of some of the most basic and prominent modeling needs of the CPS domain (Chapter 3).

2. Showing that a small language for modeling hybrid (continuous/discrete) system is sufficient for modeling these aspects (Chapter 4).
3. Reporting on four larger case studies expressed within the same language (Chapter 5).

4. Formalizing the syntax and type system for a core differential equational language (Chapter 6).

5. Formalizing a declarative specification of binding-time analysis (BTA) and a big step semantics for specialization, along with a formal proof of type safety (Chapter 7).

6. Presenting an algorithmic specification of the BTA and show that this algorithmic specification is faithful to the declarative BTA (Chapter 8).

1.4 List of Related Publications

During my doctoral work I co-authored several research publications connected with this thesis. Here I briefly list these publications and delineate my role. This dissertation primarily focuses on expanding the contents of paper 1 and 2.


I developed the models (initially by Walid Taha and Roland Philippsen) into the form described in this dissertation with help from Chad Rose, Marcia O’Malley. The implementation of compile-time transformation in Acumen are my work.

I design and implemented the binding time analysis algorithm as well as the partial evaluator in Acumen. I defined the operational semantics together with Walid Taha. All proofs included in this paper are my work, though advice was provided by Walid Taha, Ferenc A. Bartha and Eugenio Moggi.


The prototype implementation for detecting chattering behavior and the proposed runtime algorithm for generating chattering-free dynamics internally are my work.


I was involved in most aspects of the development of Acumen throughout the
course of my PhD studies and implemented the 3D visualization.

5. W. Taha, R. Cartwright, R. Philippsen, and Y. Zeng, “Developing a first course on cyber-physical systems,” in *Proceedings of the WESE’14: Workshop on Embedded and Cyber-Physical Systems Education*, p. 6, ACM, 2014 I was involved in the development of course project as well as the platform used in the course.


I supported several instances of the CPS course and developed a major part of the models used in the course. In addition, I was responsible for the integration of student’s feedbacks into Acumen.


I helped with dynamic and visualization part of the modeling process.
Part I

Modeling
Chapter 2

Background

Model-based tools have the potential to significantly improve the process of developing novel cyber-physical systems (CPS). This chapter presents the syntax and the informal semantics of Acumen, a minimalist language and compares it with other standard modeling and simulation tools that are widely used in the CPS domain.

2.1 Introduction

Model-based design tools can revolutionize the process of developing new products. This is especially the case for cyber-physical systems (CPSs), which consist of networks of computational and physical components. The presence of computational and networking components means a bigger state space and more non-determinism, both of which limit the utility of physical testing. This observation has also been made in the context of automotive systems with advanced driver assistance and autonomy functions [13],[12], and smart home design [14].

Model-based tools provide functions on models such as simulation, analysis, verification, and conformance testing [15, 16, 17, 3, 4]. Using such tools shifts the user’s focus from building and maintaining disjointed software artifacts like the “simulation code” or the “analysis code” to “the model of the product” being studied or developed. In addition to what is often described as “raising the level of abstraction,” the model-based approach reduces the pressure to maintain different (often implicit)
models of the same system. Avoiding such duplication reduces the work needed to ensure the consistency of these different models.

Modeling and simulation tools can improve software development for robotics in two ways. First, they can virtualize the robot and allow fast and cheap testing of real robots. Second, they make it easier to explore and gradually develop the specification of software and hardware before we start implementing it. For example, we can start with an idealized controller, gradually adding more realistic constraints such as quantization, discretization, and energy consumption.

The overarching goal of our research is to develop the semantic foundations for rigorous simulation languages for the cyber-physical systems domain (see, for example [18, 19, 20, 21, 22, 23]). Because semantics research must focus primarily on minimal calculi, it is critical that the choice of constructs in a calculus has domain-based justification that is grounded in an accurate understanding of the needs of the CPS domain. There is a need for guidelines for the demands of the CPS domain in terms of modeling language features, as well as evaluations of the extent to which particular features are able to meet these demands. Domain-based justification is of significance not only for our own research but for any other research effort on the semantics of hybrid systems modeling languages. Beyond the research community, such guidelines can also illuminate the design space for providers of such tools. It affects decisions about syntax, semantics, building implementations, and understanding how a language could and should be used. This thesis reports on the results of our efforts to address this question.

After introducing the syntax and the informal semantics of Acumen, a minimalist language used for the investigation (Section 2.2), we compare it with other standard modeling and simulation tools that are widely used in the robotics domain (Section
5.2). Then we turn to the different aspects identified and treated. Each section explains an aspect and how it can modeled in a minimal formalism. Simpler and more self-evident aspects are presented in earlier sections. Naturally, a larger part of these earlier sections is spent explaining the formalism than is the case in later sections.

Visual and geometric presentation is a critical aspect of analytical modeling that can hide in plain sight. We begin by showing how to support visual and geometric presentation in both static and dynamic scenes, as well as in simple and composite components (Section 3.1). Traditionally, visualization is not part of analytical modeling, but it is indispensable for efficiently understanding both the specification and the results of model-based simulations or analysis. Basic mechanics and dynamics come next (Section 3.2), along with a range of analytical principles used to model physical systems. After a brief introduction to the notion of sub-models (Section 3.3), we return to expressing the differential equations needed to model control (Section 3.4). We then consider how to express a more realistic model of control systems by capturing the way in which implementation on a digital computer introduces both discretization and quantization (Section 3.5).

The four larger case studies carried out as part of this work are drawn from the robotics domain. The first is a quadcopter, a non-trivial, rigid body system that is often used as a CPS example. This case study shows that the language can simply and directly express Newtonian models (Section 4.1). The second case study is a research robot called the RiceWrist-S. For this robot, developing a Newtonian model is difficult and inconvenient. The case study helps illustrate how the more advanced technique of Lagrangian modeling can be advantageous for some problems. As a prelude to modeling the RiceWrist-S robot [24], we consider two ostensibly simple
dynamic systems, namely a single pendulum and a double pendulum. For the second system, we show that Lagrangian modeling leads to a much simpler mathematical formalization (Section 4.2). We then confirm this insight by showing how Lagrangian modeling enables us to construct a simple model of the dynamics for the RiceWrist-S (Section 4.3). The third case study is an example of an integrated electromechanical system, namely a linear solenoid actuator. This case study illustrates the core language’s ability to model multiphysics, such as coupled mechanical and electrical subsystems. Whereas the first three case studies are continuous systems, the fourth study (Section 4.5), a compass gait biped, is a classic example drawn from the domain of robotic walking. It shows how to model hybrid systems with non-trivial dynamics using the core language. We summarize our observations about the needs of the CPS domain and language constructs (Section 9), and then conclude.

2.2 Acumen: A Small, Experimental Language for Hybrid Modeling

In this section we introduce the syntax and the (informal) semantics for the small language that we will use in the rest of this thesis.

The syntax for the Acumen language [31, 32] consists of the following constructs:

- Constant literal values (e.g., true, 5, 1.3, "Hello")
- Vectors and matrices (e.g., (1, 2), ((1, 2), (3, 4)))
- Expressions and operators on ground and composite types (+, -, ...)
- Model definitions (model C (x, y, z) = ... )
- Model instantiation and termination operations (create, terminate)
Table 2.1: Comparison of Modeling and Simulation Tools

<table>
<thead>
<tr>
<th></th>
<th>Partial derivatives</th>
<th>Compact derivatives</th>
<th>Support for equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB R2015b [25]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Simscape R2015b [4]</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Octave4.0 [26]</td>
<td>No</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>OpenModelica 1.9.3 [27]</td>
<td>Limited</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Dymola5.3 [28]</td>
<td>No</td>
<td>-</td>
<td>Yes</td>
</tr>
<tr>
<td>Mathematica 7.0 [29]</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>20-sim4.6 [30]</td>
<td>Yes</td>
<td>Unknown</td>
<td>Yes</td>
</tr>
<tr>
<td>This thesis</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
• Variable derivatives ($x’, x''$, ...) with respect to time

• Variable declarations (initially ... always). For convenience, we included in the set of variables a special variable called _3D for generating 3D animations.

• Time and partial derivatives (e.g., ($x^2$)', $(x+y)’ [x]$)

• Continuous equation and discrete equations (=, +=)

• Conditional constraints (if, and match)

Derivatives can be applied to expressions or variables. The time derivative on a variable 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 can be used to compute the value of the variable itself. In principle, one can imagine that, 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 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 can be used to compute the value of the variable itself. The partial derivative is an operator that takes two expressions and returns the result of the first expression differentiated with respect to the second expression. The syntax for this operator is `expr’ [expr]`. For two scalars, the result is simply the first expression partially differentiated with respect to the second one. If one expression is a scalar, and the other a vector, the operator is applied to all elements of the vector. For two vectors, a matrix is generated where every row is the result of the
corresponding element in the first vector, partially differentiated with the second vector. For example, let \( f = x + y \), \( g = (x^2, y^2) \) and \( q = (x, y) \), then the operators \( f'[x] \) and \( f'[y] \) would both result in 1. Moreover \( g'[x], f'[q], g'[q] \) will return \((2 \times x, 0), (1, 1)\) and \((2 \times x, 0), (0, 2 \times y)\) respectively. Allowing arbitrary expressions instead of just variables in partial derivatives allows us to express families of equations like the Euler-Lagrange equation directly.

Continuous equations are used for equating continuous behaviors (as in differential equations). Discrete equations are used to define values at discontinuities, where an instantaneous change of a value (a reset) can occur in juxtaposition to continuous dynamics. Initial values for variables (at the time of the creation of a new instance) are treated as discrete equations. Currently, we take a conservative approach to initial conditions, which requires users to express them explicitly even for variables where there is a continuous equation that may immediately override this explicit initial value. For the sake of minimality, Acumen has no special notation for introducing constants (in the sense of variables that do not change value over time).

This language is used for a term-long project in a course on CPSs [33], which has been enthusiastically received in the first four offerings (see for example [34, 35]). The parsimonious language seems to help students connect different concepts and avoid the introduction of artificial distinctions between manifestations of the same concept in different contexts. This bodes well for the utility of such a language for reducing the need for different notations to model the same system during different phases of a product development process. However, to fully overcome this challenge, we must develop a clear understanding of how different features in such a language match up with the demands of different types of cyber-physical systems.

The Acumen distribution contains implementations of differential equation solvers
for simulation, accessible from a “Semantics” menu. For this thesis, we pick the “Traditional” semantics, which simply uses a Runge-Kutta method and a constant time step for integration.

2.3 Related Work

Today, a wide range of tools exists to support the modeling and simulation of cyber-physical systems [15]. In this section we identify the main differences between these tools and the language that used in the present research. Table 5.1 provides an overview of how several related tools compare on key properties of relevance to the present work. The table summarizes the presence of the features discussed in this thesis in important modeling languages. We emphasize that Acumen is an experimental research prototype, and that the other languages have numerous features not present in Acumen. Furthermore, we have not evaluated how complex it would be to add such constructs to the existing languages. The properties are:

- Partial derivatives: The ability of a tool to support the expression of a partial derivative of two expressions directly.

- Compact derivatives: The ability to compute symbolic derivatives without producing exponentially large expressions.

- Support for equations: The ability to write equations instead of just assignments.

The main observations summarized in the table are as follows. Scripting or programming languages such as Octave [26], MATLAB, and its graphical extension Simulink, focus on providing a convenient programming language for a broad class of
scientific problems. However, these formalisms are not intended to capture equational mathematical models directly. For example, they limit the user to only expressing so-called causal models, which are not expressed as equations, but rather, as directed (continuous or discrete) assignments. The Simscape [4] language from MathWorks is based on MATLAB and is a dedicated textual language for modeling physical systems. It supports equations but not partial derivatives. To express equations that require partial derivatives, one must use another toolbox to compute the derivative symbolically and then copy and paste the result into the model manually [4]. Copying and pasting are problematic because they make it difficult to avoid code duplication in differentiation, and harm the traceability/maintainability of the code.

Symbolic algebra tools such as Mathematica [29] and Maple, in principle, can assist in manipulating analytical models and in making them executable. In practice, however, as the size of the physical system being modeled increases, symbolic computing systems can both take an exponentially long time to compute a symbolic result and produce a result that is exponentially larger than necessary. To avoid this problem, Acumen uses a specialized symbolic differentiation procedure that operates on partial derivatives as they appear in the context of the equations. The procedure computes compact derivatives by avoiding any inlining and by reusing any expressions either appearing in the original system of equations or that have been generated during symbolic differentiation anywhere in the system of equations. For many common situations, this procedure outperforms Mathematica by being polynomial both in time and size, with respect to size of the term being differentiated [18].

With regard to the discussion of Lagrangian modeling, the thesis includes a comparison between code in Acumen, MATLAB, and Modelica. The code for the latter two languages is included in the Supplemental Material (Section ??).
OpenModelica [27] and Dymola support equational modeling and provide many solvers for differential algebraic equation (DAE) systems. However, partial derivatives of arbitrary expressions are not directly supported in the current implementation of Modelica. While partial derivatives can be included by advanced Modelica users, for example, with inline semantics and the user-defined \texttt{pder()} operator [36], this indirect method does not lend itself to many modeling methods, such as the Euler-Lagrange.

