Formalization of Bond Graph using HOL Light

By

Ujala Qasim

NUST-2016-173058-MS-CSE-09

Supervisor:

Dr. Osman Hasan

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computational Science and Engineering

Department of Computational Engineering

Research Centre for Modelling and Simulation (RCMS),

National University of Sciences and Technology (NUST), Islamabad,

Pakistan.

(September 2020)

Annex A to NUST Letter No. 0972/102/Exams/Thesis-Cert dated 23 Dec 16.

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr/Ms ____________________ Registration No. _________________________ of **__RCMS___** has been vetted by undersigned, found complete in all aspects as per NUST Statutes/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature with stamp: ____________________________

Name of Supervisor: _____________________________

Date: ____

Signature of HoD with stamp: ______________________

Date: ____

Countersign by

Signature (Dean/Principal): ________________

Date: ___________________________________

Approval

It is certified that the contents and form of the thesis entitled "**Formalization of Bond Graph using HOL Light**" submitted by **Ujala Qasim** have been found satisfactory for the requirement of the degree. This thesis is submitted to the **Research Centre for Modelling and Simulation (RCMS)** in partial fulfillment of the requirements for the degree of Masters of Science in the field of **Computational Science and Engineering**, Department of **Computational Engineering**, University of **National University of Sciences and Technology (NUST), Islamabad, Pakistan.**

Advisor: **Dr. Osman Hasan**

Signature: ________________

Date: ____________________

Committee Member 1: **Dr. Salma Sherbaz**

Signature: __________________

Date: __________________

Committee Member 2: **Dr. Junaid Ahmad Khan**

Signature: __________________

Date: __________________

Dedicated to my Parents and Brother

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at National University of Sciences & Technology (NUST) Research Centre for Modelling & Simulation (RCMS) or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST RCMS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Ujala Qasim**

Signature: ________________

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Osman Hasan for the continuous support, patience, immense knowledge and encouragement through out my MS research. He gave me the valuable chance to join his research group and learn so much from SAVe family. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor.

I am indebted to Dr. Adnan Rashid, who not only inspired and encouraged me but also followed the progress of my work and provided valuable feedback at various stages. His continued support and guidance has been critical to my research.

Besides that, I would like to thank members of my thesis committee, Dr. Salma Sherbaz and Dr. Junaid Ahmad Khan. Their encouragement and insightful comments in study and research helped in the completion of my thesis.

Last but not the least, I would like to thank my family for loving and supporting me throughout my life. Without them, I could have never made it this far.

Table of Contents

List of Tables

List of Figures

Abstract

Bond graph is a unified graphical approach for describing the dynamics of complex engineering and physical systems and is widely adopted in a variety of energy domains, such as, electrical, mechanical, medical, thermal and fluid mechanics. Traditionally, these dynamics are analyzed using paper-and-pencil based proof methods and computer-based techniques. However, both of these techniques suffer from their inherent limitations, such as, human-error proneness, approximations of results and limited computer resources. Thus, these techniques cannot be trusted for performing the bond graph based dynamical analysis of systems from the safety-critical domains like robotics and medicine. Formal methods in particular, theorem proving, can overcome the shortcomings of the traditional methods and provide an accurate analysis of the systems. It has been used for analyzing the dynamics of engineering and physical systems. In this thesis, we propose to use higher-order-logic theorem proving for performing the bond graph based analysis of the physical systems. In particular, we provide a formalization of bond graph, which mainly include conversion of a bond graph model to its corresponding mathematical model (state-space model) and verification of its various properties, such as, stability. To illustrate the practical effectiveness of our proposed approach, we present the formal stability analysis of a prosthetic mechatronic hand using HOL Light theorem prover. Moreover, to facilitate a non-HOL user, we encode our formally verified stability theorems in MATLAB to perform the stability analysis of the prosthetic mechatronic hand.

Chapter 1

Introduction

1.1 Background

Bond Graph (BG) is a linear, labelled, directed and domain-independent graphical approach for modelling dynamics of physical systems and is widely adopted for capturing the dynamics of physical systems belonging to multidisciplinary energy domains, such as, electromechanical, hydroelectric and mechatronics. The concept of bond graphs was introduced by H.M. Paynter [20] of Massachusetts Institute of Technology (MIT) in 1961, to describe the dynamics of different systems belonging to interdisciplinary domains and exhibiting analogous dynamical behaviour. The first step in BG based dynamical analysis of a system is to apply causal equations on different components of a system. Next step involves solving the set of equations simultaneously to obtain the corresponding set of differential equations. These equations are then transformed into state-space representations.

1.2 Motivation

For capturing the dynamics of the physical systems, there are some other graphical approaches namely, Signal-flow graphs and Block diagram representations. However, BG models are considered as a better approach due to their ability to communicate seamlessly between different components belonging to multidisciplinary domains based on bi-directional flow of information and this provide a deep insight to the computational structure of the physical systems. Due to their distinguishing features provided above, bond graphs have been widely used in automobiles [28], biological systems [8], aerospace [10, 11] and transportation systems [16].

Traditionally, the BG based analysis is performed using paper-and-pencil proof method. However, the analysis is prone to error due to high human involvement for analyzing complex physical systems and thus could not ensure absolute accuracy of the analysis. Similalry, computer based symbolic and numerical techniques have been widely used analyzing BG based models of the systems. Some of the widely used tools are ENPORT [13, 23], DESIS [7], MTT [3], CAMP-G [12] and 20-sim [1]. The numerical methods involves the approximation of the mathematical expressions and results due to the finite precision of computer arithmetic. It also involves a finite number of iterations based on limited computational resources and computer memory. Similarly, the symbolic techniques are based on a large numbers of unverified symbolic algorithms present in the core of the associated tools. Based on the above-mentioned limitations, the computer-based methods cannot be trusted for performing the bond graph based analysis of the safety-critical systems, such as, aerospace, medicine and transportation where an inaccurate analysis can lead to disastrous consequences.

Formal methods are computer-based mathematical analysis techniques that involve constructing a mathematical model based on an appropriate logic and verification of its various properties based on deductive reasoning. Higher-order-logic theorem proving is a widely adopted formal methods for performing the accurate analysis of the engineering and physical systems. In this thesis, we propose a higher-order-logic based framework, as represented in Figure. 1, for bond graph based analysis of physical systems. It mainly involves the formalization of bond graph and verification of its various properties using multivariate calculus theories of HOL Light theorem prover [14]. To illustrate the practical effectiveness of our proposed framework, we formally analyze the dynamics of mechatronic prosthetic hand [24] based on the formalization of bond graph. To the best of our knowledge, bond graph theory has not been formalized in any of higher-order-logic theorem provers.

1.3 Proposed Methodology

The main objective of this thesis is the development of a theorem proving based framework for the formal bond graph representation based analysis of the physical and multidisciplinary engineering systems. Our proposed framework, depicted in Figure 1.1 accepts the linear BG representation of a system and its corresponding parameters, such as, values of its different components from a user.

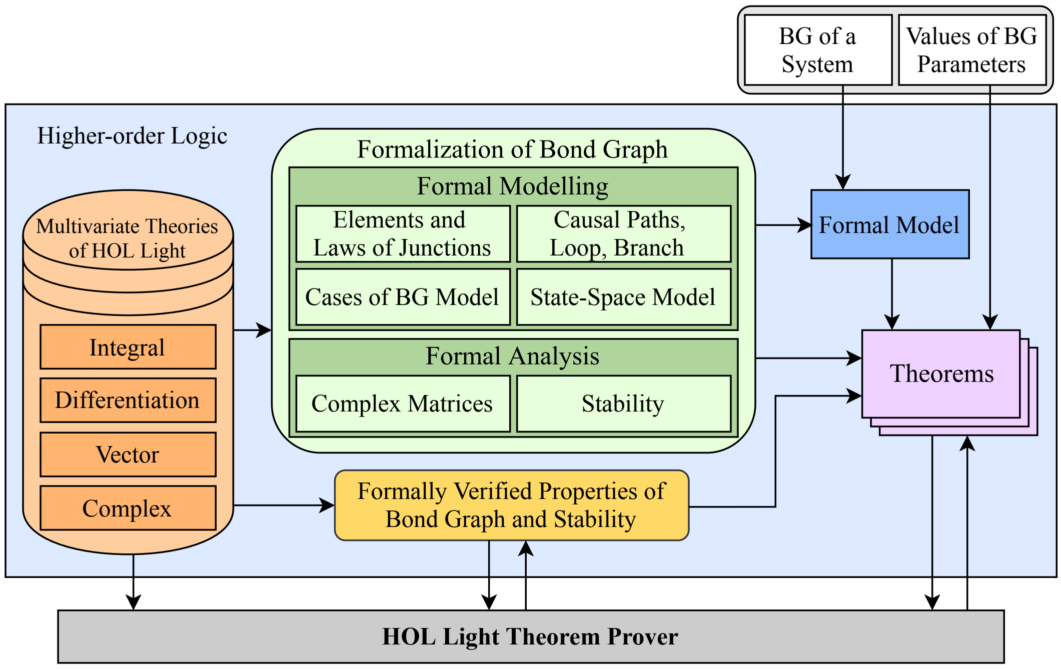


Figure 1.1: Proposed Framework

The first step is the development of the mathematical model (state-space representation), which involves the conversion of BG representation to a set of differential equations. It is based on application of the laws of junctions and causality of paths on the BG representation of the underlying system. The development of the state-space model from a set of differential equations is based on complex-valued vectors and matrices, which are developed as a part of the proposed formalization. The next step is to use the state-space models of the BG representation to formally analyze various dynamical properties, such as, stability, of the underlying system. For practical illustration of the proposed formalization, we present a formal analysis of anthropomorphic mechatronic prosthetic hand [24].

1.4 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we provide preliminaries that includes a brief introduction of HOL Light theorem prover, multivariable calculus theories of HOL Light, basic concepts of bond graphs along with the extraction of state equations from a working example of mass-spring-damper system, an algorithm describing the step for the formal analysis of bond graphs and some set of assumptions or rules for the formal modelling of bond graphs. Chapter 3 presents the formalization of the fundamental components of BG representations of different systems. We provide the definition of stability in Chapter 4 of the thesis. Chapter 4 presents the formal verification of various properties of BG model, definition of stability, general stability theorems for stability analysis of a BG representation of a system and the properties of stability analysis. Next, in Chapter 5, we illustrate the practical effectiveness of the proposed formalization of bond graphs by successfully applying it for the formal verification of the anthropomoric mechatronic prosthetic hand and stability analysis of the corresponding hand. Finally, Chapter 6 concludes the thesis and outlines some future research directions.

Chapter 2

Preliminaries

In this chapter, we provide an introduction to the HOL Light theorem prover and a brief overview about the multivariate calculus theories of HOL Light. We also describe bond graph representations with an example, algorithms (steps) for the bond graph based analysis of systems' dynamics and some set of assumptions for the proposed formalization of bond graphs. The purpose is to introduce the basic concepts of both HOL Light and bond graph that are necessary for understading the rest of the thesis.

2.1 HOL Light Theorem Prover

HOL Light is an interactive theorem prover developed by John Harrison in 1996 at the University of Cambridge. It belongs to the HOL theorem prover family and is used for conducting proofs in higher-order logic. HOL Light has been extensively used as a tool for the formal verification of both hardware and software systems along with the formalization of pure mathematical theorems.
For secure theorem proving, the logic in HOL Light is built on top of the strongly-typed functional programming language known as objective CAML (OCaml). HOL Light has the smallest trusted core (i.e., approximately 400 lines of OCaml code) compared to all other HOL theorem provers. Soundness is assured as every new theorem must be verified by applying the basic axioms and primitive inference rules or any other previously verified theorems/inference rules.

2.1.1 Terms and Types

The logic of HOL Light is based on λ-calculus (lambda-calculus). There are four kinds of term: constants, variables, function applications and function abstractions. The special feature of HOL Light is that every term has a well-defined type (number, set, function, etc.) Semantically, types denote sets and terms denote members of these sets.

2.1.2 Theories

A HOL Light theory is a collection of types, constants, axioms, definitions and theorems and is built in a hierarchical structure. A theory can inherit definitions and theorems from other available HOL Light theories. These theories are available to a user and can be reloaded in the HOL Light system by an easy command. We utilized HOL Light theories of boolean algebra, arithmetic, real, complex numbers, sets, lists, vectors, differentiation and integration in our work. The availability of these built-in mathematical theories is the main motivation behind choosing HOL Light for our proposed work.

2.1.3 Writing Proofs

HOL Light supports two types of interactive proof styles namely forward and backward. In forward proof, the user starts with already proved theorems and applies inference rules to reach the desired theorem. While in the backward proof, the user starts with the desired theorem and applies tactics (ML function that breaks a goal into simpler subgoals) to reduce it to simpler intermediate subgoals. The user can prove some of these intermediate goals by matching assumptions or axioms. These steps are repeated for the remaining subgoals until the user is left with no further subgoals and this concludes that the desired theorem is proved. In

simple words, backward proof method is the reverse of the forward proof method. Mostly, backward proof method is prefered over forward method as it does not requires the exact details of a proof in advance unlike forward method.
There are many automatic proof procedures and proof assistants available in HOL Light which help the user in concluding a proof efficiently. To prove a goal, user provides necessary tactics to the proof editor while other steps are proved automatically by automatic proof procedures.

2.1.4 HOL Light Symbols

The following Table provides some frequently used mathematical interpretations of HOL Light symbols and functions in this thesis.

HOL Light Symbols	**Standard Symbols**	**Meaning**
`/\`	and	Logical *and*
`\/`	or	Logical *or*
`~`	not	Logical *negation*
`==>`	$\longrightarrow$	Implication
`<=>`	=	Equality
`!x.t`	$\forall x.t$	For all $x : t$
λ`x.t`	$\lambda x.t$	Function that maps x *to t(x)*
`num`	{0,1,2,...}	Positive Integers data type
`real`	All real numbers	Real data type
`complex`	All complex numbers	Complex data type
`SUC n`	(n + 1)	Successor of natural number
`&a`	$\mathbb{N} \rightarrow \mathbb{R}$	Typecasting from Integers to Reals
`EL n L`	*element*	nth element of list L
`Append L1 L2`	*append*	Combine two lists together
`LENGTH L`	*length*	Length of list L
`(a,b)`	$a \times b$	A pair of two elements
`FST`	*fst(a,b) = a*	First component of a pair
`SND`	*snd(a,b) = b*	Second component of a pair
`{x|P(x)}`	$\{x \mid P(x)\}$	Set of all x such that *P(x)*

Table 2.1: HOL Light Symbols and Functions

2.1.5 Multivariable Calculus Theories in HOL Light

In HOL Light, a n-dimensional vector is represented as a $\mathbb{R}^{\mathbb{N}}$ column matrix with all of its elements as real number. All of the vector operations are then handled as matrix manipulations. Thus, a complex number is defined by the data-type $\mathbb{R}^2$, i.e, a column matrix having two elements. Similarly, a real number can be expressed by a 1-dimensional vector $\mathbb{R}^1$ or a number on the real line $\mathbb{R}$. In multivariate calculus theories of HOL Light, all theorems have been formally verified for functions with data-type $\mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{R}^{\mathbb{M}}$.
We provide some of the frequently used HOL Light functions in our proposed formalization.