The language that comes closest to supporting the language features emerging from our study is 20-sim. It is a commercial modeling and simulation language for multidomain dynamic systems. It supports both equational modeling and partial derivatives [30]. However, it is unclear whether the symbolic differentiation produces a compact result.

Finally, it is worth addressing the relation to the Robot Operating System (ROS), which is not addressed in Table 5.1. ROS is a middleware designed to be a communication interface for sending and receiving sensor data in many robotic devices [37]. It is concerned more with interactions between each individual component rather than the overall design of the mechatronic system. Thus, it does not help with modeling the physical context or the intended continuous dynamics of the system. Rather, it is intended to help with implementing real-time control of the hardware.
Chapter 3

Basic Aspects of Cyber-physical Systems

This chapter presents a number of aspects that are representative of some of the most basic and prominent modeling needs of the CPS domain.

3.1 Geometry and Visual Form

Most physical systems either take up space or have an effect on space. As a result, visual presentation has a role in CPS design. For many people, it is hard to imagine a design without conjuring an image of a general visual form. If we want to replace physical prototyping with virtual prototyping, visualization becomes a necessity. Animating the evolving state of a hybrid system using visual geometric presentation often reveals behaviors of the system that might otherwise be undetected.

A hybrid modeling and simulation language can be naturally extended with a lightweight mechanism for three-dimensional (3D) visualization [38]. In the core language of Acumen, the user can specify 3D visualizations through a special variable called _3D. This variable is read by the implementation and used to generate a dynamic 3D scene. In principle, any graphical rendering technology can be used by an implementation to realize these visualizations. The current implementation uses the jPCT library [39] and supports four primary objects: Sphere, Cylinder, Box, and Cone. The following Acumen model illustrates how simple a visualization primitive can be:
Figure 3.1: The 3D Output Generated for an Instance of the Model Sphere.

model sphere \((m,D)\) =

initially
\[ q = (0,0,1), \ _3D = () \]

always
\[ _3D = (\text{Sphere center} = D+q \]
\[ \text{size} = 0.03*\sqrt{m} \]
\[ \text{color} = (m/3,2+\sin(m),2-m/2)) \]

This is a custom-made model that a user may have created to represent a particle with a given mass and position. The parameter \(m\) represents the mass of the sphere. In this example the mass is reflected in the size and color of the sphere object displayed in the 3D view. The parameter \(D\) is a display reference point. Passing different \(D\) values to an individual instance facilitates creating visualizations where instances of the same model can appear in different places. The initially section declares variables present in each instance and their initial values at simulation time, when an instance of the model is created. The variable \(q\) represents the position of the sphere.
The special variable _3D must be bound to a vector with a format determined by the 3D visualization system used in our implementation of the core language. The continuous equation is computed for as long as the instance exists in simulation. The _3D vector has the following format. The first field, in this case Sphere, indicates the shape we want in the visualization. The second field is the coordinates for the center of the shape. The third field is the radius. In this example the user has chosen to make the radius a simple function of the mass. This function is not intended to have any physical meaning other than to produce an illustrative visualization. The next field is a vector that represents the red/green/blue (RGB) color components for this sphere, using an ad hoc formula to generate a color based on the mass. Fig. 3.1 depicts a visualization generated using this model.

We can create an instance of the model described above by writing $s = create\ sphere\ (5,(0,0,0))$ in the initialization (initially) section and then $s.q = (0.1,0.2,0.3)$ in the body. To generate 3D animations, all we have to do is to let the value of $q$ vary over time, as in the following code:

```plaintext
model moving_sphere (m, D) =
    initially s = create sphere(m,D),
    \hspace{1cm} t = 0, t' = 0
always
    t' = 5,
    s.q = (sin(t)\times\sqrt{1 - (\sin(t/10)^2))},
    \hspace{1cm} \cos(t)\times\sqrt{1 - (\sin(t/10)^2))},
    \hspace{1cm} \sin(t/10))

Here the variable $t$ and its derivative $t'$ are introduced to model a local variable that progresses at exactly five times the rate of time. All that is needed to accomplish
this is to include the equation $t'=5$. The time-varying variable $t$ is then used to generate some interesting values for the $X$, $Y$, and $Z$ components of the position field $q$ that represents the center of the sphere model $s$.

As noted earlier, we can place different instances of a model (such as `moving_sphere`) at different locations by varying the $D$ parameter. By changing the value of the position parameter $q$, we can create an animation with two spheres moving in a synchronized fashion.

It is useful to note that a 3D visualization facility can be used to visualize not only spatial parameters and dimensions but also abstract values such as energy. For example, it can be useful to define models that assist in visualizing such values during a simulation. The following code defines a model to visualize a scalar value as a cylinder, whose length is proportional to that value:

```plaintext
model display_bar (v,c,D) =
    initially
       _3D = ()
    always
       _3D = (Cylinder center=D+(0,0.2,v/2)
                radius=0.02 length=v  color=c
                rotation=(-1*pi/2,0,0))
```

Following the 3D primitive name `Cylinder`, the next argument represents the center of the cylinder. We take this to be $v/2$ because this will allow us to keep one end of the cylinder fixed as the value of $v$ changes. The next two arguments specify the radius and length of the cylinder. The next argument is `color`, and the last specifies orientation angles for the cylinder.
In addition to having a mechanism for specifying visual form, working with geometry places some intuitive but nevertheless specific requirements on the modeling languages. In particular, it is generally necessary to perform some vector and trigonometric calculation to create the desired shape. This need arises even in simple situations. An example of such a context is drawing a cylinder between two points. Often, visualizations cannot be specified directly because the underlying library uses a different approach to describe the orientation of a figure. In the case of cylinders, it is common to use polar coordinates (two angles) to specify the orientation of the axis of a cylinder. Once we have figured out all necessary calculations, they can be encapsulated in one model as follows:

model cylinder (D) =

initially
q1 = (0,0,0), q2 = (0,0,0),
dis = (0,0,0), r = 0.01,
l = 0.01, alpha = 0, theta = pi/2,
x = 0, y = 0, z = 0, _3D = ()

always
dis = q1 - q2,
x = dis(0), y = dis(1), z = dis(2),
l = norm(q1-q2), alpha = asin(z/l),
if y>0 then
theta = asin(x/(l*cos(alpha)))
else
theta = -asin(x/(l*cos(alpha)))+pi,
_3D = (Cylinder center=(q1+q2)/2)+D
The *dot* and *norm* operators compute the dot product and the vector norm (or length). Creating such a model is a good exercise in making customized building blocks for visualization.

If we are used to programming in a typical programming language where such functions are present, the convenience of such operations is no surprise. From the semantic point of view, what is significant is their necessity. The necessity of supporting transcendental functions means that approximating the set of real numbers with the set of rationals is not possible, as the results of transcendental functions are not rational. In addition to having significant implications for representability and computability of approximations to these operations, their presence implies that nonlinearities are hard to avoid when working with general physical systems. All of these complications arise even before there is any consideration of dynamics or a time dimension in the description of physical systems.

### 3.2 Particle Dynamics and Impacts

The most basic approach to model the mechanical dynamics of a system is to view it as a point mass, or a *particle*. In contrast to the syntax needed to describe geometric and visual objects, describing particle dynamics can be done more concisely.

A point mass that can only move in one dimension can be represented as follows:

```
model mass_1d (m, q0, D) =
initially
```
\[ q = q_0, \quad q' = 0, \quad q'' = 0, \]
\[ f = 0, \quad e_k = 0, \]
\[ s = \text{create sphere} \ (m, D) \]
always
\[ q'' = \frac{f}{m}, \]
\[ e_k = 0.5 \times m \times (q')^2, \]
\[ s.q = (0, 0, q) \]

initial position \( q_0 \), and a reference point for visualization. Internally, the mass keeps track of a position \( q \), its first and second derivatives \( q' \) and \( q'' \), a force \( f \), and the kinetic energy \( e_k \). For visualization, a sphere instance is created during initialization. The body of the model definition specifies that the acceleration of the object, \( q'' \), is determined by Newton’s law \( F = ma \), where we are solving for acceleration (which is just \( q'' \) here). Finally, we set the position \( q \) of the visual sphere instance to be the same as the position \( q \) of the current instance.

Supporting vector operations makes it possible to define a similar model that has a three-dimensional position almost as simply:

\[ \text{model mass} \ (m, q_0, D) = \]
\[ \text{initially} \]
\[ q=q_0, \quad q'=(0,0,0), \quad q''=(0,0,0), \]
\[ f=(0,0,0), \quad e_k=0, \]
\[ s = \text{create sphere} \ (m, D) \]
always
\[ q'' = \frac{f}{m}, \]
\[ e_k = 0.5 \times m \times (\text{dot}(q', q'))^2, \]
Note that it is convenient in technical discourse in science and engineering to refer to the derivatives of vectors (and not just scalars) in specifications of dynamics. We can induce continuous behaviors in such models by means of an external continuous equation. For example, the effect of a gravitational force on a mass \( m \) by a continuous equation \( m \cdot f = m \cdot (0, 0, -9.81) \). The expression for energy uses the built-in dot product operation on vectors. An idealized 3D spring can be modeled as follows:

```plaintext
model spring (k, L0, D) =
  initially
  q1 = (0, 0, 0), q2 = (0, 0, 0),
  f1 = (0, 0, 0), f2 = (0, 0, 0),
  dl = (0, 0, 0), e_p = 0
  always
  dl = q2 - q1 * (1 - L0 / norm(q2 - q1)),
  f1 = k * dl, f2 = -k * dl,
  e_p = 0.5 * k * dot(dl, dl)
```

This model associates a different force with each end of the spring. It computes a potential energy \( e_p \) rather than a kinetic energy. No visualization is included in this model, but this can be easily achieved using the techniques presented above. An important physical effect in dynamics is impact, often modeled as a sudden change. Discrete equations can be used for this purpose. The following model provides an example of the use of discrete equations to model a classic hybrid system, the bouncing ball:

```plaintext
model bouncing_ball (D) =
```
initially
m = create mass_1d (10, 2,D),
bk = create display_bar
    (0,(3,0.2,0.2),D+(0.1,0.2,0)),
bp = create display_bar
    (0,(0.2,3,0.2),D+(-0.1,0.2,0)),
bt = create display_bar
    (0,(0.2,0.2,3),D+(0,0.2,0))
always
m.f = m.m * -9.81,
if (m.q < 0 && m.q’ < 0) then
    m.q’ += -0.9 * m.q’
noelse,
bk.v = m.e_k / (m.m * 9.81),
bp.v = (m.m * 9.81 * m.q)
    / (m.m * 9.81),
bt.v = bk.v + bp.v

The model uses the mass model along with a continuous gravity model and a ground-impact model, where the ball loses 10% of its velocity. The model `display_bar` is used to display colored bars to present some additional information in the 3D output. The mass model used here has only one degree of freedom along the Z axis. We use three display bars to visually represent the kinetic and potential energy, as well as their sum. The discrete equation occurs inside the `if` statement that detects impact with the ground plane. Fig. 3.2 shows a sequence of screenshots, one including the Integrated Development Environment (IDE), which results from running this
Figure 3.2: The IDE of the Implementation with the Bouncing Ball Model and Simulation Results.
example. It can be seen that, as expected, the total energy decreases at each impact, while the kinetic and potential energies reach their respective maxima and minima at the height of the bounce and the impact at ground level.

Thus, to describe even the basic Newtonian dynamics for simple particles, there is a need for not only ordinary differential equations (ODEs) but also hybrid ODEs. It is worth emphasizing that modeling these seemingly elementary aspects of physical systems necessitates the support of continuous equations, derivatives, discrete equations, and conditionals.

### 3.3 Composite Models

Most systems are composite, in that they consist of smaller, interacting components. We now consider the specific requirements that the need for modeling such compositions introduces. The most elementary requirement is that of a mechanism to connect components by relating fields in different components though continuous equations. For example, the following model specifies a system consisting of three masses connected by two springs. Note that the model uses an instance of the model `display_bar` to draw a cylinder to display the kinetic energy in the system.

```plaintext
model example_3 (D) =

initially

m1 = create mass_1d (30,1,D),
m2 = create mass_1d (10,-1,D),
m3 = create mass_1d (5,-1.5,D),
s1 = create spring (5,1.75,D),
s2 = create spring (5,0.5,D),
b = create display_bar(0,(0.1,3,0.1),D),
```

always
\[ s_1.q_1 = m_1.p, \quad s_1.q_2 = m_2.p, \]
\[ s_2.q_1 = m_2.p, \quad s_2.q_2 = m_3.p, \]
\[ m_1.f = s_1.f_1, \]
\[ m_2.f = s_1.f_2 + s_2.f_1, \]
\[ m_3.f = s_2.f_2, \]
\[ b.v = (m_1.e_k + m_2.e_k + m_3.e_k + s_1.e_p + s_2.e_p) \times 12 \]

This example uses the mass and spring model. The system consists of three masses and two springs as shown in Fig. 3.3. The interaction between them can be expressed through external continuous equations. For example, the effect of the two spring forces acting upon mass \( m_2 \) is captured by a continuous equation \( m_2.f = s_1.f_2 + s_2.f_1 \).

Some modeling languages such as Modelica support more sophisticated mechanisms for expressing connections. In particular, they support notions of connectors that can serve as sum-equalizing or value-preserving junctions. These notions can make it more intuitive to model systems built by connecting more basic components. However, they do not appear to provide the ability to express connections that cannot be expressed using the more elementary notion of equality presented here. As such, they do not have a natural place in a minimal formalism for modeling cyber-physical systems. Their utility, however, suggests that additional syntactic extensions can
make such a minimal formalism more convenient for practical modeling.

### 3.4 Control

A pervasive aspect of physical systems is the presence of one or more mechanisms to drive their operation to a certain goal or function. The basic role of control is to bring a certain quantity close to a desired target by manipulating the value of some parameters that affect this quantity. For the model presented above, and given a controller instance `c`, control can be introduced as follows:

```plaintext
// Goal is spring length at rest
c.r = s1.l+s2.l,

// Value is actual spring length
   c.y = m1.p-m3.p,

// Add c.u
   m1.f = s1.f1 + c.u, m2.f = s1.f2 + s2.f1,

// Subtract c.u
   m3.f = s2.f2 - c.u;
```

In this model the goal value for the controller is to have the length of the system be the same as the natural lengths of the two springs. The quantity that we wish to control is the position of the first mass minus the position of the third one. The way we will achieve this is to take a force value `u` that is generated by the controller, and apply it to both sides of the system that we have constructed, but in opposing directions.

Now the question that remains is how the controller `c` should compute its output force `u`, given the goal `r` and measured value `y`. This is a prototypical question in the
design of control systems, and one that can be approached in a variety of different ways. Three of the most basic types of controllers are 1) proportional feedback, 2) proportional/differential feedback, and 3) proportional/integral/differential feedback. The first type can be used successfully in some systems, such as those governed by first-order differential equations, or higher-ordered systems that can dissipate energy. It can be modeled as follows:

```plaintext
model controller_p (k_p) =
initially
  r = (0,0,0), y = (0,0,0), u = (0,0,0),
  e = (0,0,0)
always
  e = r - y, u = k_p * e
```

The force $u$ computed is directly proportional (hence the name) to the error term $e$, which is the difference between the goal $r$ and current value $y$ of the quantity that we want to control. The higher the constant $k_p$, the higher the force that will be applied for the same amount of difference between the goal value and the current value.

If the system has inertia or does not dissipate the extra energy introduced by the control force, it might oscillate indefinitely as a result of the proportional control. To deal with this problem, a slightly more sophisticated controller can add a force opposing the direction of the motion (or rate of change) of the value being measured. Such a proportional/differential (PD) controller can be defined as follows:

```plaintext
model controller_pd (k_p, k_d) =
initially
```
\[ r = (0,0,0), \quad y = (0,0,0), \quad u = (0,0,0), \]
\[ e = (0,0,0), \quad r_{\dot{}} = (0,0,0), \]
\[ e_{\dot{}} = (0,0,0), \quad y_{\dot{}} = (0,0,0) \]
always
\[ e = r - y, \]
\[ e_{\dot{}} = r_{\dot{}} - y_{\dot{}} , \]
\[ u = k_p \times e + k_d \times e_{\dot{}} \]

Note that this controller has two extra fields, \( r_{\dot{}} \) and \( y_{\dot{}} \), that should be provided from outside the model to serve as the speed reading that should affect the final force \( u \).

An interesting feature of the first two types of controllers described above is that they do not keep track of history. We may wish to build a controller that exerts a higher force only after a weaker force has been tested for some time. This can be helpful, for example, if there are external constant forces (such as gravity) acting on our system, and we do not know their precise quantities ahead of time. This type of behavior can be achieved by adopting a proportional/integral/differential (PID) controller such as the following:

model controller_pid (k_p, k_i, k_d) =
initially
\[ r = (0,0,0), \quad y = (0,0,0), \quad u = (0,0,0), \]
\[ e = (0,0,0), \quad r_{\dot{}} = (0,0,0), \]
\[ e_{\dot{}} = (0,0,0), \quad y_{\dot{}} = (0,0,0), \]
\[ e_i = (0,0,0), \quad e_i' = (0,0,0) \]
always
\[ e = r - y, \]
\[ e_{dot} = r_{dot} - y_{dot}, \]
\[ u = k_p \times e + k_d \times e_{dot} + k_i \times e_i, \]
\[ e_i' = e \]

The variable \( e_i \) is being used to integrate the difference between the goal \( r \) and the value \( y \) over time, so no extra inputs are needed.

Using the formalism presented above, it is easy to model several instances of the three-mass/two-spring example, and to visualize the result of simulating it by showing both the behavior of the mass and the energy of the system with different controllers. The controllers presented here illustrate the design of basic, idealized control. The language is also expressive enough to capture more realistic models, such as when a system does not have velocity as input but rather uses discrete sampling of position to compute an estimate of velocity that is then used in the PD controller. The experiment shows that a P controller will not dissipate any energy and therefore will not stabilize the system. In fact, at times it will add energy to the system and at others absorb energy from it. This example motivates formally analyzing this system to show that this controller will function essentially as simply another spring between the two extreme masses. The PD controller will suffice in stabilizing the system quickly, and this will be clear from the height of the bar representing the energy in the system.

From the point of view of domain needs, it appears that high-level modeling of controllers as continuous systems does not introduce additional demands on the modeling language beyond that presented above for particle dynamics and composite models. In particular, support for ODEs suffices. More sophisticated controllers can employ different control dynamics in different modes, in which case the full features
needed to model hybrid ODEs would be useful.

### 3.5 Discretization and Quantization

The one aspect of controllers that we have not captured in the models presented above is that they are generally implemented by digital computers. The introduction of digital components in a physical setting introduces a need for both discretization and quantization. Both notions involve mapping the set of reals into a discrete set, such as the naturals or an isomorphic set. The term ‘discretization’ refers to performing this type of operation on quantities representing time, whereas ‘quantization’ refers to performing it on other values. Both effects can be concisely expressed with few additional requirements on the modeling formalism. To model discretization, the key mechanism needed is to define a local clock and to allow actions to be performed or observed only at clock transitions. The following model describes a proportional/integral/differential feedback (PID) controller (like the one presented above) with discretization and quantization effects.

```plaintext
model force_controller_pid_d
    (k_p, k_i, k_d, period) =

    initially
    r=(0,0,0), y=(0,0,0), s=(0,0,0),
    u=(0,0,0), t = 0, t' = 0,
    e=(0,0,0), sensor = (0,0,0)
    r_dot=(0,0,0), y_dot=(0,0,0),
    e_i=(0,0,0), e_i'=(0,0,0)