Definition 1 Cx

```
⊢ ∀a.  Cx a = complex (a,&0)
```

The function `Cx` accepts a real number and returns its equivalent complex number with imaginary part equal to zero. Here the operator `&` type-casts a natural number to its corresponding real number.

Definition 2 Re and lift

```
⊢ ∀z.  Re z = z$1
⊢ ∀x.  lift x = (lambda i.  x)
```

The function `Re` accepts a complex number and returns its real part. The notation `z$i` represents the i^{th} component of a vector `z`. The function `lift` accepts a real number and maps it to a 1-dimensional vector using `lambda` operator.

Definition 3 Integral and Derivative of a Vector-valued Function

```
⊢ ∀f i.  integral i f = (@y.(f has_integral y) i)
⊢ ∀f net.  vector_derivative f net =
                    (@f'.(f has_vector_derivative f') net)
```

The function `integral` accepts an integrand function `f`:$\mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{R}^{\mathbb{M}}$ and a vector space `i`:$\mathbb{R}^{\mathbb{N}} \rightarrow \mathbb{B}$, which defines the region of integration and returns the corresponding vector integral. Here $\mathbb{B}$ is the boolean data type. The function `has_integral` defines the same relationship in the relational form. The function `vector_derivative` accepts a function `f`:$\mathbb{R}^1 \rightarrow \mathbb{R}^{\mathbb{M}}$ that needs to be differentiated and a `net`:$\mathbb{R}^1 \rightarrow \mathbb{B}$ that defines the point at which function `f` has to be differentiated and returns a vector of data-type $\mathbb{R}^{\mathbb{M}}$ representing the differential of `f` at `net`. The function `has_vector_derivative` defines the same relationship in the relational form. The Hilbert choice operator `@`:$(\alpha \rightarrow$ `bool`$) \rightarrow \alpha$ returns value of the integral and differential, if they exist.

2.2 Bond Graphs

In this section, we describe the theory behind bond graphs in detail as well as an illustrative example that goes through the steps of extracting system equations. The concept of bond graph (BG) was introduced by H.M. Paynter and he believed that the energy and power are two fundamental variables of all physical systems' interactions. BG is a domain independent graphical approach for modelling physical systems to describe the dynamic behaviour. The ability of BG to model large and complex systems from simple and universal concepts makes it an attractive technique for formalization.

2.2.1 Notation

A Bond graph [6] (BG) is a directed graph consisting of bonds and components that are connected together as shown in Figure 2.2. A power bond is the most powerful component of a BG bridging any two components of BG and providing an exchanged power between them. It is represented by a half arrow whose head indicates the direction of a positive power flow, as shown in Figure 2.1. It represents a bond with positive direction towards B and negative power direction towards A.

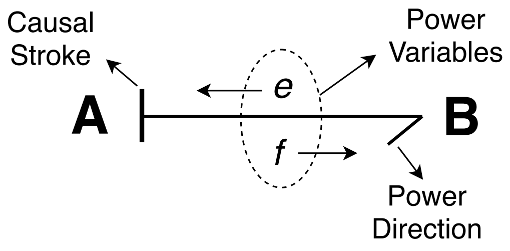


Figure 2.1: Bond Representation

2.2.2 Generalized Variables

BG models are based on three different types of analogies/groups known as signal, component and connection analogies [10]. There are four types of signals in BG known as effort (e), flow (f), integrated effort (p) and integrated flow (q) signals. These signals capture information about different phenomenas of a domain/area. In the hydraulic domain, total pressure (e), volume flow rate (f), pressure momentum (p) and volume (q) are represented by these corresponding signals [5], respectively.

Since power is a product of effort and flow signals, therefore, a power bond is composed of effort and flow signals (variables) as shown in Figure 2.1. The other two signals, i.e., integrated effort and flow signals belong to a class of energy variables. The integrated effort or generalized momentum is mathematically expressed as a time integral over the effort signal, as follows:

$$p(t) = \int^{t} e(t)dt = p_0 + \int_{0}^{t} e(t)dt \tag{2.1}$$

Where p_0 represents the value of p at time $t = 0$. Similarly, the integrated flow also known as generalized displacement is mathematically described as a time integral of a flow signal.

$$q(t) = \int^{t} f(t)dt = q_0 + \int_{0}^{t} f(t)dt \tag{2.2}$$

Where q_0 represents the value of p at time $t = 0$. Therefore, the effort and flow signals are mathematically represented as the differentials of the generalized momentum and displacement, respectively.

$$e(t) = \dot{p}(t) \tag{2.3}$$

$$f(t) = \dot{q}(t) \tag{2.4}$$

2.2.3 Causality

The simulation of a bond graph model is primarily based on the order of the computation of the effort and flow variables. This order is represented by a perpendicular bar (causal stroke) added to an end of a bond (tail of the arrow, as shown in Figure 2.1), thus indicating the corresponding variables acting as an input (cause) and output (effect), respectively. This cause and effect phenomenon is generally known as causality, indicating the direction of the effort and flow signals (variables) in a bond graph model. In Figure 2.1, e is the input to A and f is the output from A. Similarly, f acts as the input to B and e acts as an output.

A user should have prior knowledge of causality, i.e., how to assign it manually, to construct the bond graph models of the complex physical systems. Many approaches exist in the literature [17] for a systematic assignment of the casuality. However, Sequential Causality Assignment Procedure (SCAP) is preferred [4, 6] due to the systematic process and provides state-space representation of a system.

2.2.4 Components

There are nine basic elements/components, catagorized into four groups according to their characteristics, as shown in the following Table 2.2.

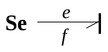
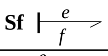
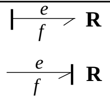
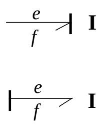
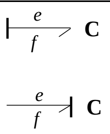
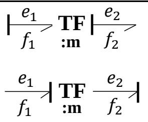
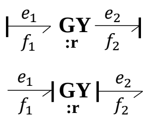
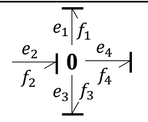
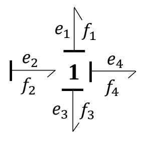
Group	Name	Symbol	Causal Equation	Meaning and Causality Rule
Supply Sources	Effort Source		$e(t)$ is known	Output of Se, Sf is effort and flow *Rule*: causality is compulsory It is an input for a system
	Flow Source		$f(t)$ is known	
Passive Elements Dissipator	(Generalized) Resistor		$e(t) = R.f(t)$ $f(t) = \frac{e(t)}{R}$	Output is an effort *Rule*: free causality Output is a flow variable
Energy Store	(Generalized) Inductor		$f(t) = \frac{1}{I}\int e(t)\,d(t) = \frac{P(t)}{I}$ $e(t) = I.\dot{f}(t)$	Flow is an output *Rule*: integral causality Effort is an output *Rule*: derivative causality
	(Generalized) Capacitor		$e(t) = \frac{1}{C}\int f(t)\,d(t) = \frac{q(t)}{C}$ $f(t) = C.\dot{e}(t)$	Effort is an output *Rule*: integral causality Flow is an output *Rule*: derivative causality
Reversible Transformation Transducers	(Generalized) Transformer		$e_1 = m.e_2\,, f_2 = m.f_1$ $e_2 = \frac{1}{m}.e_1\,, f_1 = \frac{1}{m}.f_2$	Only one effort and one flow are inputs *Rule*: only one causal stroke near TF
	(Generalized) Gyrator		$e_1 = r.f_2\,, e_2 = r.f_1$ $f_2 = \frac{1}{r}.e_1\,, f_1 = \frac{1}{r}.e_2$	Two efforts and two flows are inputs *Rule*: two or no causal stroke near GY
Distribution Junctions	Common effort junction, 0-junction, flow junction		$e_2 = e_1 = e_3 = e_4\,,$ $f_2 - f_1 - f_3 - f_4 = 0$	Only one effort is input *Rule*: only one bond can have causal stroke towards 0-junction
	Common flow junction, 1-junction, effort junction		$f_2 = f_1 = f_3 = f_4\,,$ $e_2 - e_1 - e_3 - e_4 = 0$	Only one flow is input *Rule*: only one bond can have causal stroke away from 1-junction

Table 2.2: Basic Components of Bond Graph

Component analogies are further catagorized into three groups namely supply, passive elements and reversible transformation. The supply group contains sources of power variables. The Passive elements contains storage elements I and C as well as dissipative element R. The reversible transformation group contains transducers TF and GY to convert one form of energy to another. Connection analogies consist of junctions 0 and 1 capturing the summation and equality laws.

2.2.5 Terminologies of Bond Graph

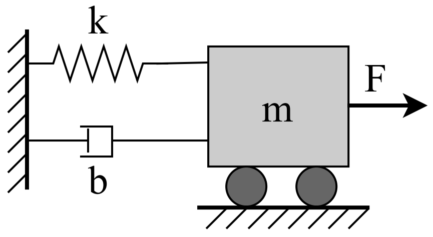
This table provides description of some important terminologies/properties of BG depicted in Figure 2.2 and 2.4.

Name	**Description**
Strong Bond	A single bond that cause effort in 0-junction and flow in 1-junction
Passive Element	A one port element which stores input power as potential energy (C-element), as kinetic energy (I-element) or transforms it into dissipative power (R-element)
Causal Bond Graph	A bond graph is called causally completed or causal if the causal stroke known as causality is added on an end of each bond
Causal Path	A sequence of bonds with/without a transformer in between having causality at the same end of all bonds and a sequence of bonds with a gyrator in between, and all the bonds of one side of the gyrator having same end causality while all the bonds on the other side with causality on opposite end. That means gyrator changes the direction of effort/flow variables [4]. A causal path can be a backward or forward or both depending upon the junction structure, elements and causality
Branch	A branch is a series of junctions having parent-child relationship. Two different sequences of junctions can be connceted with a common bond or a two-port element. Thus, one of the junction's sequence acts as parent branch and the other one as child
Causal Loop	A causal loop is a closed causal path with bonds (of the child branch) either connected to a similar junction or two different junctions of the parent branch

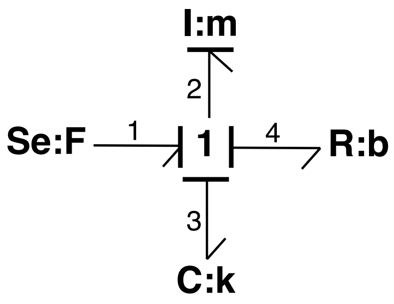
Table 2.3: Terminologies of a Bond Graph Representation

2.2.6 Illustrative Example

In this section, we illustrate a bond graph based analysis of a most commonly used example of Mass-Spring-Damper (MSD) system [18] in the mechanical domain. Figure 2.2 represents MSD system and its corresponding bond graph representation.



(a) Mass-Spring-Damper (MSD) System



(b) Bond Graph

Figure 2.2: MSD System and its Corresponding Bond Graph Representation

Generally, the bond graph representation capturing the dynamics of a system is based on the transformation/mapping of system's components to their bond graph model counterpart and it varies according to the systems from various domains, such as, electrical, mechatronic and medicine [26]. For the case of MSD system (Figure 2.2a), the Force F is mapped to S_e. Similarly, the Mass (m) is mapped to I. Moreover, Spring (k) and Damper (b) are mapped to energy storing element C and passive element R, as shown in Figure 2.2b [6]. In MSD system, sum of all forces acting on the mass is equal to zero, i.e., equilibrium state. All of the components move with the same velocity when applied force. The force and velocity are mapped onto the effort and flow variables. The 1_*junction* means that the value of flow variable (velocity) through all connected bonds (components) is the same, and the summation of effort variables (force) considering the power direction of bonds is equal to zero, as shown in Table 2.2. Each bond is labelled by a number and causality is assigned on every bond by following SCAP approach. Thus, the BG represented in Figure 2.2b is known as causal bond graph. In 1*_junction*, only one bond is responsible for the flow variable (cause), known as strong bond, which is bond number 2 as illustrated in Figure 2.2b.