    always
    t' = 1,
```
if (t>period) then
    t = 0,
    sensor = floor(y*10)/10,
    u = k_p*e + k_i*e_i - k_d*e_dot,
noelse,
    e = r - sensor, e_dot = r_dot - y_dot,
    e_i' = e

The variables \( t \) and its derivative \( t' \) are used in a manner similar to that performed earlier in this thesis to generate an interesting signal for a moving sphere. Here we do two new things with the variable \( t \). Firstly, we have a conditional statement based on this variable that waits until \((t > \text{period})\). The parameter \( \text{period} \) models the time it takes the particular microprocessor that implements our controller to produce the new value of the result of the controller. Once the condition is true the first thing we do is reset the counter. The second action is to reset its value to 0 using the statement \( t=0 \) as soon as that condition is true. In addition to this reset, the conditional also allows the equation for the variable \( u \) in the original model to take effect only for that instant when \( t \) has surpassed the value of \( \text{period} \). Because no other definition is given for this value until this event occurs again (at the start of the next period), the value \( u \) remains constant until that change occurs. With this model, it is easy to illustrate that, as the sampling period goes up, the system that we are trying to control can become unstable. Quantization can be modeled using the floor function that quantizes the value of input signal \( y \) by letting \( \text{sensor} = \text{floor}(y*10)/10 \) as shown above. Quantization can also be modeled using a conditional statement that updates the value of \( \text{sensor} \) up or down by a fixed amount, based the parameter \( \text{quanta} \), as shown below. We have one
conditional statement that waits until \((\text{sensor} + \text{quanta} < y)\), then increases the value of \text{sensor} by \text{quanta}. Similarly, there is another conditional statement that decreases the value of \text{sensor} by the same amount.

\[
\ldots
\]

\[
\text{if } (\text{sensor} + \text{quanta} < y) \text{ then }
\]

\[
\text{sensor}^+ = \text{sensor} + \text{quanta}
\]

\[
\text{else if } (\text{sensor} - \text{quanta} > y) \text{ then }
\]

\[
\text{sensor}^+ = \text{sensor} - \text{quanta}
\]

\[
\text{noelse,}
\]

\[
\ldots
\]

Simple equations using \textit{if} statements and increment/decrement operations suffice to express the operations of rounding functions. Thus, there is again no additional expressivity here beyond the operators we have already introduced for modeling hybrid systems.

We now consider larger case studies to gain better understanding of the practical expressivity of this core set of language constructs.
This chapter considers four, more complex, case studies from the domain of robotics. The first, a quadcopter, illustrates that these constructs can support the modeling of interesting systems. The second, a serial robot, provides a concrete example of why it is important to support static partial derivatives, namely, that it significantly improves the way models of rigid body dynamics can be expressed. The third, a linear solenoid actuator, illustrates the language's ability to integrate multiphysics subsystems. The fourth and final, a compass gait biped, shows how a hybrid system with non-trivial dynamics is modeled.

4.1 Case Study I: Quadcopter

A rigid body system consists of a set of solid bodies with well-defined masses and inertias, connected by constraints on distances and/or angles between the solid bodies. The dynamics of many robotic systems can be modeled with reasonable accuracy as rigid body systems. It is widely used for describing road vehicles, gear systems, walking bipeds, etc. In this section, we consider an example of a complex system that can be successfully modeled as a simple rigid body, namely, the quadcopter.

4.1.1 Background

The quadcopter is a popular mechatronic system with four rotor blades to provide thrust. This robust design has seen use in many UAV applications, such as surveil-
Figure 4.1: Free Body Diagram of the Quadcopter.

lance, inspection, and search and rescue. Modeling a quadcopter is technically challenging, because it consists of a close combination of different types of physics, including aerodynamics and mechanics. A mathematical model of a quadcopter may need to address a wide range of effects, including gravity, ground effects, aerodynamics, inertial counter torques, friction, and gyroscopic effects.

4.1.2 Reducing Model Complexity Through Control

Even if we limit ourselves to considering just six degrees of freedom (three for position and three for orientation), the system is underactuated (one actuation from each rotor vs. six degrees of freedom) and is therefore not trivial to control. Fortunately, controllers exist that can ensure that actuation is realized by getting the four rotors to work in pairs, to balance the forces and torques of the system. With this approach, the quadcopter can be usefully modeled as a single rigid body with mass and inertia, by taking account of abstract force, gravity, and actuation control torques. This model is depicted in Fig. 4.1.
4.1.3 Mathematical Model

To generate the equations for the dynamics of our common quadcopter model [40], we first construct the rotational matrix to translate from an inertial (globally-fixed) reference frame to the body-fixed reference frame shown in Fig. 4.1. This matrix represents rotation about the $y$ axis ($\theta$), followed by rotation about the $x$ axis ($\phi$), and then rotation about the $z$ axis ($\psi$).

\[
R = \begin{bmatrix}
  c_\phi c_\theta & c_\psi s_\theta c_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\
  s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\
  -s_\theta & c_\theta s_\phi & c_\theta c_\phi
\end{bmatrix} \tag{4.1}
\]

Here, $c$, $s$ and $t$ refer to $\cos$, $\sin$, and $\tan$, respectively. Next, summing forces on the quadcopter results in:

\[
\sum F = m\ddot{a} = G + RT \tag{4.2}
\]

where $G$ is the force due to gravity, $R$ is the rotational matrix, and $T$ is the thrust from the motors. This expands to:

\[
\begin{bmatrix}
  \ddot{x} \\
  \ddot{y} \\
  \ddot{z}
\end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix}
  c_\psi s_\theta c_\phi + s_\psi s_\phi \\
  s_\psi s_\theta c_\phi - c_\psi s_\phi \\
  c_\theta c_\phi
\end{bmatrix} \tag{4.3}
\]

Finally, by summing moments about the center of mass, the equations for the dynamics for each of the rotational degrees of freedom can be determined as follows:
\[
\begin{bmatrix}
\ddot{\phi} \\
\ddot{\theta} \\
\ddot{\psi}
\end{bmatrix} =
\begin{bmatrix}
0 & \dot{c}_\phi t_\theta + \dot{\theta} \frac{s_\phi}{c_\theta} & -\dot{\phi} s_\phi c_\theta + \dot{\theta} \frac{c_\phi}{c_\theta} \\
0 & -\dot{\phi} s_\phi & -\dot{\phi} c_\phi \\
0 & \dot{c}_\phi \frac{s_\phi}{c_\theta} + \dot{\theta} \frac{t_\phi}{c_\theta} & -\dot{\phi} \frac{s_\phi}{c_\theta} + \dot{\theta} \frac{c_\phi}{c_\theta}
\end{bmatrix}
\begin{bmatrix}
\nu
\end{bmatrix}
\]

\[+ W_\eta^{-1} \dot{\nu} \quad (4.4)\]

Where

\[\nu = \begin{bmatrix}
p \\q\\r
\end{bmatrix} = W_\eta \begin{bmatrix}
\dot{\phi} \\
\dot{\theta} \\
\dot{\psi}
\end{bmatrix} = \begin{bmatrix}
1 & 0 & -s_\theta \\
0 & c_\phi & c_\theta s_\phi h_i \\
0 & -s_\phi & c_\theta c_\phi
\end{bmatrix} \begin{bmatrix}
\dot{\phi} \\
\dot{\theta} \\
\dot{\psi}
\end{bmatrix} \quad (4.5)\]

\[
\dot{\nu} = \begin{bmatrix}
(I_y y - I_z z) \frac{q r}{I_{x z}} \\
(I_z z - I_x x) \frac{q r}{I_{y y}} - I_r \\
(I_x x - I_y y) \frac{p r}{I_{z z}}
\end{bmatrix} = \begin{bmatrix}
\frac{q}{I_{x x}} \\
\frac{-p}{I_{y y}} \\
0
\end{bmatrix} \omega_i + \begin{bmatrix}
\frac{\tau_\phi}{I_{x x}} \\
\frac{\tau_\theta}{I_{y y}} \\
\frac{\tau_\psi}{I_{z z}}
\end{bmatrix} \quad (4.6)
\]

\[
\begin{bmatrix}
\tau_\phi \\
\tau_\theta \\
\tau_\psi
\end{bmatrix} = \begin{bmatrix}
lk (-\omega_2^2 + \omega_3^2) \\
lk (-\omega_1^2 + \omega_3^2) \\
\sum_{i=1}^{4} \tau_{M_i}
\end{bmatrix} \quad (4.7)
\]

### 4.1.4 Acumen Model for Quadcopter

The equations derived earlier for the dynamics can be expressed simply in the core language for hybrid systems, and are included with the supplementary materials. Fig. 4.2 presents snapshots of a 3D visualization of the quadcopter responding to a signal from a basic stabilizing controller [40]. Here the controller is bringing the
quadcopter from an initial setting, indicated by the green sphere, to the desired height and to the roll, pitch, and yaw angles of zero. Yellow arrows attached to each rotor indicate the relative thrust. This example shows that the Acumen core language can model the dynamics of non-trivial robots that are widely used in both research and education today.

## 4.2 Lagrangian Modeling, and Why we Need it

While the quadcopter case study does not point to the need for additional constructs, working with other case studies does reveal this need. Mathematical modeling of rigid body systems draws heavily on the field of classical mechanics. The goal is to derive the mathematical expression of the system’s dynamics. It so happens that with the quadcopter the Newton method for analysis is convenient. However, as we will see in this section, there are many systems for which other methods are noticeably more convenient.

The Newton method is focused on taking into consideration the forces and torques operating on a rigid body, and then computing the linear and angular accelerations of the center of mass of that body. In general, this method consists of isolating the rigid body of interest in a free body diagram, selecting coordinate frame and summing forces and torques on the body with respect to that frame, then using kinematics to express the linear and angular acceleration terms, before finally deriving the equations for the dynamics. Since the Newton method requires each rigid body to be isolated, forces and torques are modeled explicitly for the interfaces between them in multi-body systems, thus yielding models where forces are readily available for inspection and/or analysis.

It is standard practice for mechanical engineers to use the Lagrangian method to
analyze rigid body systems when modeling internal forces between rigid bodies are not
the focus of the investigation. However, supporting this method requires constructs
beyond the ones discussed so far. The Lagrangian method is based on the notion of
a function called Lagrangian \( L = T - V \), which is the difference between the kinetic
energy \( T \) and potential energy \( V \). In Lagrangian modeling of physical systems, this
condition should be seen as the analogy of the combined conditions \( \Sigma F = ma \) and
\( \Sigma \tau = I \omega'' \) in Newtonian mechanics. The Euler-Lagrange equation is as follows:

\[
\forall i \in \{1...|q|\}, \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = Q \tag{4.8}
\]

The \( \forall \) quantifier is used to introduce the index variable for a family of equations.
In the Acumen syntax, the keyword \texttt{foreach} represents this quantifier. The name
contained in the \( i \)th element of the tuple is looked up. \( Q \) represents non-conservative
forces (e.g., friction) acting on the system. Using just this equation, the modeling
process is reduced to specifying the kinetic and potential energy in the system. Part
of the power of the method comes from the fact that this can be done using Cartesian,
polar, spherical, or any other generalized coordinates. The Euler Lagrange equation
makes it possible to describe large and complex systems in a modular fashion, since all
that is needed for each new object is an expression for both its potential and kinetic
energy. For example, this method makes modeling the interactions between electrical
and mechanical subsystems much more straightforward than with other methods.
Compared to the classic Newtonian, force-vector based methods, the Euler-Lagrange
equation can provide a more direct specification of the dynamics, particularly in
systems with coupled dynamics. The Lagrangian modeling process consists of four
steps:

1. Start with a description of the components of the system, consisting of rigid
bodies and joints. This description generally comes with a set of variable names which are collectively called the generalized coordinates vector \( q \). While more commonly associated with Newtonian modeling, a basic free body diagram can be an intuitive way to capture the potential and kinetic energy information, as well as assisting in judiciously choosing the generalized coordinates.

2. Determine the expression for the total kinetic and potential energy \( T \) and \( V \), respectively, of the system, in terms of \( q \).

3. Identify and include any external forces \( Q \), such as friction.

4. Substitute the values into the Euler-Lagrange equation (4.8).

This process and its benefits can be illustrated with two small examples. The benefits apply whether or not the language supports directed or undirected equations. Fig. ?? presents a free body diagram marked up with generalized coordinates (\( \theta \) in one, and \( \theta_1, \theta_2 \) in the other) for a single and a double pendulum system. First, we consider the single pendulum. A direct application of the angular part of Newton’s law gives us the following equation:

\[
\ddot{\theta} = \frac{g}{l} \cos \theta
\] (4.9)

which can be easily expressed in Acumen.

Lagrangian modeling can be applied to the single pendulum problem, but Newton’s method works well enough here. However, Lagrangian modeling does pay off for a double pendulum. It is instructive for language designers to recognize that such a seemingly small change in the complexity of the rigid body makes the model that most of us learn about in high-school much more cumbersome than necessary. Whether or not this difficulty in modeling is due to a weakness in Newtonian modeling or intrinsic
complexity in this seemingly simple example is not obvious: The double pendulum is sophisticated enough to be widely used to model a human standing or walking [41], or a basic two-link robot such as the MIT-Manus [42].

To derive a model for the double pendulum using Lagrangian modeling, we proceed as follows:

1. A minimal set of generalized coordinates is chosen, \( q = (\theta_1, \theta_2) \), for this case it is the angle related to the pose of each link of the pendulum.

2. The kinetic and potential energies are defined as follows:

\[
T = \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2
\]

\[
V = m_1 g y_1 + m_2 g y_2
\]  

where we have introduced shorthands for velocities \( v_1^2 = l_1^2 \dot{\theta}_1^2 \) and \( v_2^2 = v_1^2 + \frac{1}{2} m_2 (l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_2 - \theta_1)) \) and positions of center of mass \( y_1 = \ldots \)
\[ y_2 = y_1 + l_2 \sin \theta_2 \] Substituting these terms we get:

\[
T = \frac{1}{2} m_1 (l_1 \dot{\theta}_1)^2 \\
+ \frac{1}{2} m_2 (l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_2 - \theta_1)) \tag{4.12}
\]

\[
V = m_1 g l_1 \sin \theta_1 + m_2 g l_2 \sin \theta_2 + m_2 g l_1 \sin \theta_1; \tag{4.13}
\]

3. We assume frictionless joints, and so there are no external forces \((Q = 0)\).

4. A Euler-Lagrange equation (4.8) is written in Acumen syntax.

The equations derived earlier for the dynamics can be expressed in our core language as follows:

```acumen
model double_pendulum(m_1, m_2, L_1, L_2, g) =

initially
\[
t_1 = 0, t_2 = 0, \\
t_1' = 0, t_2' = 0, \\
t_1'' = 0, t_2'' = 0, \\
q = (0,0)
\]```

Figure 4.4: The RiceWrist-S, with Superimposed Axes of Rotation.
always

\[
q = (t_1, t_2),
\]

\[
T = 0.5\times m_1 \times (l_1 \times t_1')^2 + 0.5\times m_2 \times (l_1 \times t_1')^2 + l_2^2 \times t_2'^2 + 2 \times l_1 \times l_2 \times t_1' \times t_2' \times \cos(t_2 - t_1),
\]

\[
V = m_1 \times g \times l_1 \times \sin(t_1) + m_2 \times g \times l_2 \times \sin(t_2) + m_2 \times g \times l_1 \times \sin(t_1),
\]

\[
L = T - V,
\]

\[
\begin{align*}
\text{foreach } i \text{ in length}(q) \\
L'[(q(i))']' - L'[q(i)] = 0
\end{align*}
\]

Here \( L'[q(i)] \) is the Acumen syntax for partial derivatives of \( \frac{\partial L}{\partial q_i} \) and \( L'[(q(i))']' \) is the syntax for \( \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) \).

To illustrate why supporting partial derivatives and equations in the language can make the modeling process easier, let us consider the same double pendulum model in MATLAB and OpenModelica. As already shown in Section 5.2, MATLAB does not support writing equations directly. In order to use the ODE solver it provides, one has to manually differentiate the Euler-Lagrange equations and transform them into the explicit ODE form. In OpenModelica, writing equations directly is supported, however, no partial derivatives can appear in the equations. Thus, it is the responsibility of the user to eliminate the partial derivatives and transform the Euler-Lagrangian equations into differentiation algebraic equation from. The complete model in MATLAB and OpenModelica can be seen in Supplemental Material (Section ??).
4.3 Case Study II: The RiceWrist-S robot

Engineers utilize Lagrangian modeling in the manner presented above to model multi-link robots much more directly than with the Newtonian method. In this section we present one such case study, using the RiceWrist-S research robot.

4.3.1 Background

Each year, approximately 795,000 people suffer a stroke in the United States, where stroke injuries are the leading cause of long-term disability. The RiceWrist-S is a research robot designed to assist in the rehabilitation of the wrist and forearm after neurological injuries such as stroke (Fig. 4.4). It consists of a revolute joint for
each of the three degrees of freedom at the wrist. A good starting point for its
dynamic modeling is the gimbal, a commonly studied mechanical device that, like
the RiceWrist-S, features several rotational axes intersecting at one point.

4.3.2 Analytical Model

We can apply the Lagrangian modeling process to determine the dynamics of a gimbal
as follows:

1. We take \( q = (\theta_1, \theta_2, \theta_3) \), where each of the angles corresponds to one of the
three rotations possible in the RiceWrist-S (Fig. 4.5). We choose to represent
the mass of the system as centralized to three locations, one at the origin, one
at the bottom of the outermost ring, and one at the end of the third link. The
masses in this figure correspond to the motors and handle depicted in Fig. 4.4.

2. To describe the energies concisely, it is convenient to use the following angular
velocities of the gimbal frames in the kinetic energy terms, and the resulting
heights for the potential energy terms:

\[
\begin{align*}
\omega_1 &= \dot{\theta}_1 \cdot \hat{z}_0 \\
\omega_2 &= \dot{\theta}_1 \cdot \hat{z}_0 + \dot{\theta}_2 \cdot \hat{z}_1 \\
\omega_3 &= \dot{\theta}_1 \cdot \hat{x}_1 + \dot{\theta}_2 \cdot \hat{z}_1 + \dot{\theta}_3 \cdot \hat{z}_2
\end{align*}
\]

where \( \hat{x}_i, \hat{y}_i, \hat{z}_i \) refers to the unit vector and coordinate frame about which these
rotations occur, as shown in Fig. 4.5.

Here, the \( \omega_i \) terms correspond to the \( m_i \) masses, and describe the angular ve-
locities of each mass. Since this is a rotational system, many of the rotations do
not occur in the coordinate frames of the respective gimbal. Therefore, in order
Table 4.1: Denavit Hartenberg Parameters.

<table>
<thead>
<tr>
<th>Joint</th>
<th>rot(x)</th>
<th>tr(x)</th>
<th>rot(z)</th>
<th>tr(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forearm</td>
<td>0</td>
<td>0</td>
<td>\theta_1</td>
<td>0</td>
</tr>
<tr>
<td>Wrist F/E</td>
<td>$-\frac{\pi}{2}$</td>
<td>0</td>
<td>\theta_2</td>
<td>0</td>
</tr>
<tr>
<td>Wrist R/U</td>
<td>$-\frac{\pi}{2}$</td>
<td>0</td>
<td>\theta_2</td>
<td>0</td>
</tr>
</tbody>
</table>

to express each \( \omega_i \) in terms of the same coordinate frame, we apply the following coordinate transforms, which follow the Denavit-Hartenberg convention:

\[
\omega_1 = \dot{\theta}_1 \cdot \hat{z}_0
\]

\[
\omega_2 = \dot{\theta}_1 \cdot \hat{z}_0 + T_{0}^{1} \cdot \dot{\theta}_2 \cdot \hat{z}_1
\]

\[
\omega_3 = \dot{\theta}_1 \cdot \hat{z}_0 + T_{0}^{1} \cdot \dot{\theta}_2 \cdot \hat{z}_1 + T_{0}^{1} T_{1}^{2} \cdot \dot{\theta}_3 \cdot \hat{z}_2
\]

where the elements of \( T_{i}^{i-1} \) are given in Table 4.1 and each coordinate rotation is defined as:

\[
T_{i}^{i-1} = \begin{bmatrix}
    c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} \\
    s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} \\
    0 & s_{\alpha_i} & c_{\alpha_i}
\end{bmatrix}
\]

(4.20)

Note that these coordinate transforms are essential for the standard development of multi-link robot kinematics. Since the gimbal’s frames are all coincident, only the rotation about the individual \( x \) and \( z \) frames, \( \alpha_i \) and \( \theta_i \), respectively, can be non-zero. Next, we express the heights above the predefined plane of zero potential energy (in Fig. 4.5, the plane perpendicular to \( x_0 \)) of each of the
masses $m_1, m_2, m_3$, respectively, as the following:

\begin{align*}
    h_1 &= l_2 (1 - \cos(\theta_1)) \quad (4.21) \\
    h_2 &= 0 \quad (4.22) \\
    h_3 &= -l_3 \cos(\theta_1) \sin(\theta_3) \quad (4.23)
\end{align*}

With this completed, the $T$ and $V$ terms can be quickly and easily defined.

Since this is a purely rotational-only system, $T$ is defined as the sum of the rotational energy terms, shown below:

\[ T = \frac{1}{2} (I_1 \omega_1 \cdot \omega_1 + I_2 \omega_2 \cdot \omega_2 + I_3 \omega_3 \cdot \omega_3) \quad (4.24) \]

where $I_i$ is the rotational inertia corresponding to $\theta_i$, and $\omega_i$ is defined as above. Since there are no potential energy storage elements other than those caused by gravity, $V$ can be expressed with these heights:

\[ V = m_1 gh_1 + m_2 gh_2 + m_3 gh_3 \]
\[ = m_1 gl_2 (1 - \cos(\theta_1)) - m_3 gl_3 \cos(\theta_1) \sin(\theta_3) \quad (4.25) \]

3. Again, we assume frictionless joints, so $Q = 0$. In this model, we neglect electrical motor dynamics, due to the prevalence of high-performance motor drivers with fast or negligible dynamics.

4. Substitute the values into the Euler-Lagrange equation.

The Euler-Lagrangian model above can be expressed in Acumen as follows:

\[
\text{model RWS\_3DOF\_gimbal} \\
(m1,m2,m3,I1,I2,I3,l1,l2,l3) =
\]
initially
\[ t_1 = 0, t_2 = 0, t_3 = 0, \]
\[ t_1' = 0, t_2' = 0, t_3' = 0, \]
\[ t_1'' = 0, t_2'' = 0, t_3'' = 0, \]
g = 9.81, \( q = \text{zeros}(3), \)
\( T = 0, \ V = 0, \ L = 0, \)
\( T_{01} = \text{zeros}(3,3), \ T_{12} = \text{zeros}(3,3), \)
w1 = \text{zeros}(3), w2 = \text{zeros}(3),
w3 = \text{zeros}(3)
always
\[ m_1 = 2, m_2 = 1, m_3 = 1, I_1 = 1, \]
\[ I_2 = 1, I_3 = 1, l_1 = 3, l_2 = 1.5, \]
\[ q = (t_1, t_2, t_3), \ g = 9.81, l_3 = 0.75, \]
\[ T_{01} = ((-\cos(t_2), -\sin(t_2), 0), \]
\[ (0, 0, 1), \]
\[ (-\sin(t_2), -\cos(t_2), 0)), \]
\[ T_{12} = ((-\cos(t_3), -\sin(t_3), 0), \]
\[ (0, 0, -1), \]
\[ (\sin(t_3), \cos(t_3), 0)), \]
w1 = (0,0,t1'), w2 = w1 + (0,0,t2') * T01,
w3 = w2 + (0,0,t3') * (T01 * T12),
\[ T = 0.5 * (I_1 * \text{dot}(w_1,w_1) + \]
\[ I_2 * \text{dot}(w_2,w_2) + \]
\[ I_3 * \text{dot}(w_3,w_3)), \]
\[ V = m_1 g l_2 (1 - \cos(t_1)) - \]
\[ m3 \cdot g \cdot l3 \cdot \sin(t1) \cdot \sin(t3), \]
\[ L = T - V, \]

foreach i in length(q)
\[ L'[(q(i))']' - L'[q(i)] = 0 \]

Fig. ?? shows the compound motion of the RW-S Gimbal model. This model led us to understand the need for Lagrangian modeling and, as a result, provides us with concrete justification for introducing support for static partial derivatives and undirected equations. Supporting static partial derivatives and Euler-Lagrangian equations allows the modeler to specify the dynamics directly instead of performing symbolic differentiation and equation solving (acausal to casual transformation) manually.

4.4 Case Study III: Electromechanical Systems.

In this section we present a case study of an integrated electromechanical system, the basics of which are integral for fields such as robotics.

4.4.1 Background

Simple electromechanical systems like the solenoid attached to a mechanical spring and damper, shown in Fig. 4.6, exhibit the coupled dynamics of much more complicated systems. They can demonstrate the core language’s ability to perform multiphysics simulations, such as coupled mechanical and electrical subsystems.

This system in Fig. 4.6 is powered by an input voltage \( e(t) \) that drives a solenoid with a core of mass \( m \) and inductance \( L(x) \), which is the inductance of the solenoid given as a function of the core’s position. The system also has an electrical resistance
$R$, coupled to a linear stiffness and damping of $k$ and $\beta$, respectively, constraining its movement to be in only one direction: perpendicular to gravity.

### 4.4.2 Analytical Model

To derive a model for the system, we proceed as follows:

1. Choose generalized coordinates to be the electrical charge of the circuit and the position of the solenoid’s core, that is, $q_1 = q$, $q_2 = x$.

2. The kinetic energy term is the sum of the field energy stored in the inductor and the velocity of the solenoid’s mass, defined as follows:

$$T = \frac{1}{2} L(x) \dot{q}_1^2 + \frac{1}{2} m \dot{q}_2^2$$  \hspace{1cm} (4.26)

Since the solenoid travels in a direction perpendicular to gravity, and neglecting any capacitance of the electrical circuit, the potential energy term is simply the mechanical energy stored in the spring, defined as:

$$V = \frac{1}{2} k q_2^2$$  \hspace{1cm} (4.27)

3. The dissipation energy term of the electrical resistance and mechanical damping can be included in the external forces:

$$Q = -\frac{\partial D}{\partial \dot{q}} + e(t)$$ \hspace{1cm} (4.28)

where $D$ is defined as

$$D = \frac{1}{2} R \dot{q}_1^2 + \frac{1}{2} \beta \dot{q}_2^2$$ \hspace{1cm} (4.29)

4. Write the Euler-Lagrange equation (4.8) in Acumen syntax.
While this example seems straightforward, it does show one important benefit of using the Euler-Lagrange method, namely the development of coupled dynamics expressions. In this case, the coupled terms are the force exerted on the solenoid’s mass by the magnetic field of the inductor, and the induced voltage due to the translation \( x \), which can be seen to be:

\[
F = \frac{1}{2} \frac{\partial L}{\partial q_2} \dot{q}_1^2
\]

(4.30)

and

\[
V = \frac{1}{2} \frac{\partial L}{\partial q_2} q_1 q_2
\]

(4.31)

If we use Newton’s method, these terms would need to be known so that we can create the free body diagrams.

### 4.5 Case Study IV: A Compass Gait Biped

McGeer first introduced the concept of passive dynamic walking (PDW) in 1990 [43], shedding light on achieving stable human-like locomotion without any external forces. A number of biped robots have been developed following this model, including the Cornell biped [44] and Amber [45, 46], demonstrating great success in the domain of
robotic walking. Hence it is worthwhile to investigate how well the core formalism that we have identified in this thesis can model such systems.

The compass gait biped model is a two-dimensional unactuated rigid body system placed on a downward surface, inclined with a fixed angle \( \gamma \) from the horizontal plane. A diagram of the model is shown in Fig. 4.7 with its physical parameters. The configuration of this two-link mechanism can be described by the generalized coordinates \( q = [\theta_1, \theta_2] \), where \( \theta_1 \) is the angle from the vertical line to the stance leg, and \( \theta_2 \) is the angle between the two legs. It is a hybrid model featuring two phases. At the start of each 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.

![Figure 4.7 : A Compass Gait Biped](image)
4.5.1 Continuous Dynamics

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. 4.7, from which it is easy to define the kinetic and potential energy of the system as follows:

\[
T = \frac{1}{2} m_1 (\dot{x}_1^2 + \dot{y}_1^2 + \dot{x}_2^2 + \dot{y}_2^2 + \dot{x}_3^2 + \dot{y}_3^2)
\]

\[
V = m_1 g (y_1 + y_3) + m_2 g y_2
\]

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.

4.5.2 Discrete Events

When the swing foot impacts the surface of the slope, a discrete event is triggered to update the angular velocities, in order to prevent the biped from falling through the floor. The perpendicular distance from the walking surface to the tip of the swing leg is given by

\[
guard = l \sin \gamma (\sin \theta_1 + \sin (\theta_2 - \theta_1))
\]

where \(\gamma\) 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 described as follows:

\[
guard \leq 0 \land guard < 0
\]

Using conservation of angular momentum [1], the explicit solution of post-impact velocities is as follows:

\[
H_1 = \begin{bmatrix}
    m_1 l^2 (\frac{5}{4} - \frac{\cos \theta_2}{2}) + m_2 l^2 & \frac{m_1}{4} l^2 (1 - 2 \cos \theta_2) \\
    \frac{m_1}{2} l^2 \cos \theta_2 & \frac{m_1}{4} l^2
\end{bmatrix}
\]
$$H_2 = \begin{bmatrix} -\frac{m_1}{4} l^2 + (m_2 l^2 + m_1 l^2) \cos \theta_2^- & -\frac{m_1}{4} l^2 \\ -\frac{m_1}{4} l^2 & 0 \end{bmatrix}$$

$$\theta_1^+ = \theta_2^- - \theta_1^- \quad \theta_2^+ = -\theta_2^-$$

$$[\dot{\theta}_1^+, \dot{\theta}_2^+]^T = H_1^{-1} \cdot H_2 \cdot [\dot{\theta}_1^-, \dot{\theta}_2^-]^T$$

Fig. 8.1 in the Supplementary Material section shows the compass gait biped mathematical model and the Acumen program. This example shows that the core language can support a direct mapping from a mathematical model to simulation code for a hybrid system model with complex dynamics.

Thus, this case study suggests that a formalism supporting hybrid ODEs and partial derivatives suffices to model the dynamics of non-trivial robot rigid body dynamics.
Part II

Compile-Time Extensions to Hybrid ODEs
Chapter 5

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 [47], HyTech [48], PHAVer [49], dReach [50], dReal [51], SpaceEx [52], and Flow* [53]. 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.

5.1 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 [54]. Examples illustrating the use of these extension can be found in the directory examples/04_Experimental/04_BTA. Since both partial deriva-
tives 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 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) [55, 56, 57, 58], 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 thesis 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 5.2), we introduce the syntax and type system for a core differential equation language (Section 6). Then, we present a declarative specification of binding-time analysis (BTA) and a big step semantics for specialization (Section 7), 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 8), 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.
5.2 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) [59, 60, 61] 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 thesis 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 [62] and Tempo [63]; template instantiation mechanisms, namely, C++ Templates [64] and Template Haskell [65]; multi-stage programming languages, namely, MetaOCaml [64] and LMS [66]; the Verilog Preprocessor [67]; and the hybridization technique [68].