To obtain a mathematical model (state-space model) of the given BG representation, we need to apply laws of components and junctions given in Table 2.2. The mathematical equations of various components are expressed as follows:

$$e_1(t) = F(t) \tag{2.5}$$

$$f_2(t) = \frac{1}{m} * (e_0 + \int_0^t e_2(t)\ d(t))$$

By using Equation 2.1, we can rewrite the equation for mass (m) as:

$$f_2(t) = \frac{p_2(t)}{m} \tag{2.6}$$

Similarly, an equation for the spring can be represented in terms of energy variable (q) by using Equation 2.2.

$$e_3(t) = \frac{1}{k} * (f_0 + \int_0^t f_2(t)\ d(t))$$

$$e_3(t) = \frac{q_3(t)}{k} \tag{2.7}$$

The equation of the dissipative component (damper) depends algebraically on the input as:

$$e_4(t) = b\ .\ f_4(t)$$

Now, we apply both laws of 1*_junction*, i.e., summation and equality for the case of MSD system. Since there is only one bond, i.e,. Bond 1, which has a positive power direction, as shown in Figure 2.2b. Therefore, all of the effort variables except e_1 have negative sign in the application of the summation law.

$$e_1 - e_2 - e_3 - e_4 = 0 \tag{2.8}$$

Similarly, since Bond 2 is the strong bond, as shown in Figure 2.2b. Therefore, the equality law for 1_*junction* is mathematically expressed as follows:

$$f_2 = f_1 = f_3 = f_4 \tag{2.9}$$

Next, to obtain the state equations, i.e., equations of the energy storing components, we use equations of the components, i.e., Equations (2.5, 2.7, **??**) in Equation (2.8).

$$F(t) - e_2(t) - \frac{1}{k} \ . \ q_3(t) - b \ . \ f_4(t) = 0$$

We are calculating state equation for mass (Bond 2) and all the entries of the above equation are in the form of generalized momentum (p) and generalized displacement (q) except variable $f_4(t)$. Thus, to convert $f_4(t)$ into an energy variable (p or q), we follow the causal strokes of the bonds till we reach an energy storing (I, C) or a source (S_e, S_f) component with integral causality. In Figure 2.2b, we follow the causal strokes from Bond 4 to 2 and by back propagation $f_{2,4}$ of the causal strokes in junction structure (1_*junction*), we can rewrite above equation using Equation 2.9:

$$F(t) - e_2(t) - \frac{1}{k} \ . \ q_3(t) - b \ . \ f_2(t) = 0$$

By using Equation 2.6 and 2.3, above equation can be rewritten as:

$$\dot{p}_2(t) = e_2(t) = -\frac{b}{m} \ . \ p_2(t) - \frac{1}{k} \ . \ q_3(t) + F(t) \tag{2.10}$$

Similarly, the equation for the storage component k is as follows:

$$f_3(t) = f_2(t)$$

By using Equation 2.6 and 2.4 in the above equation, we obtain final form of the state equation for component k (Bond 3) as follows:

$$\dot{q}_3(t) = f_3(t) = \frac{1}{m} \, . \, p_2(t) \tag{2.11}$$

Finally, by applying various properties of vectors and matrices, Equations (2.10) and (2.11) can be transformed to their corresponding state-space models as follows:

$$\dot{x}(t) = A \, . \, x(t) + B \, . \, u(t)$$

$$\begin{bmatrix} \dot{p}_2(t) \\ \dot{q}_3(t) \end{bmatrix} = \begin{bmatrix} -\frac{b}{m} & -\frac{1}{k} \\ \frac{1}{m} & 0 \end{bmatrix} \begin{bmatrix} p_2(t) \\ q_3(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} F(t) \end{bmatrix} \tag{2.12}$$

The above state-space model corresponding to the BG representations is used for analyzing various properties of the systems, such as, stability.

2.3 Algorithm/Flow for Bond graph based Formal Analysis

This section provides an algorithm (step that need to be followed) for performing the bond graph based formal analysis of the dynamics of systems.

Step 1: Take a bond graph representation as an input from the user. The input BG should be:

(a) Causally completed

(b) Simplified after applying the simpification rules if necessary

Step 2: Apply different cases depending on the junction, its components and their causality.

Step 3: Apply laws of a junction, i.e., the equality or summation laws, one by one, based on the nature of the bond causality and junction.

Step 4: If the BG representation has only one junction, go to Step 9, otherwise move to the next step.

Step 5: Start following a causal path.

Step 6: When a terminating causal element is found in the path, go to Step 9, otherwise, go to the next step.

Step 7: Follow forward or backward path, or both, based on the causality of bonds and the nature of a junction.

Step 8: Go back to Step 6.

Step 9: Extract Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DEAs) of the components of a junction.

Step 10: Simplify the differential equations to obtain the state equations.

Step 11: Generate state-space models from the state equations.

Step 12: Analyze various properties of the system, such as, stability.

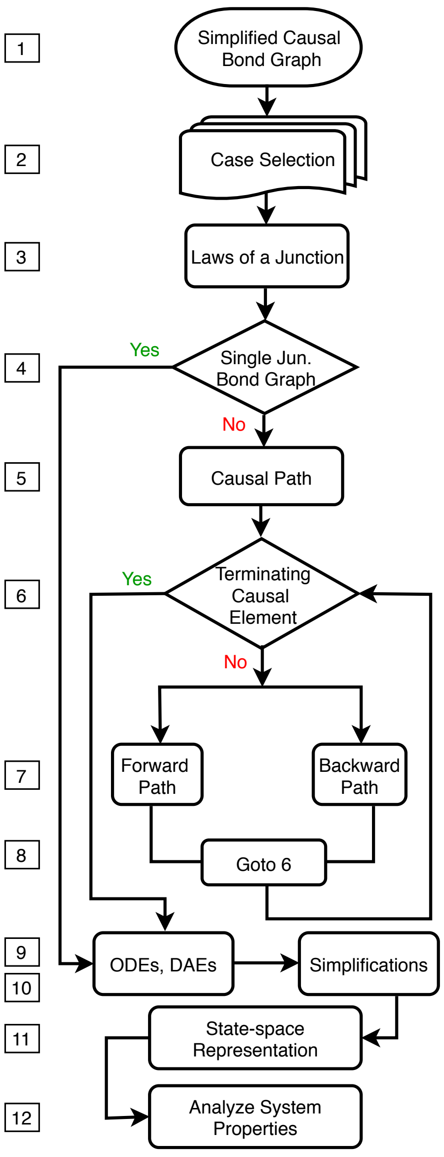


Figure 2.3: Algorithm for Bond Graph based Analysis of Systems

2.4 Some Rules/Assumptions for the Formalization of Bond Graphs

This section provides rules that are followed, to obtain a relabeled BG representation, during the formalization of the bond graphs in HOL Light. Figure 2.4 provides two different representations of an arbitrary bond graph, i.e., the given BG representation and the relabelled BG representation after applying some rules/assumptions to facilitate the corresponding formalization of the bond graphs. The rules for the formalization of the bond graph representation are as follows:

1. Arrange all one-port components/elements on the top of the corresponding junction and label all bonds of a junction in anti-clockwise direction as shown, in Figure 2.4b.

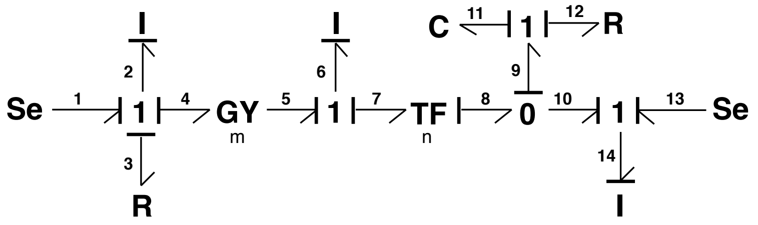
2. A bond $\mathbf{B}_{pqr}$ is labelled according to its branch `p`, junction `q` and bond position `r` in a BG representation using integers. This applies to all bonds present in a BG representation. For example, Bond 1 in Figure 2.4a is relabelled as Bond 113, as shown in Figure 2.4b. Bond 113 is the third bond of the first junction, which is an element of the first (parent) branch. Similarly, Bond 11 of Figure 2.4a is relabelled as Bond 211 representing the first bond of the first junction that lies in second (child) branch, as shown in Figure 2.4b.

3. One port (a connecting bond, Se/Sf, I, C, R) and two port (TF, GY) components/elements of a BG are also distinguished by using integers from 0 to 6, respectively. For example, in Figure 2.4b, Bond 133 or 142 is a connecting bond/-common bond (connecting 0 and 1 junction), so integer 0 is assigned to it in our formalization of BG. To recognize TF and GY, integer 5 and 6 are used respectively in HOL Light. Bond 113 and 143 are connected to a source component (Se), thus, the integer value 1 is assigned to it.

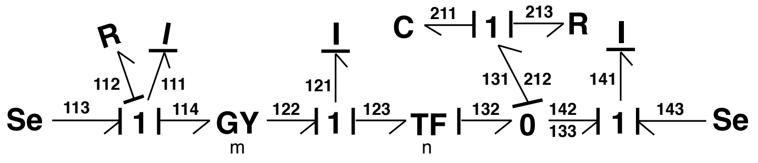
4. The causality of a bond towards respective junction is represented by the

Boolean variables **T** and **F** for the causal stroke in the opposite direction of the junction. The causality of Bond 133 is **F** with respect to 0_*junction*, whereas, the causality of Bond 142 is **T** with respect to 1_*junction*.

5. The direction of a bond is a Boolean variable **T** when it is towards the junction and **F** for the opposite direction. Bond 113 has a positive direction, therefore, it is represented by **T** and Bond 112 is represented by **F** in HOL Light.



(a) Bond Graph Representation



(b) Relabelling of Bond Graph Using Rules

Figure 2.4: Representations of a Bond Graph

6. The modulus of a two-port component is represented in the form of a list containing modulus of effort and flow variables. A two-port component (GY, TF) consists of two equations as described in Table 2.2. For example, in Figure 2.4a, the list of the transformer component for the Bond 7 and 8 is in the form of $\left[\frac{\mathtt{1}}{\mathtt{n}}, \mathtt{n}\right]$ and $\left[\mathtt{n}, \frac{\mathtt{1}}{\mathtt{n}}\right]$, respectively.

7. 1_*junction* is represented by a Boolean value **T** and **F** presents the 0-junction.

8. If a branch appears on a junction, its presence is represented by **T** and the absence by **F**. Only 0_*junction* has a branch in Figure 2.4, so its presence in HOL Light is represented by a Boolean variable **T**.

9. If a junction has a single or multiple branches, the bond numbers associated to the connecting/common bonds of a parent branch are used to differentiate branches. Otherwise, integer 0 is used. The branch on 0_*junction* is recognized by value 131, because Bond 131 is the connecting bond of both branches.

Chapter 3

Formalization of Bond Graph Representation

This chapter provides formal definitions of the building blocks of bond graph which can be utilized to formalize a wide variety of systems in the higher-order-logic. We have built our formalization of bond graph upon foundational definitions discussed in Section 2.1.5.

3.1 Bond Graph Representation

This section provides the formalization based on the higher-order-logic of the bond graph representation using HOL Light. A bond is generally represented by its various components, such as, causality, power direction, branch (common bond), type of element (Se, Sf, I, C, R, TF, GY), effort and flow. We model a bond as a 6-tuple using the type abbreviation in HOL Light to define new types as follows:

Definition 4 Bond Graph

```
new_type_abbrev ("bond", `:causality # direction # (branch # num)
                 # ele_type # mod # (effort # flow)`)
new_type_abbrev ("bonds", `:(bond) list`)
new_type_abbrev ("jun", `:jun_num # jun_type # bonds`)
new_type_abbrev ("jun_list", `:(jun) list`)
```

where the first element of the 6-tuple captures the causality of the bond. Similarly, the second element, i.e., direction models power direction. The third element of the 6-tuple, i.e., `(branch # num)` provides the information of a branch as a pair of Boolean and an integer data type, where the first element of pair represents branch presence and the second element models the position of common bond. The next two elements of the 6-tuple, i.e., `ele_type` and `mod` model the type of a component (it can take integer values from 0 to 6) and the modulus of the two-port elements in the form of a list consisting values of the complex data type as described in Section 2.4, respectively. The last element of the 6-tuple is itself a pair of function of type $\mathbb{R}^1 \rightarrow \mathbb{R}^2$, providing the effort and flow of the bond. Similarly, the second type (bonds) provides a list of bonds of a single junction of a BG representation. The next type (jun) is a 3-tuple composed of junction number (integer), type of a junction (it takes a Boolean value True and False for the junctions 1 and 0, respectively) and bonds of a junction. Finally, the type `jun_list` provides a list of all the junctions that are present in a BG, thus capturing the overall BG representation of a system.

3.2 Generalized Variables

As discussed in Section 2.2.2, e and f are power variables while `momentum` and `displacement` represents energy variables. Their formalized functions are as follows:

Definition 5 Generalized Variables of Bond Graph

```
⊢ ∀e f.  power e f = (λt.  e t * f t)
⊢ ∀e.  momentum e p0 =
          (λt.  p0 + integral (interval [lift(&0), t]) e)
⊢ ∀f.  displacement f q0 =
          (λt.  q0 + integral (interval [lift(&0), t]) f)
⊢ ∀e.  momentum_der e p0 =
          (λt.  vector_derivative (momentum e p0) (at t))
⊢ ∀f.  displacement_der f q0 =
          (λt.  vector_derivative (displacement f q0) (at t))
```

3.3 Causality

Now, we model the causality of a bond, as discussed in Section 2.2.3 and is formalized in HOL Light as the following recursive functions.

Definition 6 Causality

```
⊢ ∀j i k.  causality_f j i 0 = 0 ∧
            causality_f j i (SUC k) =
             (if causality j i (SUC k) = F then SUC k
             else causality_f j i k)
⊢ ∀j i k.  causality_t j i 0 = 0 ∧
            causality_t j i (SUC k) =
             (if causality j i (SUC k) = T then SUC k
             else causality_t j i k)
```

The $\texttt{causality}_\texttt{f}$ accept a list of junction `j`, an element `i` of the list `j` and a bond `k` of the junction `i`, and returns the causality of the bond `k`, i.e., it returns a Boolean value `T` for the case of direction of bond away from a junction, as described in Section 2.4. Similarly, the function $\texttt{causality}_\texttt{t}$ returns true (`T`) if the direction of a bond `k` is towards a junction `i`.

3.4 Power Direction

A power bond carries information of causality as well as its power direction as shown in Figure 2.1. Its formalized function is as follows:

Definition 7 Direction of a Bond

```
⊢ ∀j i k.  bond_direction_cond j i k =
         (if bond_direction j i k = T then Cx (&1) else -- Cx (&1)
```

The function `bond_direction` accepts a list of junction `j`, an element `i` of `j` and a bond `k` of the junction `i` and returns `T` for a positive power direction, otherwise it returns false. It basically extracts the second element of the 6-tuple capturing a bond graph representation. Moreover, the function `bond_direction_cond` assigns the complex numbers 1 and -1 to the Boolean values `T` and `F`, respectively.

3.5 Components

In this section, we provide formalized functions for the components namely, Se, Sf, I, C, R, TF, GY, 0 and 1 junctions, of a BG representation, as discussed in Section 2.2.4. There are two types of one-port components as follows:

3.5.1 Active Elements

In bond graphs, sources are called active elements with one variable equal to zero. $\texttt{src}_\texttt{e}$ supply effort to the system and $\texttt{src}_\texttt{f}$ supply flow to the system.

Definition 8 Active Elements

$\vdash \forall \texttt{e}.\ \ \texttt{src}_\texttt{e}\ \texttt{e} = (\lambda \texttt{t}.\ \ \texttt{Cx e})$

$\vdash \forall \texttt{f}.\ \ \texttt{src}_\texttt{f}\ \texttt{f} = (\lambda \texttt{t}.\ \ \texttt{Cx f})$

3.5.2 Passive Elements

These are the passive elements of bond graph as described in Table 2.2. Both integral and differential causal elements are formalized. $\texttt{compliance}_\texttt{e}$ represents integrally causal C-element and $\texttt{compliance}_\texttt{f}$ captures differential causality.