Table 5.1 provides an overview of how these different systems related to these key
Table 5.1: Comparison of Compile-time Transformation

<table>
<thead>
<tr>
<th></th>
<th>Static checking</th>
<th>Source level reporting</th>
<th>Compile-time</th>
<th>Domain-specific</th>
<th>Let insertion</th>
<th>Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-Mix</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Tempo</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C++ Template</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>Template Haskell</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>MetaOcaml</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LMS</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Verilog preprocessor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>Hybridization</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
</tr>
<tr>
<td>This thesis</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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 two-level languages) need to be limited to general purpose languages. In fact, as this thesis 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 [69], 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.
Syntax

\[ \begin{align*}
\text{n} & \in \text{Names}, \quad i \in \mathbb{N}, \quad q \in \mathbb{Q} \quad \text{and} \quad t \in \mathbb{B} \\
\text{Constant} & \quad k ::= i | q | t \\
\text{Variable} & \quad x ::= n | x' \\
\text{Type} & \quad \tau ::= \text{nat} | \text{bool} | \text{real} | \prod_{j \in 1..n} \tau_j \\
\text{Type Environment} & \quad \Gamma ::= \{x_j : \tau_j\}^{j \in 1..n} \text{ such that whenever } x_i = x_j \text{ then } i = j \\
\text{Function} & \quad f ::= + | - | \times | \div | ^\wedge | \&\& | || | > | >= | == | != | \sin | \cos \\
\text{Expression} & \quad e ::= k | x | (e_j)^{j \in 1..n} | e_1(e_2) | f(e) | \frac{d}{dt}e | \frac{\partial}{\partial e_2}e_1 \\
\text{Equation} & \quad s ::= x = e | e_1 = e_2, \quad e_1 \neq x | x^+ = e | \text{if } e \text{ then } s_1 \text{ else } s_2 | \\
& \quad \forall n \in e. \ s | \{s_j\}^{j \in 1..n}
\end{align*} \]

\[ \begin{align*}
\Gamma \vdash e : \tau \\
\Gamma \vdash i : \text{nat} & \quad \Gamma \vdash q : \text{real} & \Gamma \vdash t : \text{bool} \\
\Gamma \vdash n : \tau & \quad \Gamma \vdash x' : \text{real} \\
\Gamma \vdash (e_j)^{j \in 1..n} : \prod_{j \in 1..n} \tau_j \\
\Gamma \vdash e_i : \prod_{j \in 1..n} \tau_j & \quad \Gamma \vdash i : \text{nat} & \quad i < m \\
\Gamma \vdash e_1(i) : \tau_i & \quad \Gamma \vdash e_1(e_2) : \tau \\
\Gamma \vdash e : \prod_{j \in 1..n} \tau_{f,j} & \quad \Gamma \vdash e_1 : \text{real} & \quad \Gamma \vdash e_2 : \text{real} \\
\Gamma \vdash f(e) : \tau & \quad \Gamma \vdash \frac{d}{dt}e : \text{real} & \quad \Gamma \vdash \frac{\partial}{\partial e_2}e_1 : \text{real} \\
\Gamma \vdash s \\
\Gamma \vdash e_1 : \tau & \quad \Gamma \vdash e_2 : \tau \\
\Gamma \vdash e_1 = e_2 & \quad \Gamma \vdash x : \tau \\
\Gamma \vdash x^+ = e & \quad \Gamma \vdash e : \text{bool} & \quad \Gamma \vdash s_1 & \quad \Gamma \vdash s_2 \\
\Gamma \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \end{align*} \]

\[ \begin{align*}
\Gamma \vdash e : \prod_{j \in 1..n} \tau & \quad \Gamma, n : \tau \vdash s \\
\Gamma \vdash \forall n \in e. \ s & \quad \Gamma \vdash \{s_j\}^{j \in 1..n} \\
\end{align*} \]

Figure 5.1 : Syntax and Type system for Acumen
Chapter 6

A Differential Equation Language

This chapter presents the syntax and type system for a core differential equation language called Acumen. In addition, one example in the field of robotics was used to show the expressiveness of this minimal language as well as highlighting the general steps of Acumen’s offline partial evaluator.

6.1 Syntax and Type System

Fig. 5.1 describes the syntax of Acumen. We use the following notational conventions:

- Writing \( \langle e_j \rangle_{j \in 1 \ldots n} \) denotes a vector \( \langle e_1, e_2, \ldots, e_m \rangle \). We will occasionally omit the superscript \( j \in \{1 \ldots m\} \) and write \( \langle e_j \rangle \) when the range of \( j \) is clear from context.

- Writing \( \{ e_j \}_{j \in 1 \ldots n} \) denotes a set \( \{ e_1, e_2, \ldots, e_m \} \), 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 \( \{ \text{true}, \text{false} \} \), 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 prod-
ucts, respectively. A type environment is a partial function from variables to type terms. We treat environments as graphs of functions or as functions.

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 $\left(\frac{\partial}{\partial e_2} e_1\right)$ 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 $\text{expr}'[\text{expr}]$. 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 \( \{s_j\}_{j=1}^n \).

A Acumen expression \( e \) has type \( \tau \) under environment \( \Gamma \) when the judgement \( \Gamma \vdash e : \tau \) is derivable according to the rules presented in Fig. 5.1. 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 \( j \)th argument to the function. Partial derivatives have straightforward rules. The rules for equations are straightforward. Finally, environment extension of environment \( \Gamma \).
with the binding \( x : \tau \), written \( \Gamma_1, x : \tau \) is an environment \( \Gamma_2 \) defined as follows:

\[
\Gamma_2(y) = \begin{cases} 
\tau & \text{if } y = x, \\
\Gamma_1(y) & \text{otherwise.}
\end{cases}
\]
6.2 Example: A Lagrangian Model

Figure 6.1: Compiling Pendulum/Mass Example

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

\[ q = (x, \theta), a = 1, m = 2, M = 5, g = 9.8, \]
\[ k = 2, I = \frac{4}{3}ma^2, L = T - V, \]
\[ T = \frac{1}{2}(M + m)x^2 + ma\dot{x}\cos(\theta) + \frac{2}{3}ma^2\dot{\theta}^2 \]
\[ V = \frac{1}{2}kx^2 + mg(1 - \cos(\theta)), \]
\[ \forall i \in \{1...|q|\}, \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = 0 \]

(b) After Binding-Time Analysis (BTA)

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

(c) After Specialization (Implicit ODEs)

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

(d) After Symbolic Gaussian Elimination (Explicit ODEs)

\[ A = \sin(\theta), B = \cos(\theta), \]
\[ \ddot{x} = \frac{2(A\dot{\theta}^2 - x) - B\ddot{\theta}}{7} \]
\[ \ddot{\theta} = \frac{-\frac{98}{3}A - 4B(A\dot{\theta}^2 - x)}{\frac{98}{3} - 4B^2} \]
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 6.1(a) provides the Acumen-17 model of a second order nonlinear system shown in Figure 6.2. 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 $\theta$, 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 6.1(a), the $\forall$ quantifier is used to introduce the index variable for a family of equations. In the ASCII-based syntax, the keyword $\text{foreach}$ represent this quantifier. The intent is to express as concisely and as close to what would typically appear in a mechanics textbook:

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

This notation generally has a syntactic interpretation, that is, the name contained in the $i$th element of the vector is looked up. In other words, this family of equations literally represents the following two 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. 6.1(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 $\theta$ 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. 6.1(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 $\theta$ 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. 6.1(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. 6.1(a) are as follows:

The Acumen implementation supports an enclosure simulation semantics that
\[ \begin{align*}
ux &= 100 \cdot (2 - x) + 30 \cdot (0 - \dot{x}), 
\dot{ut} &= 100 \cdot (\pi - \theta) + 40 \cdot (0 - \dot{\theta}), 
\end{align*} \]

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

produces rigorous over-approximations (guaranteed upper and lower bounds) for all simulations [70]. 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:

\[(a) \ \theta \quad (b) \ \dot{\theta} \quad (c) \ x \quad (d) \ \ddot{x}\]
6.3 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. 6.3(a) shows such a cam mechanism, the curved outline of cam $r$ is a function of rotational angle $\theta$, defined as below:

$$r = (1.5 - \cos(\theta) / 2) \times (1 + \cos(2\theta) / 5)$$

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. 8.1 in the Appendix shows the mathematical model and the corresponding Acumen program. Clearly, supporting partial
derivatives in the language greatly simplifies the modeling task, and can save the modeler much tedious and error-prone work.

6.4 A Compass Gait Biped Example

The Compass gait biped model [44, 45] is a two dimensional unactuated rigid body system placed on a downward surface inclined at a fixed angle $\gamma$ from the horizontal plane. A diagram of the model is shown in Fig. 6.3(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 $\theta_1$ is the angle from the vertical line to the stance leg and $\theta_2$ is the angle between two legs. It is a hybrid model featuring two phases. At the start of each 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.

6.4.1 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. 6.3(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

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

Where $\gamma$ 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: $\text{guard} \leq 0 \wedge \dot{\text{guard}} < 0$. Using conservation of angular momentum [1], the explicit solution of post impact velocities can be determined. Fig. 8.1 in the Appendix shows the mathematical model and the full Acumen model. 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.
Chapter 7

BTA and Specialization

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

7.1 Binding Time Analysis

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 [55]. Fig. 7.1 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 re-
\[
\text{Binding Time Environment: } \Gamma^b = \{x_j : \tau_j^b\}_{j \in 1..n} \text{ and } x_i = x_j \implies i = j
\]

**Expression**

\[
e^b ::= k^b \mid x^b \mid (e_j^b)_{j \in 1..n}^b \mid e_j^b e_2^b \mid f(e^b)
\]

\[
e^b \mid (\frac{\partial}{\partial x^b})e^b \mid (\frac{\partial}{\partial x^b})c_1^b
\]

**Equation**

\[
s^b ::= (x^b = e^b)^b \mid (x^b = e^b)^b \mid (e_1^b = e_2^b)^b, e_1 \neq x
\]

\[
| (\text{if } e^b \text{ then } s_1^b \text{ else } s_2^b)^b \mid (\forall n^b \in e^b, s^b)^b
\]

\[
| (\{s_j^b\}_{j \in 1..n}^b)
\]

**Binding Time Type**

\[
\tau^b ::= \text{nat}^b \mid \text{bool}^b \mid \text{real}^b \mid (\prod_{j \in 1..n} \tau_j^b)^b
\]

\[
\Gamma^b \vdash \tau^b
\]

**Figure 7.1**: Binding Time Analysis for Acumen
quires 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.

7.2 Specialization

Fig. 8.1 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 \( \rightarrow_e \) 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 \( \rightarrow_s \) 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
\[ \frac{\partial}{\partial x^2} e_1, \text{ when } e_2 \text{ can not be specialized to a variable } x^b. \text{ It is analogous to the traditional division by zero error.} \]

### 7.3 Type Safety

**Definition 1.** The erasure relation \( \cdot \) for \( e^b, s^b \) and \( \Gamma^b \) 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. \( \square \)

**Lemma 2.** Substitution type preservation \( \forall \Gamma^b, x, e, s, \tau. \)

\[
\Gamma^b \vdash v^{b_1} : \tau^{b_1} \land \Gamma^b, x : \tau^{b_1} \vdash e : \tau^{b_2} \\
\quad \Rightarrow \quad \Gamma^b \vdash e[x := v^{b_1}]^{b_2} : \tau^{b_2}
\]

\[
\Gamma^b \vdash v^{b_1} : \tau^{b_1} \land \Gamma^b, x : \tau^{b_1} \vdash s^{b_2} \\
\quad \Rightarrow \quad \Gamma^b \vdash s[x := v^{b_1}]^{b_2}
\]

**Proof.** By induction on the derivation of \( \Gamma^b \vdash e^b : \tau^b \) and \( \Gamma^b \vdash s^b \) respectively. \( \square \)

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

\[
\Gamma^b \vdash e^b : \tau^b \land e^b \rightarrow_e v^b \Rightarrow \Gamma^b \vdash v^b : \tau
\]

\[
\Gamma^b \vdash s^b \land s^b \rightarrow_s w^b \Rightarrow \Gamma^b \vdash w^b
\]
Lemma 4. Static value $\forall \Gamma^b, e$.

\[
\Gamma^b \vdash v^S : \text{nat}^S \implies |v^S| = i \\
\Gamma^b \vdash v^S : \text{bool}^S \implies |v^S| = t \\
\Gamma^b \vdash v^S : \text{real}^S \implies |v^S| = q
\]

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. 7.1, we can prove the lemma above.

Theorem 1. Type safety of specialization Let $\Gamma^D$ denote $\{x_j : \tau^D\}_{j\in1...n}$ and $\forall \Gamma^D, e, b, v^b, w^b$

\[
\Gamma^D \vdash e : \tau^b \land e^b \hookrightarrow e r \implies r \neq \text{err} \land \Gamma^D \vdash r : \tau^b \\
\Gamma^D \vdash s^b \land s^b \hookrightarrow s r \implies r \neq \text{err} \land \Gamma^D \vdash r
\]

Due to the page limits, the proof can be found in Section 1 of the supplementary document.
Chapter 8

Algorithmic Specification

This chapter 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.

8.1 Generating Binding Time Constraints

Fig. 8.1 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 $S \sqsubseteq D, S \sqsubseteq S$ and $D \sqsubseteq D$.

8.2 Control Flow and Control Scope

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

$$x = 1, \text{ if } t < 5 \text{ then } y = x \text{ else } y' = x$$

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 $\pi : \text{Variable} \to \text{Label} \cup \text{D}$ is called local environment. □

Definition 3. A map $\rho : l \to \pi$ 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 $\pi$, 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 $\rho$:

$$
\Delta(l_0, \{s_j\}_{j\in1...n}, \rho) = \Delta(l_0, \{s_j\}_{j\in2...m}, \Delta(l_0, s_1, \rho))
$$

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

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

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:

$$
\{\text{root} \to \{x \to \text{root}_{11}\}, \text{root}_{22} \to \{x \to \text{root}_{11}, y \to \text{root}_{211}, \}, \\
\text{root}_{23} \to \{x \to \text{root}_{11}, y' \to \text{root}_{2311}\}\}
$$

Definition 4. $C = \{c_j\}_{j\in1...n}$ is called a constraint set. The labels of $C$ are defined as

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

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

$$
c \equiv S \subseteq l, \ c \equiv l \subseteq D, \ c \equiv l \subseteq \tilde{l}
$$
and is of *error form* if $D \subseteq S \in C$. The sets of $C$ in normal form and of error are denoted by $NF$ and $Error$, respectively.

Fig. 8.1 defines function $[\cdot]$ that takes $e$ in scope $l_0$ and *global environment* $\rho$ returns a constraint set. It generates a constraint $l \subseteq S$ 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 $\frac{\partial}{\partial e_1}e_1$, we generate constraints between label of $e$ and all its subexpressions $e_i$, that is $\bigcup_i \{ l_i \subseteq l \}$. 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 $l_1 \subseteq \rho(l_0,n)$ and $l_1 \subseteq \rho(l_0, x')$ respectively. For $x^+ = e$, $e_1 = e_2$ and $\{s_j\}$, the binding time depends on subexpressions and sub-constraints. Notice when generating constraints for a equation $e_1 = e_2$, we make sure it is not an assignment, that is $e_1 \neq x$. 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* $\rho$ when inductively generate constraint from $s$. 
### Syntax

\[ l := \text{root} \mid l_i \text{ where } i \in \mathbb{N} \]

### Binding Time Expression

\[ B := b \mid l \]

### Constraint

\[ c := B \sqsubseteq B \]

### Satisfied Constraint

\[ \vdash c := \vdash S \sqsubseteq D \mid \vdash S \sqsubseteq S \mid \vdash D \sqsubseteq D \]

### Constraint Generation Function

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>( { l \subseteq S } )</td>
<td>( x = e )</td>
<td>( [x]<em>{l_0,l_1,\rho} \cup [e]</em>{l_0,l_2,\rho} )</td>
</tr>
<tr>
<td>( n )</td>
<td>( { \rho(l_0,n) \subseteq l, \rho(l_0,n) } )</td>
<td>( x^+ = e )</td>
<td>( [e]_{l_0,l_2,\rho} \cup { l_2 \subseteq l } )</td>
</tr>
<tr>
<td>( l \subseteq \rho(l_0,n) } )</td>
<td>( e_1 = e_2 )</td>
<td>( e_1 )</td>
<td>( [e_1]<em>{l_0,l_1,\rho} \cup [e_2]</em>{l_0,l_2,\rho} )</td>
</tr>
<tr>
<td>( x' )</td>
<td>( [x]_{l_0,l_1,\rho} \cup { \rho(l_0,x') \subseteq l, D \sqsubseteq l_1 } )</td>
<td>( j \in 1...n { t_j \subseteq l } )</td>
<td>( [s_1]_{l_2,l_2,\rho} )</td>
</tr>
<tr>
<td>( \langle e_j \rangle )</td>
<td>( \bigcup_{j \in 1...n} [e_j]_{l_0,l_j,\rho} \cup { e } ) ( \langle e \rangle )</td>
<td>( j \in 1...n { t_j \subseteq l } )</td>
<td>( [s_2]_{l_3,l_3,\rho} )</td>
</tr>
<tr>
<td>( e_1(e_2) )</td>
<td>( [e_1]<em>{l_0,l_1,\rho} \cup [e_2]</em>{l_0,l_2,\rho} ) ( \cup { l_1 \subseteq l, l_2 \subseteq l } ) ( \wedge n \in e . s ) ( [n]<em>{l_0,l_1,\rho} \cup [s]</em>{l_0,l_3,\rho(l_0,n)\rightarrow l_1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( f(e) )</td>
<td>( [e]_{l_0,l_1,\rho} \cup { l_1 \subseteq l } ) ( \cup { l_2 \subseteq l } ) ( e )</td>
<td>( [e]_{l_0,l_2,\rho} \cup { l_2 \subseteq l_1 } )</td>
<td></td>
</tr>
<tr>
<td>( \frac{d}{dx} e )</td>
<td>( [e]_{l_0,l_1,\rho} \cup { l_1 \subseteq l } ) ( \cup { l_3 \subseteq l } ) ( e )</td>
<td>( { l_3 \subseteq l } )</td>
<td></td>
</tr>
<tr>
<td>( \frac{\partial}{\partial e_1} e )</td>
<td>( [e_1]<em>{l_0,l_1,\rho} \cup [e_2]</em>{l_0,l_2,\rho} ) ( { s } ) ( s_j )</td>
<td>( \cup { l_j \subseteq l } )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.1: Constraint Generation
**Definition 5.** A map \( \sigma : \text{Label} \rightarrow \text{Binding time} \) is called a *substitution*, and \( \sigma \) is the identity function on labels not in domain of \( \sigma \). As Label is finite, \( \text{dom}_\sigma = \{l_1, \ldots, l_n\} \). Thus, \( \sigma \) may be described by \([l_1 \mapsto \sigma(l_1), \ldots, l_n \mapsto \sigma(l_n)]\).

Given two substitutions \( \sigma \) and \( \tilde{\sigma} \), the *extension* of \( \sigma \) with \( \tilde{\sigma} \) is defined as \( \sigma \triangleright \tilde{\sigma} : \text{dom}_\sigma \cup \text{dom}_{\tilde{\sigma}} \rightarrow \text{Binding time} \) such that

\[
\sigma \triangleright \tilde{\sigma}(l) = \begin{cases} 
\sigma(l) & \text{if } l \in \text{dom}_\sigma, \\
\tilde{\sigma}(l) & \text{otherwise.}
\end{cases}
\]

**Definition 6.** A substitution \( \sigma \) 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 \( \sigma_c \) is a *minimum solution* to \( C \), denoted by \( \sigma_c \vdash_{\text{min}} C \), when

- \( \sigma_c \vdash C \),
- \( \text{dom}_{\sigma_c} = \text{Labels}(C) \),
- \( \sigma \vdash C, l \in \text{Labels}(C) \implies \vdash \sigma_c(l) \sqsubseteq \sigma(l) \).

\( \square \)

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

\[
\forall C, \sigma_1, \sigma_2. \quad \sigma_1, \sigma_2 \vdash_{\text{min}} C \implies \sigma_1 = \sigma_2.
\]

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

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

\[
C \in \text{NF} \implies \exists \sigma_c. \sigma_c \vdash_{\text{min}} C.
\]
Proof. Consider \( C \in NF \). Define a substitution \( \sigma_c \) as \([l_1 \mapsto S, \ldots, l_n \mapsto S] \), where \( \text{Labels}(C) = \{l_1, \ldots, l_n\} \). As \( \sigma_c \) clearly solves all constraints of the form \( l \subseteq \tilde{l}, S \subseteq l \) or \( D \subseteq l \) with \( l, \tilde{l} \in NF \), it is a solution \( \sigma_c \vdash C \).

The minimality follows from \( \text{dom}_{\sigma_c} = \text{Labels}(C) \) and from the fact that both \( \vdash S \subseteq S \) and \( \vdash S \subseteq D \).

\[\Box\]

8.3 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. 8.2, a set of normalizing rewrite rules are shown.

Definition 7. For a \( C \in C \) the application of a normalization rewrite rule from Fig. 8.2 returns a constraint set \( \tilde{C} \) and a substitution \( \sigma \). We denote this by \( (\rightarrow) \)

\[ C \rightarrow^\sigma \tilde{C} \]. Exhaustive application is denoted by \( C \rightarrow^{* \sigma} \tilde{C}_k \).

\[ \Box \]

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

Proof. By inspecting Fig. 8.2, it is obvious that every rule reduces the number of constraints by one.

\[ \Box \]
Lemma 8 (Solution preservation).

\[ C \rightarrow^\sigma \tilde{C} \land \tilde{\sigma} \vdash \tilde{C} \implies \sigma \triangleright \tilde{\sigma} \vdash C. \]

Proof. By case analysis of the normalization rewrite rules. a), b), c): Since \( \sigma \) is the empty substitution, it is clear that \( \sigma \triangleright \tilde{\sigma} = \tilde{\sigma} \). As \( \text{Labels}(C) = \text{Labels}(\tilde{C}) \), it follows that \( \sigma \triangleright \tilde{\sigma} \vdash C \).

d), e): The constraint removed from \( C \) is solved by \( \sigma \). Thus, \( \tilde{\sigma} \vdash \sigma(C) \) follows from \( \tilde{\sigma} \vdash \tilde{C} \). As \( \text{Labels}(\sigma(C)) = \text{Labels}(\tilde{C}) \) and \( \text{dom}_\sigma \cap \text{Labels}(\tilde{C}) = \emptyset \), we obtain \( \sigma \triangleright \tilde{\sigma} \vdash C \). \( \square \)

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

\[ \tilde{C} \in \text{NF} \lor \tilde{C} \in \text{Error}. \]

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 \( S \subseteq l, l \subseteq D, l \subseteq \tilde{I} \) or \( D \subseteq S \).

If \( D \subseteq S \) is present, then \( \tilde{C} \in \text{Error} \), else \( \tilde{C} \in \text{NF} \). \( \square \)

Lemma 10 (Minimal solution preservation).

\[ C \rightarrow^\sigma \tilde{C} \land \sigma_{\tilde{\varepsilon}} \vdash_{\text{min}} \tilde{C} \implies \sigma \triangleright \sigma_{\tilde{\varepsilon}} \vdash_{\text{min}} C. \]

Proof. Lemma 8 implies that \( \sigma_{\varepsilon} = \sigma \triangleright \sigma_{\tilde{\varepsilon}} \vdash C \). To show minimality first note that \( \text{dom}_{\sigma_{\varepsilon}} = \text{dom}_\sigma \cup \text{Labels}(\tilde{C}) = \text{Labels}(C) \) with the union being disjoint. Now assume that \( l \in \text{Labels}(C) \), \( \tilde{\sigma} \vdash C \) and consider the following two cases.

\( l \in \text{Labels}(\tilde{C}) \): Clearly \( \tilde{\sigma} \vdash \tilde{C} \). As \( \sigma_{\varepsilon} \) is minimal, \( \models \sigma_{\varepsilon}(l) \subseteq \tilde{\sigma}(l) \). Thus, from \( \sigma_{\varepsilon}(l) = \sigma_{\varepsilon}(l) \) we get that \( \models \sigma_{\varepsilon}(l) \subseteq \tilde{\sigma}(l) \).

\( l \in \text{dom}_\sigma \): \( \sigma \) was obtained by applying rule e) or d). Thus, either \( \sigma_{\varepsilon}(l) = \sigma(l) = S \) or \( \sigma_{\varepsilon}(l) = \sigma(l) = D = \tilde{\sigma}(l) \). In both cases \( \models \sigma_{\varepsilon}(l) \subseteq \tilde{\sigma}(l) \) holds. \( \square \)
Theorem 2 (Unique minimal solution).

\[ C \to^* \sigma \tilde{C} \land \tilde{C} \in NF \implies \exists! \sigma_c. \sigma_c \vdash_{\text{min}} C. \]

Proof. We obtain \( \sigma_\varepsilon \) such that \( \sigma_\varepsilon \vdash_{\text{min}} \tilde{C} \) from Lemma 6. By definition we have that \( \exists k \in \mathbb{N} \) and \( \sigma_1, \ldots, \sigma_k \) such that \( C \to^{\sigma_1} \tilde{C}_1 \to^{\sigma_2} \ldots \to^{\sigma_k} \tilde{C}_k = \tilde{C} \) and \( \sigma = \sigma_1 \triangleright \ldots \triangleright \sigma_k \).

Thus, by introducing \( \tilde{C}_0 = C \), we get \( \sigma_i \triangleright \ldots \triangleright \sigma_k \triangleright \sigma_\varepsilon \vdash_{\text{min}} \tilde{C}_{i-1} \) for all \( 1 \leq i \leq k \) from Lemma 10. Thus, \( \sigma_c \vdash_{\text{min}} C \) with \( \sigma_c = \sigma \triangleright \sigma_\varepsilon \).

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.