Definition 9 Passive Elements

$\vdash \forall \texttt{R f.}\ \ \texttt{res}_\texttt{e}\ \texttt{R f = (}\lambda\texttt{t.}\ \ \texttt{Cx R} * \texttt{f t)}$

$\vdash \forall \texttt{R e.}\ \ \texttt{res}_\texttt{f}\ \texttt{R e = (}\lambda\texttt{t.}\ \ \texttt{Cx}\left(\frac{\texttt{\&1}}{\texttt{R}}\right) * \texttt{e t)}$

$\vdash \forall \texttt{C f.}\ \ \texttt{compliance}_\texttt{e}\ \texttt{C f q}_0\texttt{= (}\lambda\texttt{t.}\ \ \texttt{Cx}\left(\frac{\texttt{\&1}}{\texttt{C}}\right) * \texttt{displacement f q}_0\ \texttt{t)}$

$\vdash \forall \texttt{C e.}\ \ \texttt{compliance}_\texttt{f}\ \texttt{C e = (}\lambda\texttt{t.}\ \ \texttt{Cx C} * \texttt{vector\_derivative e (at t))}$

$\vdash \forall \texttt{L f.}\ \ \texttt{inductor}_\texttt{e}\ \texttt{L f = (}\lambda\texttt{t.}\ \ \texttt{Cx L} * \texttt{vector\_derivative f (at t))}$

$\vdash \forall \texttt{L e.}\ \ \texttt{inductor}_\texttt{f}\ \texttt{L e p}_0\ \texttt{= (}\lambda\texttt{t.}\ \ \texttt{Cx}\left(\frac{\texttt{\&1}}{\texttt{L}}\right) * \texttt{momentum e p}_0\ \texttt{t)}$

3.6 Modulus of Transducers

A two-port element consists of two equations with different modulus for different variables (effort and flow), as shown in Table 2.2.

Definition 10 Modulus of Two-Port Elements

```
⊢ ∀j i k.  modulus_cond j i k =
           (if causality j i k = T then EL 1 (modulus j i k)
            else EL 0 (modulus j i k))
```

The function `modulus_cond` verifies the causality of a bond towards a junction and provides first element (flow variable) of the list `modulus` accordingly. Similarly, if the causality of a bond is not towards the junction then the modulus of efffort variable is returned.

3.7 Laws of a Junction

A single junction consists of multi-ports and captures equality and summation laws for variables (effort, flow) depending on the type of junction (0,1). We formalize each case separately and provide the equality laws for the junctions 0 and 1 (presented as Equation (2.9) for MSD system example in Section 2.2.6) of a bond graph representation in HOL Light as follows:

Definition 11 Equality Laws

```
⊢ ∀j i k1 k2 t.  jun_0_e j i k1 k2 t =
                  bond_effort j i k1 t = bond_effort j i k2 t
⊢ ∀j i k1 k2 t.  jun_1_f j i k1 k2 t =
                  bond_flow j i k1 t = bond_flow j i k2 t
```

The functions `jun_0_e` and `jun_1_f` provide laws of equality for junctions 0 and 1, respectively. The function `jun_0_e` accepts a list of junctions `j:(jun)list`, an element `i` of the list `j` representing a junction (0*_junction*), where the equality law is applied and two different bonds `k1` and `k2` of the junction `i`, and returns law of equality as an equation of effort. It uses the function `bond_effort` to capture the effort (e) variable. The function `jun_1_f` provides law of equality as an equation of flow for 1*_junction* using the function `bond_flow`.

Next, we formalize the summation laws for the junctions 0 and 1 of a bond graph representation in HOL Light as follows:

Definition 12 Summation Laws

```
⊢ ∀j i t.  jun_1_e j i t = vsum (0...bonds_lenght j i - 3)
                          (λk.  bond_effort_wd j i k t)
⊢ ∀j i t.  jun_0_f j i t = vsum (0...bonds_lenght j i - 3)
                          (λk.  bond_flow_wd j i k t)
```

The function `jun_1_e` accepts a list of junctions `j:(jun) list`, an element `i` of the list `j`, representing a 1_*junction*, and a time variable `t` and returns the summation of the effort variables of all bonds in the junction i, i.e., 1_*junction* except the last two bonds. Here, the function `bond_effort_wd` models the effort variable of a bond with power direction. Similarly, the function `jun_0_f` provides a summation law for 0_*junction* by skipping the last two bonds. Moreover, any two junctions are connected by a common bond, For example, junction 0 and 1 are connected by Bond 10 as shown in Figure 2.4a and by the laws of summation and equality, their flow variables for both junctions are mathematically expressed as:

$$f_8(t) - f_9(t) - f_{10}(t) = 0, \;\; f_8(t) = n * f_7(t), \;\; f_{10}(t) = f_{14}(t)$$

We can simplify above equations by substituting the value of flow variables of common bond and transformer (TF), i.e., Bond 10 and 8, from second and third equation to first equation, resulting into the following equation:

$$n * f_7(t) - f_9(t) - f_{14}(t) = 0$$

So, in order to obtain final form of the equations, we eliminate these intermediate steps (substitution of equations) by excluding last two bonds of a junction.

3.8 Strong Bond

A strong bond is responsible for providing effort and flow in 0 and 1 junctions, respectively, and its formalized definitions are as follows:

Definition 13 Strong Bond

$\vdash \forall$`j i.  strong_bond_f j i = causality`$_{\mathtt{f}}$` j i (bonds_length j i - 1)`

$\vdash \forall$`j i.  strong_bond_t j i = causality`$_{\mathtt{t}}$` j i (bonds_length j i - 1)`

The functions `strong_bond_f` and `strong_bond_t` use the functions $\texttt{causality}_{\texttt{f}}$ and $\texttt{causality}_{\texttt{t}}$ that check all bonds of a junction to detect the strong bonds with causality towards a junction and in the opposite direction of a junction, respectively. Here, the function `bond_length` provides the total number of bonds of a single junction.

3.9 Skipped Bonds and Summation Law of Last Junction

Next, we model the bonds (second last and last) that are skipped during summation law of the last junction as follows:

Definition 14 Skipped Bonds of the Last Junction

```
⊢ ∀j i t.  bw_last_jun_e j i t = (if i = 0
           then snd_last_bond_dir j i * snd_last_bond_e j i t
           else Cx (&0))
⊢ ∀j i t.  bw_last_jun_f j i t = (if i = 0
           then snd_last_bond_dir j i * snd_last_bond_f j i t
           else Cx (&0))
⊢ ∀j i t.  fw_last_jun_e j i t = (if i = 0
           then last_bond_dir j i * last_bond_e j i t
           else Cx (&0))
⊢ ∀j i t.  fw_last_jun_f j i t = (if i = 0
           then last_bond_dir j i * last_bond_f j i t
           else Cx (&0))
```

In the summation laws, provided as Definition 12, we skiped last two bonds of a junction that need to be modeled. By following a causal path (backward and forward), if we reach at the first junction (in backward path) or the last junction (in forward path) , i.e., i = 0 of a list `j`, we must add those skipped bonds , i.e., the second last and last bond, of junction `i` with a power direction, otherwise add a zero. The function `bw_last_jun_e` provides the effort variable with power direction modeled as `snd_last_bond_dir` of the second last bond of the junction `i`, when following a backward causal path. Similarly, following a forward causal path, the function `fw_last_jun_e` provides the effort variable with power direction modeled as `last_bond_dir` of the last bond of the junction `i`.

Definition 15 Summation Law on the Last Junction

```
⊢ ∀j i t.  bw_jun_e_sum j i t =
           (jun_1_e j i t) + (bw_last_jun_e j i t)
⊢ ∀j i t.  bw_jun_f_sum j i t =
           (jun_0_f j i t) + (bw_last_jun_f j i t)
⊢ ∀j i t.  fw_jun_e_sum j i t =
           (jun_1_e j i t) + (fw_last_jun_e j i t)
⊢ ∀j i t.  fw_jun_f_sum j i t =
           (jun_0_f j i t) + (fw_last_jun_f j i t)
```

The functions `bw_jun_e_sum` and `fw_jun_e_sum` accept a list of junctions `j:(jun) list`, an element `i` of the list `j` and a time variable `t` and returns the summation of the effort variables of 1_*junction* (last junction in the junction list `j`) including the second last and last bonds of the junction `i`, when following a backward and forward paths, respectively. Similarly, the functions `bw_jun_f_sum` and `fw_jun_f_sum` provide summation of the flow variables of 0_*junction* including second last and last bonds of the junction `i`, following the backward and forward paths.

3.10 Causal Paths

A causal path is a sequence of bonds consisting common bond/two-port components in between and is followed to find the variables of the state equations. A Causal path consists of forward, backward or both as described in Table 2.2.5.

3.10.1 Backward Path

The backward path follows causality of the bonds appearing in the list of junctions, in such a way that we reach at a terminating causal element. The formalized function of a backward path is as follows:

Definition 16 Backwrad Path

```
⊢ ∀j i t.  backwrd_path j 0 t = Cx(&0) ∧
  backwrd_path j (SUC i) t) =
 [P1] (if (type_of_jun j (SUC i) = F) ∧
       (snd_last_bond_type j (SUC i) = 0 ∨
        snd_last_bond_type j (SUC i) = 5) ∧
       (type_of_jun j i = F) ∨
       (type_of_jun j (SUC i) = T) ∧
       (snd_last_bond_type j (SUC i) = 0 ∨
        snd_last_bond_type j (SUC i) = 5) ∧
       (type_of_jun j i) = F) ∨
       (type_of_jun j (SUC i) = T) ∧
       (snd_last_bond_type j (SUC i) = 6) ∧
       (type_of_jun j i = F) ∨
       (type_of_jun j (SUC i) = F) ∧
       (snd_last_bond_type j (SUC i) = 6) ∧
       (type_of_jun j i) = F)
```

```
[P2] then if (strong_bond_t j i =
          snd_last_bond_of_jun j i) ∧  ∼(i = 0)
       then last_bond_modulus_cond j i ∗ backwrd_path j i t
       else if (strong_bond_t j i =
                last_bond_of_jun j i) ∧  ∼(i = 0)
            then last_bond_dir_cond j i ∗
                 last_bond_modulus_cond j i ∗
                 (bw_jun_f_sum j i t + backwrd_path j i t)
            else if (strong_bond_t j i =
                     last_bond_of_jun j i) ∧ (i = 0)
                 then last_bond_dir_cond j i ∗
                      last_bond_modulus_cond j i ∗
                      bw_jun_f_sum j i t
                 else last_bond_modulus_cond j i ∗
                      bond_effort j i (strong_bond_t j i) t
[P3] else if (type_of_jun j (SUC i)) = T) ∧
          (snd_last_bond_type j (SUC i)) = 6) ∧
          (type_of_jun j i) = T) ∨
          (type_of_jun j (SUC i) = T) ∧
          (snd_last_bond_type j (SUC i) = 0 ∨
           snd_last_bond_type j (SUC i) = 5) ∧
          (type_of_jun j i) = T) ∨
          (type_of_jun j (SUC i) = F) ∧
          (snd_last_bond_type j (SUC i)) = 0 ∨
           snd_last_bond_type j (SUC i) = 5) ∧
          (type_of_jun j i) = T) ∨
          (type_of_jun j (SUC i) = F) ∧
```

```
           (snd_last_bond_type j (SUC i) = 6) ∧
           (type_of_jun j i) = T)
[P4] then if (strong_bond_f j i =
                snd_last_bond_of_jun j i) ∧  ~(i = 0)
            then last_bond_modulus_cond j i * backwrd_path j i t
            else if (strong_bond_f j i =
                    last_bond_of_jun j i) ∧  ~(i = 0)
                then last_bond_dir_cond j i *
                    last_bond_modulus_cond j i *
                    (bw_jun_e_sum j i t + backwrd_path j i t)
                else if (strong_bond_f j i =
                        last_bond_of_jun j i) ∧ (i = 0)
                    then last_bond_dir_cond j i *
                        last_bond_modulus_cond j i *
                        bw_jun_e_sum j i t
                    else last_bond_modulus_cond j i *
                        bond_flow j i (strong_bond_f j i) t)
    else Cx(&0)
```

To find a causal path, we must have information about the starting and the upcoming junction of the path, type of the common bond between two junctions and a strong bond. The first part, i.e., P1 of the function `beckward_path` specifies a causal path starting from an arbitrary junction `i` (of any type, i.e., 0_*junction* or 1_*junction*) and a 0_*junction* (modeled using Boolean value `F` as an upcoming junction `i` in the path and both these junctions (junction `i` and 0_*junction*) share a common bond or a two-port element. The function `snd_last_bond_type` captures the type of sharing bond. In P2, the function `strong_bond_t` checks if the strong bond, with causality towards the junction, of the upcoming junction `i` is the last, second last or an arbitrary bond alongwith its position in the list of junction `j`. If the strong bond with causality towards the junction `i` (`strong_bond_t`) is the last bond of the junction `i` and the value of `i` is not equal to zero, i.e., it is not the first junction, the function `bw_jun_f_sum` applies the summation law on the junction `i` alongwith power direction `last_bond_dir_cond` and the modulus of last bond `last_bond_modulus_cond`, and keep following `backwrd_path`, until we reach a terminating element. The part P3 of the function `backward_path` is quite similar to part P1 of the function, except the upcoming junction is now 1*_junction*. Finally, part P4 behaves is in the similar way as part P2, using `strong_bond_f` for the causality of a strong bond in the opposite direction of the junction `i`. In other words, the function `backwrd_path` consists of different combinations of junctions (0 or 1), common bonds (connecting bond, TF or GY) and strong bonds (with causality towards or in the opposite direction of a junction), and provides their equations accordingly.

3.10.2 Forward Path

A forward path is similar to that backward path except the position of junctions and bonds and its formalized form is as follows:

Definition 17 Forwrad Path

```
⊢ ∀j i t.  fwrd_path (j:jun_list) 0 t = Cx(&0) ∧
  fwrd_path j (SUC i) t) =
[P1] (if (type_of_jun (rev j) (SUC i) = F) ∧
      (last_bond_type (rev j) (SUC i) = 0) ∨
       last_bond_type (rev j) (SUC i) = 5) ∧
      (type_of_jun (rev j) i) = F) ∨
      (type_of_jun (rev j) (SUC i) = T) ∧
      (last_bond_type (rev j) (SUC i) = 0 ∨
       last_bond_type (rev j) (SUC i) = 5) ∧
      (type_of_jun (rev j) i = F) ∨
      (type_of_jun (rev j) (SUC i) = T) ∧
      (last_bond_type (rev j) (SUC i) = 6) ∧
      (type_of_jun (rev j) i) = F) ∨
      (type_of_jun (rev j) (SUC i) = F) ∧
      (last_bond_type (rev j) (SUC i) = 6) ∧
      (type_of_jun (rev j) i = F)
[P2] then if (strong_bond_t (rev j) i = last_bond_of_jun (rev j) i)
          ∧ ~(i = 0)
      then snd_last_bond_modulus_cond (rev j) i *
          fwrd_path j i t
      else if (strong_bond_t (rev j) i =
              snd_last_bond_of_jun (rev j) i) ∧ ~(i = 0)
          then snd_last_bond_dir_cond j i *
              snd_last_bond_modulus_cond (rev j) i *
              (fw_jun_f_sum (rev j) i t + fwrd_path j i t)
          else if (strong_bond_t (rev j) i =
                  snd_last_bond_of_jun (rev j) i) ∧ (i = 0)
              then snd_last_bond_dir_cond j i *
                  snd_last_bond_modulus_cond (rev j) i *
                  fw_jun_f_sum (rev j) i t
              else snd_last_bond_modulus_cond (rev j) i *
              bond_effort (rev j) i (strong_bond_t (rev j) i) t
[P3] else if (type_of_jun (rev j) (SUC i) = T) ∧
         (last_bond_type (rev j) (SUC i) = 6) ∧
         (type_of_jun (rev j) i) = T) ∨
         (type_of_jun (rev j) (SUC i) = T) ∧
         (last_bond_type (rev j) (SUC i) = 0 ∨
          last_bond_type (rev j) (SUC i) = 5) ∧
         (type_of_jun (rev j) i = T) ∨
         (type_of_jun (rev j) (SUC i) = F) ∧
         (last_bond_type (rev j) (SUC i) = 0 ∨
          last_bond_type (rev j) (SUC i) = 5) ∧
         (type_of_jun (rev j) i = T) ∨
         (type_of_jun (rev j) (SUC i) = F) ∧
         (last_bond_type (rev j) (SUC i) = 6) ∧
         (type_of_jun (rev j) i) = T)
[P4] then if (strong_bond_f (rev j) i =
            last_bond_of_jun (rev j) i) ∧ ~(i = 0)
         then snd_last_bond_modulus_cond (rev j) i) *
              fwrd_path j i t
         else if (strong_bond_f (rev j) i =
                 snd_last_bond_of_jun (rev j) i) ∧ ~(i = 0)
              then snd_last_bond_dir_cond j i *
                   snd_last_bond_modulus_cond (rev j) i *
                   (fw_jun_e_sum (rev j) i t + fwrd_path j i t)
              else if (strong_bond_f (rev j) i =
                      snd_last_bond_of_jun (rev j) i) ∧ (i = 0)
                   then snd_last_bond_dir_cond j i *
                        snd_last_bond_modulus_cond (rev j) i *
                        fw_jun_e_sum (rev j) i t
                   else snd_last_bond_modulus_cond (rev j) i *
                 bond_flow (rev j) i (strong_bond_f (rev j) i) t
```

```
else Cx(&0)
```

The function `fwrd_path` accepts a list of junctions `j`, an element `i` of the list `j` and a time variable `t` and returns variable (effort or flow) or summation equation depending upon the structure of junctions, causality and elements. The parts P1 and P3 of the function `fwrd_path` are quite similar to that of `backwrd_path` except the reversed list of junctions `(rev j)`. Moreover, parts P2 and P4 are quite similar to that of `backwrd_path` except the placement of last and second last bonds.

Definition 18 Matching Junctions

```
⊢ ∀j n i.  jun_num_match j 0 i = jun_num (rev j) 0 ∧
           jun_num_match j (SUC n) i =
           (if i = jun_num (rev j) (SUC n) then SUC n
            else jun_num_match j n i)
⊢ ∀j i.  jun_match j i = jun_num_match j (LENGTH j - 1) i
⊢ ∀j i t.  final_fwrd_path j i t = fwrd_path j (jun_match j i) t
```

Since, `fwrd_path` function reverse the list of junctions `j`, thus the position of each entry (junction) of the list `j` changes in this process. However, we placed junction number `jun_num` in the 6-tuple as described in Definition 4, which remains unchanged in the reversion preocess. Thus for accuracy, it is necessary to match the i^{th} junction of the reversed list with the `jun_num` by using the function `jun_num_match`. There is only one junction in the reversed list that has a matching junction number, as each junction of a BG representation is assigned a number uniquely. The function `jun_match` checks all the junctions of list `j` for matching a junction and the final definition of forward path `final_fwrd_path` uses that resulting junction.

It is possible for a system to have bond graph representation with only one junction. as shown in Figure 2.2. In this case, there is no causal path (backward or forward), except the last and second last bonds with the effort or flow variables. We model this scenario in HOL Light as follows:

Definition 19 Unit Junction

```
⊢ ∀j i t.  backwrd_path_e j i t =
           (if LENGTH j - 1 = 0 then snd_last_bond_e j i t
            else backwrd_path j i t)
⊢ ∀j i t.  backwrd_path_f j i t =
           (if LENGTH j - 1 = 0 then snd_last_bond_f j i t)
            else backwrd_path j i t)
⊢ ∀j i t.  final_fwrd_path_e j i t =
           (if LENGTH j - 1 = 0 then last_bond_e j i t)
            else final_fwrd_path j i t)
⊢ ∀j i t.  final_fwrd_path_f j i t =
           (if LENGTH j - 1 = 0 then last_bond_f j i t)
            else final_fwrd_path j i t)
```

The functions `backwrd_path_e` and `backwrd_path_f` checks the length of the junction list `j` and if it is equal to zero then there is no backward path except the second last bond with the effort and flow variables, respectively. If this condition is not true, then there is a causal path. Similarly, `final_fwrd_path_e` and `final_fwrd_path_f` extract the effort and flow of the last bond in the case of only one junction, otherwise, there is a forward path to follow in a BG representation.

Next, we model the causal paths as the following HOL Light function:

Definition 20 Causal Paths

```
⊢ ∀j i t.  causal_paths j i t =
           last_bond_dir j i ∗ final_fwrd_path j i t +
           snd_last_bond_dir j i ∗ backwrd_path j i t
```

A causal path consists of forward or backward path or both depending upon the causality, position and structure of the junction. For example, in Figure 2.4a, to obtain summation equation (for flow variable) of Bond 9, we follow both forward and backward paths by eliminating last two bonds of 0_*junction*, but the power direction of the skipped bonds is important in the law of summation, that's why we multiply power direction of the last bond (`last_bond_dir`) with forward path (`final_fwrd_path`) and add it to backward path (`backwrd_path`) multiplied with power direction of the second last bond (`snd_last_bond_dir`).

3.11 Selection of Paths

Selection of path depends upon the position and causality of a junction (0,1). Following are the formalized funtions for these cases.

Definition 21 Path Selection based on the Position of Junction

```
⊢ ∀j i t.  search_path_e j i t =
         (if i = 0
          then snd_last_bond_dir j i * snd_last_bond_e j i t +
               last_bond_dir j i * final_fwrd_path_e j i t
          else if i = LENGTH j - 1
               then last_bond_dir j i * last_bond_e j i t +
                    snd_last_bond_dir j i * backwrd_path_e j i t
               else Cx(&0))
⊢ ∀j i t.  search_path_f j i t =
         (if i = 0
          then snd_last_bond_dir j i * snd_last_bond_f j i t +
               last_bond_dir j i * final_fwrd_path_f j i t
          else if i = LENGTH j - 1
```

```
        then last_bond_dir j i * last_bond_f j i t +
            snd_last_bond_dir j i * backwrd_path_f j i t
        else Cx(&0))
```

The functions search_path_e and search_path_f search a path for the effort and flow variables, respectively, based on the position (first or last) of a junction in a junction list j.

Definition 22 Path Selection Based on Integral Causality

```
⊢ ∀j i t.  path_select_t j i t =
        (if (strong_bond_t j i = last_bond_of_jun j i)
             ∧  ~(i = LENGTH j - 1)
         then final_fwrd_path j i t
         else if (strong_bond_t j i = snd_last_bond_of_jun j i)
                ∧  ~(i = 0)
             then backwrd_path j i t
             else bond_effort j i (strong_bond_t j i) t)
⊢ ∀j i t.  path_select_f j i t =
        (if (strong_bond_f j i = last_bond_of_jun j i)
             ∧  ~(i = LENGTH j - 1)
         then final_fwrd_path j i t
         else if (strong_bond_f j i = snd_last_bond_of_jun j i)
                ∧  ~(i = 0)
             then backwrd_path j i t
             else bond_flow j i (strong_bond_f j i) t)
```

The energy storing elements I and C exhibit integral causality, i.e., component C has a causality towards a junction, whereas I has a causality in the opposite direction. The function `path_select_t` selects a path by matching a strong bond (`strong_bond_t`) with last or second last bond of the junction `i` and its position in the list of junctions `j`. Similarly, the `path_select_f` choses a path depending upon the strong bond (`strong_bond_f`), second last bond (`snd_last_bond_of_jun`), last bond (`last_bond_of_jun`) of the junction `i` and the position of the junction.

3.11.1 Path Selection for Equality Law of a Junction

A junction consists of both equality and summmation laws. We have formalized law of equality for a junction (0,1) in HOL Light, as described in Definition 11. Now, we formalize the function of equality law for the case, when a junction is follwing a causal path as follows:

Definition 23 Path Selection based on the Type of the Junction

```
⊢ ∀j i k t.  path_selection j i k t =
          (if type_of_jun j i = F
          then (bond_effort j i k t = path_select_t j i t)
          else (bond_flow j i k t = path_select_f j i t))
```

The function `path_selection` selects a path for equality laws based on the type of a junction, which can be 0 or 1.

Similarly, we formalize equality law for the differential causal elements of a junction following causal path as follows:

Definition 24 Path Selection Based on Differential Causality

```
⊢ ∀j i k t.  path_selection_der j i k t =
       (if (type_of_jun j i = F) then
          (λt.  vector_derivative (bond_effort j i k) (at t)) =
          (λt.  vector_derivative (path_select_t j i) (at t))
       else
          (λt. vector_derivative (bond_flow j i k) (at t)) =
          (λt. vector_derivative (path_select_f j i) (at t)))
```

The causal elements I and C exhibiting the differential causality are provided in Table 2.2. The function `path_selection_der` provides equality law by taking a derivative on both sides of the equation, obtained using information of the type of junction and the causality of the strong bond.

Definition 25 Path Selection based on the Free Causality

```
⊢ ∀j i k t.  res_path_selection j i k t =
         (if causality j i k = F
          then if type_of_jun j i = F
               then bond_effort j i k t = path_select_t j i t
               else jun_sum_final j i t
          else if type_of_jun j i = F
               then jun_sum_final j i t
               else bond_flow j i k t = path_select_f j i t)
```

The resistive component of a bond graph representation has a free causality, i.e., its causality follows a structure of junctions rather than that of its components. The function `res_path_selection` covers all possible cases for the resistive element having free causality and provides both summation `jun_sum_final` and the equality law based on the causality of a resistive element and the type of a junction.

3.11.2 Path Selection for Summation Law of a Junction

Here, We provide formalized functions of the summation law (following a causal path) based on the causality, type and position of a junction as follows:

Definition 26 Summation Law based on the Type of Junctions

```
⊢ ∀j i t.  middle_jun_sum j i t =
        (if type_of_jun j i = T
         then jun_1_e j i t + causal_paths j i t
         else jun_0_f j i t + causal_paths j i t)
⊢ ∀j i t.  side_jun_sum j i t =
        (if (type_of_jun j i = T) then
         then jun_1_e j i t + search_path_e j i t
         else jun_0_f j i t + search_path_f j i t)
```

The function `type_of_jun` accepts a list of junctions `j`, an element `i` of the list `j` and returns **T** for a 1*_junction*, otherwise it returns **F**. The function `middle_jun_sum` combines summation law for effort and flow variables to the Boolean values **T** and **F**, respectively, alongwith the causal paths (`causal_paths`). A junction (0 or 1) attached with junctions on both sides or on one side only is known as middle or side junction respectively. The function `side_jun_sum` is quite similar to that of `middle_jun_sum` except the paths, `search_path_e` and `search_path_f`. These paths are selected based on the type of junction. In Figure 2.4a, 0*_junction* is representing a middle junction, while 1*_junctions* on both sides of the BG representation are side junctions.

Definition 27 Summation Law based on the Position of a Junction

```
⊢ ∀j i t.  jun_sum j i t =
         (if i = 0 ∨ i = LENGTH j - 1
          then side_jun_sum j i t
          else middle_jun_sum j i t)
```

The summation law of a junction depends on its position in a BG representation. The function `jun_sum` checks the position of a junction (first or last) and applies sumation law by (`side_jun_sum`), otherwise it applies sumation law on the middle jun (`middle_jun_sum`).

Definition 28 Summation Law of a Junction

```
⊢ ∀j i t.  jun_sum_final j i t = (jun_sum j i t = Cx(&0))
```

The summation law of a junction is, the sum of all the effort or flow variables equal to zero, thus, the function `jun_sum_final` provides the final form (summation equal to zero) of summation law.

Definition 29 Summation Law based on the Differential Causality

```
⊢ ∀j i t.  jun_sum_der j i t =
        (λt.  vector_derivative (jun_sum j i) (at t)) =
        (λt.  vector_derivative (λt.  vec 0) (at t))
```

The function `jun_sum_der` accepts a list of junctions `j`, an element `i` of the list `j` and a time variable `t`, and returns a differential form of the summation law.

3.12 Presence of a Branch

The concept of a branch is of great importance and is given in Table 2.2.5. It is discussed in Section 2.4 with the help of an example, and its formalized function is as follows:

Definition 30 Branch Presence

```
⊢ ∀j j1 p q p1 q1 t.  branch_presence j j1 p q p1 q1 t =
  (if (branch j p q = T) ∧ (branch j1 p1 q1 = T)
   then if branch_num j p q = branch_num j1 p1 q1
        then if type_of_ele j1 p1 q1 = 6
             then APPEND
```

```
                [bond_effort j p q t =
                EL 1 (modulus j1 p1 q1) ∗ bond_flow j1 p1 q1 t]
                [bond_flow j p q t =
                EL 0 (modulus j1 p1 q1) ∗ bond_effort j1 p1 q1 t]
            else APPEND
                [bond_effort j p q t =
                EL 0 (modulus j1 p1 q1) ∗ bond_effort j1 p1 q1 t]
                [bond_flow j p q t) =
                EL 1 (modulus j1 p1 q1) ∗ bond_flow j1 p1 q1 t]
        else [ ]
   else [ ])
⊢ ∀ j j1 t.  branch_main j j1 t =
   (if j = [ ] ∨ j1 = [ ]
    then [ ]
    else REVERSE (branch_jun j j1 (LENGTH j - 1) t))
```

The function `branch_presence` accepts two different list of junctions `j`,`j1`, a junction `p` of the list `j`, a bond `p1` of the junction `j`, a different junction `q` of the list `j1`, a bond `q1` of the junction `j1` and a time variable `t`, and returns a list of equations capturing the equality laws (for effort and flow) for common bonds of `j` and `j1`. Here `j` and `j1` represent parent and child branch, respectively. The presence of a branch in a BG representation is associated with Boolean value **T** in the 6-tuple. In a BG representation, multiple branches may appear on a junction, so in order to distinguish them, we assign a number `branch_num` to them as described in Section 2.4. The connecting/common bond of two branches can be a two-port element (TF, GY) or a simple power bond. The function `type_of_ele` returns Boolean value **T**, if the connecting bond is a gyrator (GY) and provides respective equations (effort and flow) with the modulus of GY, otherwise, it returns **F** and provides respective equations (effort and flow) with the modulus of transformer (TF) or a connecting bond (with modulus 1). For example, in Figure 2.4b, Bond 131 or 212 is a common bond connecting 0_*junction* to 1_*junction*. The function `branch_main` accepts two list of junctions `j,j1`, which are connected together, and a time variable `t`, and returns empty list, if the lists `j` or `j1` are empty. Otherwise, the function `branch_jun` checks all the junction of list `j` for branch presence.

3.13 Presence of a Causal Loop

The causal loop is a type of a branch with both ends attached to a junction as described in Table 2.2.5 and its formalized function is as follows:

Definition 31 Causal Loop Presence

```
⊢ ∀j i k i1 n t.  loop_presence j i k i1 n t =
  (if branch j i1 n = T
   then if branch_num j i k = branch_num j i1 n
        then if type_of_ele j i1 n = 6
             then APPEND
                  [bond_effort j i k t =
                   EL 1 (modulus j i1 n) ∗ bond_flow j i1 n t]
                  [bond_flow j i k t =
                   EL 0 (modulus j i1 n) ∗ bond_effort j i1 n t]
             else APPEND
                  [bond_effort j i k t =
                   EL 0 (modulus j i1 n ∗ bond_effort j i1 n t]
                  [bond_flow j i k t =
                   EL 1 (modulus j i1 n) ∗ bond_flow j i1 n t])
        else [ ]
else [ ])
```

The function `loop_presence` accepts a list of junctions `j`, a junction `i` of the list `j`, a bond `k` of the junction `i`, a junction `i1` of the list `j`, a bond `n` of the junction `i1` and a time variable `t`, and provides a list of equations capturing the equality laws (for effort and flow) for common bonds. Firstly, the function `loop_presence` checks the presence of a branch on the junction `i1`, match the branch numbers of both junctions `i` and `i1` (to check that they are connected together or not) and check the presence of gyrator component (integer value 6) as it changes effort to flow and vice versa. If the branch numbers of both junctions do not match or there is no branch then the returned list is empty.

Definition 32 Recursive Function for Loop Bonds

```
⊢ ∀j i k i1 n t.  loop_bonds j i k i1 0 t =
  (if 0 = k then [ ] else loop_presence j i k i1 0 t) ∧
  loop_bonds j i k i1 (SUC n) t =
  (if SUC n = k then [ ]
   else APPEND (loop_presence j i k i1 (SUC n) t)
             (loop_bonds j i k i1 n t))
⊢ ∀j i k m t.  loop_bonds_lst j i k i1 t =
           loop_bonds j i k i1 (bonds_length j i1 - 1) t
```

The function `loop_bonds` ensures that if a loop (a branch with both ends connected to junctions) exists on any junctions then both of its starting and the ending bonds (`k, n`) do not have similar bond numbers, i.e, to distinguish both connecting bonds of the loop. The function `loop_bonds_lst` accepts a list of junctions `j`, a junction `i` of the list `j`, a bond `k` of the junction `i`, a junction `i1` from the list `j` and ensures the presence of a loop in a junction `i1` by checking all of its bonds using (`bonds_length`).

Definition 33 Recursive Function for Loop Junctions

```
⊢ ∀j i m i1 t.  loop_jun j i k 0 t = loop_bonds_lst j i k 0 t ∧
            loop_jun j i k (SUC i1) t =
            APPEND (loop_bonds_lst j i k (SUC i1) t)
                  (loop_jun j i k i1 t)))
⊢ ∀j i k t.loop_jun_lst j i m t = loop_jun j i k (LENGTH j - 1) t
```

A causal loop is a closed causal path with bonds either connected to a similar junction or two different junctions as described in Table 2.3. The function `loop_jun` ensures the presence of a loop in a junction `i1` by checking junctions of the `j` recursively. The function `loop_jun_lst` checks all the junctions of the list `j` to find the junction on which, connecting bond of the loop is connected.

Definition 34 Causal Loop

```
⊢ ∀j i k t.  causal_loop j i k t =
  (if (branch j i k = T) ∧  ~(branch_num j i k = 0)
   then REVERSE (loop_jun_lst j i k t)
   else [ ]
```

A `causal_loop` ensures the presence of a loop in a BG representation and provides its equations by using function `loop_jun_lst`.

3.14 Cases of a BG Represention

Here, we define some cases of a BG representation based on our formalization as follows:

Definition 35 Cases

```
⊢ ∀j i t.  frst_ordr_ele_a j i t = [jun_sum_final j i t]
⊢ ∀j i k t.  frst_ordr_ele_b j i k t = [path_selection j i k t]
⊢ ∀j i k t.  zero_ordr_ele j i k t = [res_path_selection j i k t]
⊢ ∀j i t.  diff_causal_ele_a j i t = [jun_sum_der j i t]
⊢ ∀j i k t.  diff_causal_ele_b j i k t = [path_selection_der j i k t]
⊢ ∀j i k t.  branch_type_a j i k t =
          APPEND [jun_sum_final j i t] (causal_loop j i k t)
⊢ ∀j i k t.  branch_type_b j i k t =
          APPEND [path_selection j i k t] (causal_loop j i k t)
```

The function `frst_ordr_ele_a` and `frst_ordr_ele_b` model all those cases, where the junction, components and causality appear in such a way that summation and equality laws are deducted, respectively. For example, in Figure 2.4b, all 1_*junctions* with a component I (having integral causality) provides summation law and the 1_*junction* with component C (having integral causality) provides equality law by following causal paths. The function `zero_ordr_ele` models junctions with resistive component (R). The functions `diff_causal_ele_a` and `diff_causal_ele_b` are similar to first order cases except the elements with differential causality. These functions provide differential equations of the BG representation. Lastly, the functions `branch_type_a` and `branch_type_b` are also similar to first order cases except the presenece of a causal loop.

3.15 Cases Selection Procedure

The case selection procedure mainly depends upon the causality, type of components and the junction. In this procedure, every bond of a junction present in a bond graph representation is examined.

Definition 36 Case Selection

```
⊢ _ ∀j i k t.  case_selection j i k t =
(if (type_of_jun j i = T) ∧ (type_of_ele j i k = 2) ∧
     (causality j i k = F) ∨ (type_of_jun j i = F) ∧
     (type_of_ele j i k = 3) ∧ (causality j i k = T)
then frst_ordr_ele_a j i t
else if (type_of_jun j i = T) ∧ (type_of_ele j i k = 3) ∧
(causality j i k = T) ∨ (type_of_jun j i = F) ∧
(type_of_ele j i k = 2) ∧ (causality j i k = F)
then frst_ordr_ele_b j i k t
else if
   (type_of_jun j i = T) ∧ (type_of_ele j i k = 4) ∧
   (causality j i k = T) ∨ (type_of_jun j i = F) ∧
   (type_of_ele j i k = 4) ∧ (causality j i k = F) ∨
   (type_of_jun j i = T) ∧ (type_of_ele j i k = 4) ∧
   (causality j i k = F) ∨ (type_of_jun j i = F) ∧
   (type_of_ele j i k = 4) ∧ (causality j i k = T)
then zero_ordr_ele j i k t
else if
   (type_of_jun j i = T) ∧ (type_of_ele j i k = 2) ∧
   (causality j i k = T) ∨ (type_of_jun j i = F) ∧
   (type_of_ele j i k = 3) ∧ (causality j i k = F)
then diff_causal_ele_b j i k t
else if
   (type_of_jun j i = T) ∧ (type_of_ele j i k = 3) ∧
   (causality j i k = F) ∨ (type_of_jun j i = F) ∧
   (type_of_ele j i k = 2) ∧ (causality j i k = T)
then diff_causal_ele_a j i t
else if
```

```
    ((type_of_jun j i = T) ∧ (branch j i k = T) ∧
    (causality j i k = F)) ∧ (type_of_ele j i k = 0 ∨
    type_of_ele j i k = 5 ∨ type_of_ele j i k = 6) ∨
    ((type_of_jun j i = F) ∧ (branch j i k) = T) ∧
    (causality j i k) = T)) ∧ (type_of_ele j i k = 0 ∨
    type_of_ele j i k = 5 ∨ type_of_ele j i k = 6)
  then branch_type_a j i k t
  else if
    ((type_of_jun j i = T) ∧ (branch j i k = T) ∧
    (causality j i k = T)) ∧ (type_of_ele j i k = 0 ∨
    type_of_ele j i k = 5 ∨ type_of_ele j i k = 6) ∨
    ((type_of_jun j i = F) ∧ (branch j i k = T) ∧
    (causality j i k = F)) ∧ (type_of_ele j i k = 0 ∨
    type_of_ele j i k = 5 ∨ type_of_ele j i k = 6)
  then branch_type_b j i k t
  else [ ])
```

The `case_selection` takes a list of junctions `j`, an element `i` of the list `j`, a bond number `k` and a time variable `t` and provides a suitable case for a junction depending on the causality of bonds, type of elements and the type of junction.

3.15.1 Bond Selection

The following formalized function is for the bonds of a junction. It applies case selection procedure on every bond of a junction.

Definition 37 Recursive Function for the Bonds of a Junction

```
⊢ ∀j i k t.  bond_selection j i 0 t = case_selection j i 0 t ∧
  bond_selection j i (SUC k) t =
  APPEND (case_selection j i (SUC k) t) (bond_selection j i k t)
⊢ ∀j i k t.  bond_selection_lst j i t =
         bond_selection j i (bonds_length j i - 1) t
```

The function `bond_selection` is a recursive function, which applies `case_selection` to the bonds of a junction. The function `bond_selection_lst` applies case selection to all the bonds of a junction .

3.15.2 Junction Selection

Now, we apply case selection procedure on every junction of a bond graph representation, which is in the form of a list in our formalization of HOL Light.

Definition 38 Recursive function for Junctions

```
⊢ ∀j i t.  jun_selection j i t = bond_selection_lst j 0 t ∧
  jun_selection j (SUC i) t =
    APPEND (bond_selection_lst j (SUC i) t) (jun_selection j i t)
⊢ ∀j t.  bg_main j t =
       (if j = [ ] then [ ] else
       REVERSE (jun_selection j (LENGTH j - 1) t))
```

The function `jun_selection` is a recursive function, which applies the function `bond_selection_lst` to all the bonds of the junction `i` of the list `j`. The function `bg_main` is the main function of BG formalization, which accepts only two parameters, i.e., a list of junctions `j` and time variable `t`, and returns all the required equations of a BG representation by applying the function `jun_selection` on all the junctions of the list `j`.

3.16 State-Space Model

Finally, the state-space model for a bond graph representation is formalized as follows:

Definition 39 State-space Representation

```
⊢ ∀A B x x_der u.
    ss_model A B x x_der u = (x_der = A ** x + B ** u)
```

The function `ss_model` accepts a system matrix `A`:$\mathbb{C}^{N\times N}$, input matrix `B`:$\mathbb{C}^{P\times N}$, state vector `x`:$\mathbb{C}^{N}$, derivative of state vector with respect to time `x_der`:$\mathbb{C}^{N}$ and input vector `u`:$\mathbb{C}^{P}$, and provide a state-space model.

Chapter 4

Formal Verification of Stability of Bond Graphs

In this chapter, we provide the stability property of BG representation, various properties of complex matrices and the general stability theorems for stability analysis of a BG representation of a system.
Stability is an important control characteristic of physical systems that dampens out any oscillation in the performance of the system caused by various disturbances, and thus restores systems to the equilibrium conditions [19]. Thus, a stable system provides a stable response/output to a bounded input. The bond graph representation of a system is expressed as state-space models, which are based on vectors and matrices, in particular system matrix. Stability of a system depends on the location of the poles in a complex plane, which are the eigenvalues of the system's matrix. Based on the location of the poles in a complex plane, any system can be categorized as of three types, namely, stable, marginally stable and unstable system. For a stable and unstable system, the eigenvalues of the system's matrix lie in the left and right half of the complex plane, respectively. In Figure 4.1, green and red dots are representing eigenvalues of a stable and unstable system in the left and right half of the complex plane, respectively. Similarly, for a marginally stable system, the eigenvalues of the system's matrix lie on the imaginary axis of the complex plane. Yellow dots (eigenvalues) exactly lie on the imaginary axis representing a marginally stable system as shown in the following Figure 4.1.

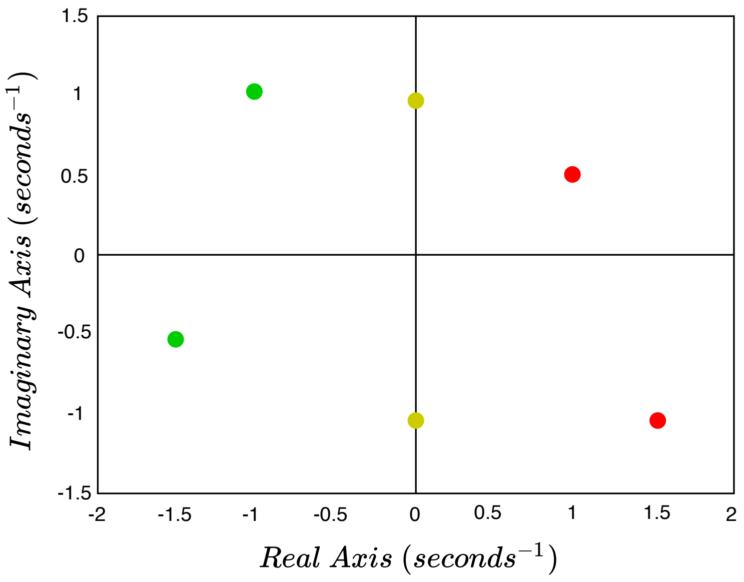


Figure 4.1: Stability Regions

For a system matrix A, the characteristic equation is described as:

$$Ax = cx$$

Where x is the eigenvector and c represents the eigenvalues. We can find out the eigenvalues of the system matrix `A` by solving the following equations involving determinant of a matrix.

$$|A - cI| = 0$$

Where I is an identity matrix. Stability has been formalized in HOL Light for the case of polynomial. However, it cannot incorporate the stability of the state-space models. We model the notion of stability, incorporating the state-space models, in HOL Light as follows:

Definition 40 Stability

```
⊢ ∀A. stable_sys A =
        ~ ({c | cdet (A - c %%% cmat (Cx (&1))) = Cx (&0) ∧
                Re (c) < &0} = EMPTY)
⊢ ∀A. unstable_sys A =
        ~ ({c | cdet (A - c %%% cmat (Cx (&1))) = Cx (&0) ∧
                Re (c) > &0} = EMPTY)
⊢ ∀A. marginally_stable_sys A =
        ~ ({c | cdet (A - c %%% cmat (Cx (&1))) = Cx (&0) ∧
                Re (c) = &0} = EMPTY)
```

The functions `stable_sys`:$\mathbb{C}^{N\times N} \rightarrow$ bool accepts a complex-valued matrix `A` and returns a Boolean value (`T`), if both conditions are satisfied, i.e., the first condition provides the eigenvalues of the state-space matrix `A`, whereas, the second condition ensures that the real part of the eigenvalues (roots) lie in left half of the complex plane and ensure a stable system. Here, `cdet` models a determinant of a complex-valued matrix. Similarly, `cmat` provides a complex-valued identity matrix. Moreover, the operator `%%%`:$\mathbb{C} \rightarrow \mathbb{C}^{N\times M} \rightarrow \mathbb{C}^{N\times M}$, provides a scalar multiplication of a complex-valued matrix. Similarly, the functions `unstable_sys` and `marginally_stable_sys` provide an unstable and a marginally stable system, respectively.

Next, we verify some important properties of the state-space models of the given bond graph representations, when they represent some special kind of matrices, such as, upper triangular, lower triangular and diagonal [22, 25]. We formalize these matrices as follows:

Definition 41 Complex Matrices

```
⊢ ∀A. ut_cmatrix A = (!i j.1 ≤ i ∧ i ≤ dimindex(:M) ∧
                      1 ≤ j ∧ j ≤ dimindex(:N) ∧
                      (j < i)
```

```
                         ⇒ (A$i$j = Cx(&0))
⊢ ∀A. lt_cmatrix A = (!i j.1 ≤ i ∧ i ≤ dimindex(:M) ∧
                         1 ≤ j ∧ j ≤ dimindex(:N) ∧
                         (i < j)
                         ⇒ (A$i$j = Cx (&0))
⊢ ∀A. diagonal_cmatrix A = !i j.1 ≤ i ∧ i ≤ dimindex(:M) ∧
                         1 ≤ j ∧ j ≤ dimindex(:N) ∧
                         ~ (i = j)
                         ⇒ (A$i$j= Cx (&0))
```

The function `ut_cmatrix` models a complex-valued upper triangular matrix, which is a square matrix with all of its entries below the main diagonal are zero. Similarly, a square matrix whose entries above the main diagonal are zero is called lower triangular matrix and is formalized in HOL Light as a function `lt_matrix`. The function `diagonal_cmatrix` provides a digonal matrix, which is a special case of triangular matrices and have all its entries, other than diagonal, equal to zero.

Next, we formally verify important properties of complex matrices, providing stability, unstability and marginal stability under different conditions, as follows:

Theorem 1 Stable Matrix Property

```
⊢ ∀A. (diagonal_cmatrix A ∨ lt_cmatrix A ∨ ut_cmatrix A) ∧
       (stable_diagonal_ele_cond A) ⇒ stable_sys A
```

For a system to be stable, two conditions are necessary to meet as described in Definition 40. The eigenvalues of a diagonal, lower or upper triangular matrix lie on the digonal entries. Thus, the above theorem states that a system matrix `A` in the form of diagonal, lower or upper triangular matrix, with all the digonal entries located in the left half of the complex plane, is stable. The condition`stable_diagonal_ele_cond` verifies that the real part of diagonal entries of `A` is less than zero (left half plane).

Theorem 2 Un-Stable Matrix Property

$\vdash \forall$`A. (diagonal_cmatrix A` $\lor$ `lt_cmatrix A` $\lor$ `ut_cmatrix A)` $\land$

`(unstable_diagonal_ele_cond A)` $\Rightarrow$ `unstable_sys A`

The above theorem states that a system matrix `A` in the form of diagonal, lower or upper triangular matrix is unstable, with all of its digonal entries (eigenvalues) in the right half of the complex plane, i.e, real part of the entries is greator than zero.

Theorem 3 Marginally Stable Matrix Property

$\vdash \forall$`A. (diagonal_cmatrix A` $\lor$ `lt_cmatrix A` $\lor$ `ut_cmatrix A)` $\land$

`(marg_stable_diagonal_ele_cond A)` $\Rightarrow$ `marginally_stable_sys A`

Similarly, the above theorem states that the given matrix is marginally stable `marginally_stable_sys` if it is in the form of a diagonal, lower or upper triangular matrix and the eigenvalues are located on the imaginary axis of the complex plane.

4.1 Polynomial Stability

For analyzing stable polynomials (quadratic and cubic in our case), we used already formalized quadratic and cubic formulas [2] and on the same note, we defined our polynomials to verify stability conditions. We verified unstable and marginally stable polynomials additionally.

Lemma 1 Stability of Polynomials

$\vdash \forall$`a b c.`

[A1] $\sim \big($`Cx (&0) = Cx a`$\big) \land$

[A2] $\big($`&0 <` $\dfrac{\texttt{b}}{\texttt{a}}$ $\land$ $\big($`b pow 2 - &4 * a * c < &0` $\lor$

$$
\begin{aligned}
&\qquad\qquad\texttt{b pow 2 - \&4 * a * c = \&0}\big) \vee \\
&\texttt{\&0 < b pow 2 - \&4 * a * c} \wedge \\
&\big(\texttt{a < \&0} \wedge \big(\texttt{b < } \sqrt{\texttt{(b pow 2 - \&4 * a * c)}} \vee \\
&\qquad\qquad \sqrt{\texttt{(b pow 2 - \&4 * a * c)}} \texttt{ < -- b}\big) \vee \\
&\texttt{\&0 < a} \wedge \big(\sqrt{\texttt{(b pow 2 - \&4 * a * c)}} \texttt{ < b} \vee \\
&\qquad\qquad \texttt{-- b < } \sqrt{\texttt{(b pow 2 - \&4 * a * c)}}\ \big)\big)\big) \\
&\Rightarrow \sim \big(\texttt{!x. Cx a * x pow 2 + Cx b * x + Cx c = Cx (\&0)} \wedge \\
&\qquad \texttt{Re x < \&0} \Leftrightarrow \texttt{x IN } \{\}\big)
\end{aligned}
$$

$$
\begin{aligned}
&\vdash \forall \texttt{a b1 c1 d1 r x.} \\
&\quad \textbf{[A1]} \sim \big(\texttt{Cx (\&0) = Cx a}\big) \wedge \\
&\quad \textbf{[A2]}\ \texttt{(Cx b = Cx b1 + Cx (a * r)} \wedge \\
&\quad \textbf{[A3]}\ \texttt{Cx c = Cx c1 + Cx (b1 * r)} \wedge \\
&\quad \textbf{[A4]}\ \texttt{Cx d = Cx (c1 * r))} \wedge \\
&\quad \textbf{[A5]}\ \big(\texttt{\&0 < r} \vee \\
&\qquad \texttt{\&0 < } \frac{\texttt{b1}}{\texttt{a}} \wedge \big(\texttt{b1 pow 2 - \&4 * a * c1 < \&0} \vee \\
&\qquad\qquad\qquad \texttt{b1 pow 2 - \&4 * a * c1 = \&0}\big) \vee \\
&\qquad \texttt{\&0 < b1 pow 2 - \&4 * a * c1} \wedge \\
&\qquad \big(\texttt{a < \&0} \wedge \big(\texttt{b1 < } \sqrt{\texttt{(b1 pow 2 - \&4 * a * c1)}} \vee \\
&\qquad\qquad\qquad \sqrt{\texttt{(b pow 2 - \&4 * a * c)}} \texttt{ < -- b1}\big) \vee \\
&\qquad \texttt{\&0 < a} \wedge \big(\sqrt{\texttt{(b1 pow 2 - \&4 * a * c1)}} \texttt{ < b1} \vee \\
&\qquad\qquad\qquad \texttt{-- b1 < } \sqrt{\texttt{(b1 pow 2 - \&4 * a * c1)}}\ \big)\big)\big) \\
&\qquad \Rightarrow \sim \big(\texttt{!x.Cx a * x pow 3 + Cx b * x pow 2 +} \\
&\qquad\qquad \texttt{Cx c * x + Cx d = Cx (\&0)} \wedge \texttt{ Re x < \&0} \Leftrightarrow \texttt{x IN } \{\}\big)
\end{aligned}
$$

First lemma provides formally verified results for the stable quadratic polynomial and covers all possible cases depending upon the nature of the discriminant and coefficients of the polynomial. Second lemma presents set of conditions for a cubic polynomial to be stable. Assumption 1 ensures that the given polynomial is cubic and next three assumptions are for the factor decomposition of the polynomial.`x` is a complex variable while all other variables are of real data type.

4.2 Matrix Stability

In the state-space representation, we deal with matrices and its corresponding operations. For the stability of a given system, entries of the matrix `A` are analyzed in these theorems. We have verified theorems for the stable, unstable and marginally stable matrices in HOL Light. In this thesis, we only present stable matrices of dimension 2 and 3.

Theorem 4 Stable Matrix

$\vdash \forall\texttt{a11 a12 a21 a22.}$

[A1] $\texttt{\&0 < (-- a11 - a22)} \land$

[A2] $\big(\texttt{(-- a11 - a22) pow 2 - \&4} * \texttt{(a11} * \texttt{a22 - a12} * \texttt{a21) < \&0} \lor$

$\texttt{(-- a11 - a22) pow 2 - \&4} * \texttt{(a11} * \texttt{a22 - a12} * \texttt{a21) = \&0}\big) \lor$

$\big(\texttt{\&0 < (-- a11 - a22) pow 2 - \&4} * \texttt{(a11} * \texttt{a22 - a12} * \texttt{a21)} \land$

$\big(\sqrt{\texttt{(-- a11 - a22) pow 2 - \&4} * \texttt{(a11} * \texttt{a22 - a12} * \texttt{a21)}} <$

$\texttt{(-- a11 - a22)} \lor \texttt{(a11 + a22) <}$

$\sqrt{\texttt{(-- a11 - a22) pow 2 - \&4} * \texttt{(a11} * \texttt{a22 - a12} * \texttt{a21)}}\ \big)\big)\big)$

$$\Rightarrow \texttt{stable\_sys} \begin{bmatrix} \texttt{Cx a11} & \texttt{Cx a12} \\ \texttt{Cx a21} & \texttt{Cx a22} \end{bmatrix}$$

$\vdash \forall\texttt{a11 a12 a13 a21 a22 a23 a31 a32 a33 b1 c1 d1 r.}$

[A1] $\texttt{Cx (-- a33 - a22 - a11) = Cx b1 + Cx r} \land$

[A2] $\texttt{Cx (-- a13} * \texttt{a31 - a12} * \texttt{a21 - a23} * \texttt{a32 +}$

$\texttt{a22} * \texttt{a33 + a11} * \texttt{a33 + a11} * \texttt{a22) = Cx c1 + Cx (b1} * \texttt{r)} \land$

[A3] $\texttt{Cx (-- a11} * \texttt{a22} * \texttt{a33 - a12} * \texttt{a31} * \texttt{a23 - a13} *$

$\texttt{a21} * \texttt{a32 + a11} * \texttt{a23} * \texttt{a32 + a12} * \texttt{a21} * \texttt{a33 +}$

a13 * a31 * a22) = Cx (c1 * r) $\wedge$

[A4] (&0 < r $\vee$

(&0 < b1 $\wedge$ (b1 pow 2 - &4 * c1 < &0 $\vee$

b1 pow 2 - &4 * c1 = &0) $\vee$

(&0 < b1 pow 2 - &4 * c1 $\wedge$ ((b1 pow 2 - &4 * c1) < b1 $\vee$

-- b1 < $\sqrt{\texttt{(b1 pow 2 - \&4 * c1)}}$))))

$$\Rightarrow \texttt{stable\_sys} \begin{bmatrix} \texttt{Cx a11} & \texttt{Cx a12} & \texttt{Cx a13} \\ \texttt{Cx a21} & \texttt{Cx a22} & \texttt{Cx a23} \\ \texttt{Cx a31} & \texttt{Cx a32} & \texttt{Cx a33} \end{bmatrix}$$

Above theorem provides all possible conditions on the entries of a 2×2 and 3×3 matrix to be stable. We used Lemma 1 in the formalization of matrix stability for solving characteristics polynomial of the given matrix. This theorem is formally verified using complex matrix theory built by us and multivariate complex and real analysis theories available in the library of the HOL Light theorem prover.

Chapter 5

Case Study: Anthropomoric Mechatronic Prosthetic Hand

In this chapter, we demonstrate the usefulness of our proposed framework by utilizing it for the analysis of a real-time system. For illustration purposes, we present the formal analysis of an Anthropomoric Mechatronic Hand by using the formal library of BG representation, given in Chapter 3. Mechatronic robotic hand is a safety-critical example of a real-time system.

Anthropomoric Mechatronic Hand [24] is a robotic hand consisting of electrical and mechanical components and is capable of conducting movements like a human hand and is widely used in robotics, such as industrial, service, surgical robots [9, 15, 21], etc. A prosthetic robotic hand improves the lives of the handicapped individuals by restoring the functions of the missing body parts. The accuracy and stability of such systems are crucial to replicate the desired movements of a human hand.

A human hand consists of four fingers and a thumb involving flexion (flex.), extension (ext.), abduction (abd.), adduction (add.), up and down movements. The flexion and extension are the movements of thumb and fingers, i.e., moving the tip of the finger/thumb towards and away from the palm, respectively. Similarly, adduction and abduction move the fingers/thumb towards and away from the middle finger, respectively. Lastly, the up and down are the movements of the thumb [27].

An anthropomoric mechatronic Prothestic hand replicates the structure of human hand consisting of bones, joints and muscles by using frames, pulley-string mechanisms and electrical actuators, respectively.

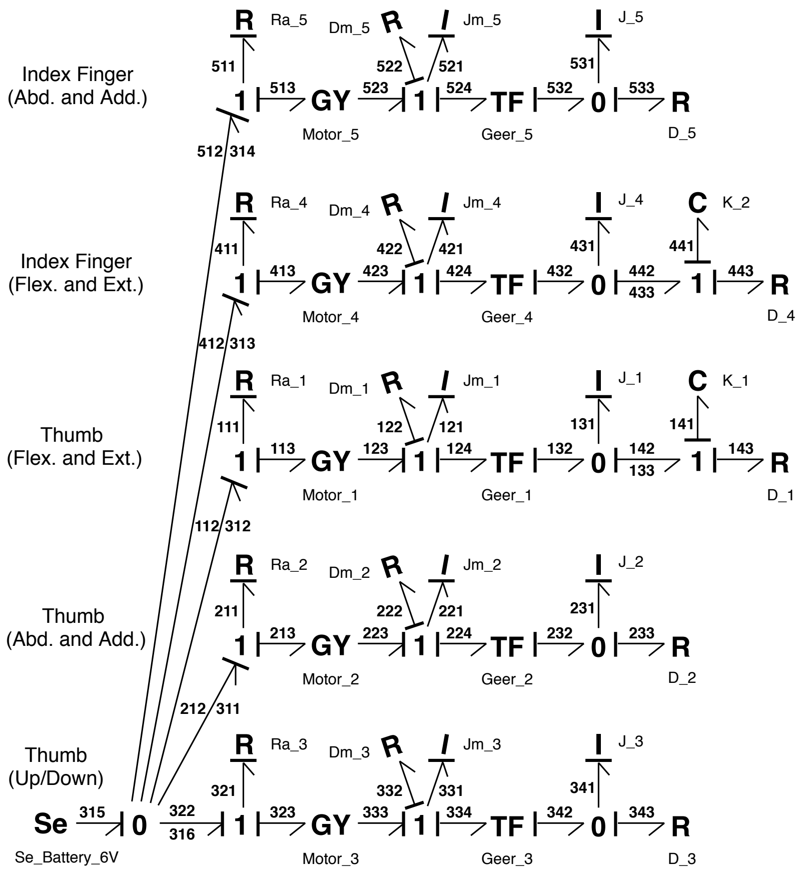


Figure 5.1: Bond Graph of Anthropomoric Mechatronic Prosthetic Hand

A bond graph representation of an anthropomoric mechatronic prosthetic hand is given in Figure 5.1. It consists BG models for index fingers for the case of abduction and adduction, and flexion and extention. It also contains BG models for thumb in the case of flexion and extention, abduction and adduction, and up and down. We formalize each of these index fingers and thumbs using our formalization of BG representation, presented in Section 3.

5.1 Formal Modeling

The first step in our formalization is to model the robotic hand according to its type configuration using BG formalization library. We model thumb (up and down), i.e., list of the junctions as the following HOL Light function:

Definition 42 Model of Thumb (Up & Down)

```
⊢ ∀ e311 f311 e312 f312 e313 f313 e314 f314 f315 e316 f316 e321
    e322 f322 e323 f323 e331 f332 e333 f333 e334 f334 e341 e342
    Ra_3 Motor_3 Jm_3 Dm_3 Geer_3 J_3 D_3 S_e p_0.
  THUMB_UP_DOWN e311 f311 e312 f312 e313 f313 e314 f314 f315 e316
   f316 e321 e322 f322 e323 f323 e331 f332 e333 f333 e334 f334
    e341 e342 Ra_3 Motor_3 Jm_3 Dm_3 Geer_3 J_3 D_3 S_e p_0 =
  [0,F,[F,F,(T,311),0,[Cx (&1); Cx (&1)],(e311,f311);
        F,F,(T,312),0,[Cx (&1); Cx (&1)],(e312,f312);
             F,F,(T,313),0,[Cx (&1); Cx (&1)],(e313,f313);
             F,F,(T,314),0,[Cx (&1); Cx (&1)],(e314,f314);
             T,T,(F,0),1,[Cx (&1); Cx (&1)],(src_e Se,f315);
             F,F,(F,0),0,[Cx (&1); Cx (&1)],(e316,f316)];
    1,T,[F,F,(F,0),4,[Cx (&1); Cx (&1)],(e321,res_f Ra_3 e321);
        T,T,(F,0),0,[Cx (&1); Cx (&1)],(e322,f322);
        T,F,(F,0),6,[Cx(&1 / Motor_3); Cx Motor_3],(e323,f323)];
    2,T,[F,F,(F,0),2,[Cx (&1); Cx (&1)],
             (momentum_der e331 p_0, inertance_f Jm_3 e331 p_0 t);
        T,F,(F,0),4,[Cx (&1); Cx (&1)],(res_e Dm_3 f332,f332);
        T,T,(F,0),6,[Cx(&1 / Motor_3); Cx Motor_3],(e333,f333);
        T,F,(F,0),5,[Cx(&1 / Geer_3); Cx Geer_3],(e334,f334)];
    3,F,[F,F,(F,0),2,[Cx (&1); Cx (&1)],
            (momentum_der e341 p_0,inertance_f J_3 e341 p_0);
     F,T,(F,0),5,[Cx Geer_3; Cx(&1 / Geer_3)],(e342,f342);
     T,F,(F,0),4,[Cx (&1); Cx (&1)],(res_e D_3 f343,f343)]]
```

Where D_3, Ra_3, Dm_3 represent the damping of pulleys, resistance and damping of the motor, respectively. The real-valued constants Jm_3, J_3 represents inertial mass of motor and frames, pulleys and strings, respectively. Motor_3, Geer_3 are the rotio/modulus of the gyrator and gear, respectively. Similarly, we model the index finger (Flex. and Ext.) in HOL Light as follows:

Definition 43 Model of Index Finger (Flex. & Ext.)

```
⊢ ∀e411 e412 f412 e413 f413 e421 f422 e423 f423 e424 f424 e431
   e432 f432 e433 f433 f441 e442 f442 f443 Ra_4 Motor_4 Jm_4
   Dm_4 Geer_4 J_4 K_2 D_4 p_0 q_0.
 INDEX_FINGER_FLEX_EXT e411 e412 f412 e413 f413 e421 f422 e423
  f423 e424 f424 e431 e432 f432 e433 f433 f441 e442 f442 f443
   Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 K_2 D_4 p_0 q_0 =
 [0,T,[F,F,(F,0),4,[Cx(&1); Cx(&1)],(e411,res_f Ra_4 e411);
             T,T,(T,313),0,[Cx (&1); Cx (&1)],(e412,f412);
             T,F,(F,0),6,[Cx (&1 / Motor_4); Cx Motor_4],(e413,f413)];
   1,T,[F,F,(F,0),2,[Cx (&1); Cx (&1)],
                  momentum_der e421, inertance_f Jm_4 e421 p_0;
      T,F,(F,0),4,[Cx (&1); Cx (&1)], res_e Dm_4 f422,f422);
      T,T,(F,0),6,[Cx (&1 / Motor_4); Cx Motor_4],(e423,f423);
      T,F,(F,0),5,[Cx (&1 / Geer_4); Cx Geer_4],(e424,f424)];
   2,F,[F,F,(F,0),2,[Cx (&1); Cx (&1)],
                  (momentum_der e431, inertance_f J_4 e431 p_0);
      F,T,(F,0),5,[Cx Geer_4; Cx (&1 / Geer_4)],(e432,f432);
```

```
     T,F,(F,0),0,[Cx (&1); Cx (&1)],(e433,f433)];
  3,T,[T,F,(F,0),3,[Cx (&1); Cx (&1)],
            (compliance_e K_2 f441 q_0, displacement_der f441 q_0);
     F,T,(F,0),0,[Cx (&1); Cx (&1)],(e442,f442);
     T,F,(F,0),4,[Cx (&1); Cx(&1)],(res_e D_4 f443,f443)]]
```

Where `D_4, Ra_4, Dm_4` represent the damping of pulleys, resistance and damping of the motor, respectively. The real-valued constants `Jm_4, J_4` represents inertial mass of motor and frames, pulleys and strings, respectively. `K_2` is the stiffness of the strings and `Motor_4, Geer_4` is the rotio of the gyrator and gear, respectively.
Now, we formalize the differential algebraic equations of the given index finger (Flex. & Ext.) as follows:

Definition 44 Equations of Index Finger (Flex. & Ext.)

```
⊢ ∀e411 e412 f412 e413 f413 e421 f422 e423 f423 e424 f424 e431
   e432 f432 e433 f433 f441 e442 f442 f443 Ra_4 Motor_4 Jm_4 Dm_4
    Geer_4 J_4 K_2 D_4 p_0 q_0 t.
  INDEX_FINGER_FLEX_EXT_EQ e411 e412 f412 e413 f413 e421 f422 e423
    f423 e424 f424 e431 e432 f432 e433 f433 f441 e442 f442 f443
    Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 K_2 D_4 p_0 q_0 t =
  [e411 t = (e412 t - Cx Motor_4 * inertance_f Jm_4 e421 p_0 t) ;
   f412 t = res_f Ra_4 e411 t ;
    momentum_der e421 p_0 t = (-- res_e Dm_4 f422 t - Cx Geer_4 *
              (compliance_e K_2 f441 q_0 t + res_e D_4 f443 t) +
              Cx Motor_4 * res_f Ra_4 e411 t);
   f422 t = inertance_f Jm_4 e421 p_0 t;
    momentum_der e431 p_0 t = (compliance_e K_2 f441 q_0 t +
                        res_e D_4 f443 t);
    displacement_der f441 q_0 t = (-- inertance_f J_4 e431 p_0 t +
                       Cx Geer_4 * inertance_f Jm_4 e421 p_0 t);
     f443 t = (-- inertance_f J_4 e431 p_0 t + Cx Geer_4 *
                    inertance_f Jm_4 e421 p_0 t)]
```

Where the function INDEX_FINGER_FLEX_EXT_EQ is used to formalize equations of the index finger (Flex. & Ext.).

5.2 Formal Verification

In the verification process, we ensure that the equations obtained from our formalized BG models are same as the modeled equations. The INDEX_FINGER_FLEX_EXT_EQS is the formally verified function for modeled equations INDEX_FINGER_FLEX_EXT_EQS and function bg_main provides differential equations of the INDEX_FINGER_FLEX_EXT as follows:

Theorem 5 Implementation of Index Finger Verifies Equations

```
⊢ ∀e411 e412 f412 e413 f413 e421 f422 e423 f423 e424 f424 e431
   e432 f432 e433 f433 f441 e442 f442 f443 Ra_4 Motor_4 Jm_4 Dm_4
    Geer_4 J_4 K_2 D_4 p_0 q_0 t.
  bg_main (INDEX_FINGER_FLEX_EXT e411 e412 f412 e413 f413 e421 f422
   e423 f423 e424 f424 e431 e432 f432 e433 f433 f441 e442 f442 f443
    Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 K_2 D_4 p_0 q_0) t =
  INDEX_FINGER_FLEX_EXT_EQ e411 e412 f412 e413 f413 e421 f422 e423
    f423 e424 f424 e431 e432 f432 e433 f433 f441 e442 f442 f443
    Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 K_2 D_4 p_0 q_0 t
```

Following theorem provides the final form of equations for the index finger (Flex. & Ext.) under some assumptions. First line of assumptions ensures that the real-values constants present in the BG model of mechatronic hand are positive values.

Theorem 6 Implementation of Index finger (Flex. & Ext.) Provides Simplified Equations

```
⊢ ∀e411 e412 f412 e413 f413 e421 f422 e423 f423 e424 f424 e431
   e432 f432 e433 f433 f441 e442 f442 f443 Ra_4 Motor_4 Jm_4 Dm_4
    Geer_4 J_4 K_2 D_4 p_0 q_0 t.
  (&0 < Jm_4) ∧ (&0 < Ra_4) ∧ (&0 < J_4) ∧ (&0 < K_2) ∧
   (f443 t = (-- inertance_f J_4 e431 p_0 t +
      Cx Geer_4 * inertance_f Jm_4 e421 p_0 t)) ∧
    ( f422 t = inertance_f Jm_4 e421 p_0 t) ∧
      (e313 t = e412 t) ∧ (e313 t = src_e Se t)
  ⇒ bg_main (INDEX_FINGER_FLEX_EXT e411 e412 f412 e413 f413 e421
    f422 e423 f423 e424 f424 e431 e432 f432 e433 f433 f441 e442
   f442 f443 Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 K_2 D_4) t =
  [e411 t = (Cx Se - Cx Motor_4 * inertance_f Jm_4 e421 p_0 t);
   f412 t = res_f Ra_4 e411 t;
  momentum_der e421 p_0 t =
```

$$\texttt{Cx}\left(\left(\frac{\texttt{-- Dm\_4}}{\texttt{Jm\_4}}\right) - \left(\frac{\texttt{(Geer\_4) pow 2 * D\_4}}{\texttt{Jm\_4}}\right) - \left(\frac{\texttt{(Motor\_4) pow 2}}{\texttt{Jm\_4 * Ra\_4}}\right)\right) *$$
$$\texttt{momentum e421 p\_0 t + Cx}\left(\frac{\texttt{