8.4 Binding Time Analysis Correctness

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

- \( \Gamma^b \vdash e : \tau^b \implies \exists! \sigma. \sigma \vdash_{\text{min}} \llbracket e \rrbracket_{l_0, l, \rho} \land \sigma(l) \subseteq b \)
- \( \Gamma^b \vdash s^b \implies \exists! \sigma. \sigma \vdash_{\text{min}} \llbracket s \rrbracket_{l_0, l, \rho} \land \sigma(l) \subseteq b \)

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

- \( \forall e. \Gamma \vdash e : \tau \land \sigma \vdash_{\text{min}} \llbracket e \rrbracket_{l_0, l, \rho} \implies \Gamma^b \vdash e : \tau^{\sigma(l)} \)
- \( \forall s. \Gamma \vdash s \land \sigma \vdash_{\text{min}} \llbracket s \rrbracket_{l_0, l, \rho} \implies \Gamma^b \vdash s^{\sigma(l)} \)

The proofs for the two lemmas above are in Section 2 of the supplementary document.
### Cam and Follower

\[ x = (1.5 - \cos\left(\frac{x}{2} - \theta\right)) \cdot (1 + \cos^2\left(\frac{x}{5} - \theta\right)) \]
\[ \ddot{\theta} = 1 \quad v = \frac{\partial}{\partial x} x \dot{\theta} \quad a = \dot{v} \]
\[ t'' = 1, \quad v = x'[t] \cdot t', \quad a = (v)' \]

### Compass Gait Biped

\[ q = [\theta_1, \theta_2] \quad m_1 = 1 \quad m_2 = 2 \quad l = 1 \]
\[ r = 0.044, \quad g = 9.8, \quad L = T - V \]
\[ x_1 = \frac{1}{2} l \sin \theta_1 \quad y_1 = \frac{1}{2} l \cos \theta_1 \]
\[ x_2 = l \sin \theta_2 \quad y_2 = l \cos \theta_2 \]
\[ x_3 = x_2 + \frac{l}{2} \sin(\theta_2 - \theta_1) \]
\[ y_3 = y_2 - \frac{l}{2} \cos(\theta_2 - \theta_1) \]
\[ \text{guard} = l \sin \gamma (\sin \theta_1 + \sin(\theta_2 - \theta_1)) \]
\[ T = \frac{1}{2} m_1 (x_1'^2 + y_1'^2) \]
\[ + \frac{1}{2} m_2 (x_2'^2 + y_2'^2) \]
\[ + \frac{1}{2} m_2 (x_3'^2 + y_3'^2) \]
\[ V = m_1 g (y_1 + y_3) + m_2 g y_2 \]
\[ H = H_1^{-1} \cdot H_2 \cdot [\dot{\theta}_1, \dot{\theta}_2]^T \]
\[ \text{guard} = l \sin(r) \cdot (\sin(t_1) + \sin(t_2 - t_1)), \]
\[ T = 1/2*ml*((x1)'^2+(y1)'^2+(x3)'^2+(y3)'^2) \]
\[ + 1/2*m2*((x2)'^2+(y2)'^2) \]
\[ V = ml*g*(y1+y3)+m2*g*y2, \]
\[ H = \text{inv}(H1) \cdot H2 \cdot \text{trans}((t1',t2')) \]
∀i ∈ {1...|q|}. \frac{d}{dt} \left( \frac{\partial L}{\partial q_i} \right) - \frac{\partial L}{\partial \dot{q}_i} = 0

h_{11} = m_1 l^2 \left( \frac{5}{4} - \cos \theta_2 \right) + m_2 l^2

h_{12} = \frac{m_1}{4} l^2 (1 - 2 \cos \theta_2)

h_{21} = \frac{m_1}{2} l^2 \cos \theta_2

h_{22} = \frac{m_1}{4} l^2

H_1 = [h_{11}, h_{12}, h_{21}, h_{22}]

h_{210} = -\frac{m_1}{4} l^2 + (m_2 l^2 + m_1 l^2) \cos \theta_2

H_2 = [h_{210} - \frac{m_1}{4} l^2, \frac{m_1}{4} l^2, 0]

if guard < 0 then
\theta_1^+ = \theta_2^- \quad \theta_2^+ = -\theta_2^-
\dot{\theta}_1^+ = H(0) \quad \dot{\theta}_2^+ = H(1)

if guard < 0 && (guard)' then
\text{t1} += \text{t2} - \text{t1}, \text{t2} += -\text{t2}, \text{t1}' += H(0), \text{t2}' += H(1)

noelse

Figure 8.1: Two Examples Problems in Mathematical Notation and in Acumen Syntax
Value \( v^b \) \( ::= \) \( k^S \mid e^D \mid (v_j^b)^{j \in 1\ldots n} \)

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

Function application \( f((v_j^S)^{j \in 1\ldots n}) = v^S \) such that \( v^S = f((v_j)^{j \in 1\ldots n})^S \)

Time derivative \( D_f((v_j^D)^{j \in 1\ldots n}, (v_j^D)^{j \in 1\ldots n}) = v^D \)
such that \( v^D = (\frac{d}{dt} f((v_j^D)^{j \in 1\ldots n}, (v_j^D)^{j \in 1\ldots n}))^D \)

Partial derivative \( P_f((v_j^D)^{j \in 1\ldots n}, (v_j^D)^{j \in 1\ldots n}, x^D) = v^D \)
such that \( v^D = \frac{\partial}{\partial x} f((v_j^D)^{j \in 1\ldots n}, (v_j^D)^{j \in 1\ldots n})^D \)

Free Variables

\[
FV(x^b) = \{x\} \quad FV((e_j^b)^{j \in 1\ldots n}) = \bigcup_j FV(e_j^b) \quad FV((f((e_j^b)^{j \in 1\ldots n}))^b)) = \bigcup_j FV(e_j^b)
\]

\[
LV(x = e) = \{x\} \quad FV(e_1^b(e_2^b)^b) = FV(e_1^b) \cup FV(e_2^b) \quad FV(\frac{d}{dt} (v^b)) = FV(v^b)
\]

\[
FV(\frac{\partial}{\partial e_2} e_1^b) = FV(e_1^b) \cup FV(e_2^b) \quad LV(\{s_j\}^{j \in 1\ldots n}) = \bigcup_j LV(s_j)
\]

\[
e^b =_{c} v^b \cup \{err\}
\]

\[
\frac{e^b}{k^b} = \frac{e_j^b}{v_j^b} \quad \frac{e^b}{(e_j^b)^{j \in 1\ldots n}^b} = \frac{e_j^b}{(v_j^b)^{j \in 1\ldots n}^b} \quad \frac{e^b}{e_1^b((e_j^b)^{j \in 1\ldots n}^b)^{e_2^b}} = \frac{e_j^b}{e_1^b((v_j^b)^{j \in 1\ldots n}^b)^{e_2^b}} \quad \frac{e^b}{e_1^b((e_j^b)^{j \in 1\ldots n}^b)^{e_2^b}} \quad \frac{e^b}{e_1^b((v_j^b)^{j \in 1\ldots n}^b)^{e_2^b}}
\]

\[
\frac{e^S}{f(e^S)^S} = \frac{e_1^b}{v_1^b} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}} \quad \frac{e^b}{e_1^b(v_1^b)^{e_2^b}}
\]

\[
\frac{e^D}{e_1^D} = \frac{e^D}{e_1^D} \quad \frac{e^S}{e_1^D} = \frac{e^S}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D}
\]

\[
i^D = \begin{cases} 1_D & \text{if } x_1 = x_2 \\ 0_D & \text{otherwise} \end{cases} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D}
\]

\[
\frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D} \quad \frac{e^D}{e_1^D}
\]
\[
    s^b \rightarrow_s w^b \cup \{\text{err}\}
\]

\[
    \frac{e^b \rightarrow_e v^b \quad s^{b_s}[x^b := v^b] \rightarrow_s w^{b_s}}{\{x^b = e^b\}^b \cup s^{b_s} \rightarrow_s \{x^b = v^b\} \cup w^{b_s}}
\]

\[
    \frac{FV(e^b) \cap LV(s^{b_s}) = \emptyset}{\{x^{+b_1} = e^b\}^b \cup s^{b_s} \rightarrow_s \{x^{+b_1} = v^b\} \cup w^{b_s}}
\]

\[
    \frac{FV(e^b) \cap LV(s^{b_s}) = \emptyset}{e^b \rightarrow_e v^b \quad s^{b_s} \rightarrow_s w^{b_s}}
\]

\[
    \frac{FV(e^b_1) \cup FV(e^b_2) \cap LV(s^{b_s}) = \emptyset}{e^b_1 \rightarrow_e v^b_1 \quad e^b_2 \rightarrow_e v^b_2 \quad s^{b_s} \rightarrow_s w^{b_s}}
\]

\[
    \frac{FV(e^S) \cap LV(s^{b_s}) = \emptyset \quad t^S_1 = \text{true}^S \quad t^S_2 = \text{false}^S}{\{\text{if } e^S \text{ then } s^{b_1} \text{ else } s^{b_2}\}^b \cup s^{b_s} \rightarrow_s w^{b_j} \cup w^{b_s}}
\]

\[
    \frac{FV(e^D) \cap LV(s^{b_s}) = \emptyset \quad e^D \rightarrow_e v^D}{s^{b_1} \rightarrow_s w^{b_1}_1 \quad s^{b_2} \rightarrow_s w^{b_2}_1 \quad s^{b_s} \rightarrow_s w^{b_s}}
\]

\[
    \frac{FV(e^b) \cap LV(s^{b_s}) = \emptyset \quad e^b \rightarrow_e (\langle v^b_j \rangle_{j \in 1..n})^b}{\{s^{b_1}[n := v^b_j]\}_{j \in 1..m}^{b_1} \rightarrow_s \{w^{b_1}_j\}_{j \in 1..m}^{b_1} \quad s^{b_s} \rightarrow_s w^{b_s}}
\]

\[
    \left\{\forall n \in e^b s^{b_1}_1 \rightarrow_s s^{b_s} \rightarrow_s (\{w^{b_1}_j\}_{j \in 1..n}^{b_1} \cup b^{b_s})\right\}
\]

Figure 8.1 : Specialization Big-Step Semantics
Chapter 9

Conclusion and Future work

This thesis illustrates what language features are needed to effectively model and simulate hybrid systems and builds the case for why modeling languages can be improved by integrated several features, most notably, partial derivatives and support for equations. It presents a novel approach to providing a semantics for supporting automatically eliminate partial derivatives in compile time in the framework of offline partial evaluation.

9.1 Summery of Results

In this thesis we have identified a number of aspects of cyber-physical systems that are arguably pervasive, and have identified the requirements that these aspects place on a modeling language. These requirements were checked against a small collection of language constructs that were found to be a reasonable match, both when considering minimal examples and larger examples. Many of these constructs are supported in existing modeling languages, although usually not in exactly the same form, and not with the same degree of minimalism. For example, while the Acumen language is minimal, it can express many examples more concisely than mainstream tools such as MATLAB and Modelica. Furthermore, there are important constructs in such modeling languages, such as partial derivatives, that do not currently enjoy sufficient support in most languages considered in the thesis. One example of particular impor-
tance from the point of view of classical mechanics is partial derivatives, which are essential for Lagrangian modeling. The only language that appears to have the necessary support is 20-sim. However, it is unclear whether the symbolic differentiation produces a compact result.

The study illustrates clearly how a small language suffices to address the needs of several aspects of cyber-physical systems, including visualization, geometry, particle dynamics and impact, control, quantization and discretization, and composition. It also illustrates that it suffices for expressing rather concise models of several interesting subsystems that arise in real robot design projects. The analysis of these aspects is thorough in that it used an implementation that supports the language constructs described, and all the examples discussed were expressed fully in this language. The models were also tested to check their soundness.

This thesis also presents a novel approach to providing a semantic for supporting automatically eliminate partial derivatives in compile time. The semantic is defined by the use of two-level languages in the framework of offline partial evaluation. Two-level languages incorporate binding time information inside types, that is, whether a piece of code is completely known at compile-time, or can be evaluated only at run-time. It facilitate the specification and justification of the key components of offline partial evaluation: binding-time analysis and specialization. We give an operational semantics for the specialization process along with a declarative and algorithmic specifications of the binding-time analysis. A type safety theorem is given to show the correctness of the semantics for specialization. The declarative specification of binding-time analysis is used to prove soundness with respect to the specialization process. The method is generic and so can be applied to many hybrid systems modeling languages or reachability analysis tools, and has been implemented in the context
of the Acumen modeling language.

9.2 Future Work

The collection of aspects identified and studied in this work is not intended to be exhaustive by any means. Most notably, it is not based on an exhaustive analysis of the needs of modeling all types of physical phenomena, nor is it based on case studies in modeling CPSs. Rather, it is intended to illustrate a minimal set of requirements from this domain on a modeling and simulation language, and is a starting point for further studies on the linguistic needs of multi-physics and large-scale modeling. In future work, we would like to further develop the set of basic aspects in several directions. For example, we would like to explore the possibility of supporting different coordinate systems (especially relative coordinate frames) to facilitate modeling composite mechanical systems. A key question in this respect is whether special language support may be needed, or whether developing appropriate modeling patterns will suffice. Similarly, we would like to study the needs of more sophisticated types of control, including parameter tuning, optimal control, and model-based control. We expect that providing support for minimization of values and functions will be key questions in this respect. At this time it is not obvious whether such support requires special language extensions or whether finding suitable modeling patterns and libraries would suffice. Similar questions apply for path and trajectory planning. The current prototype of the modeling language focuses on creating a ‘minimum viable prototype’ to enable research such as that reported on here. It is an interpreter-based prototype and not tuned for performance. It is sufficient for reasonably accurate near real-time simulation of the models presented in this thesis. However, for larger models, it is important to improve the efficiency of the implementation.
We would like to also investigate of more advanced type systems\cite{71, 72} 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. Finally, we would like to explore the space of physically-inspired modularity, for example, to support physical connectivity network structure and/or abstraction constructs that can facilitate the development of larger (composite) models in a way that mimic physical connectivity.
Bibliography


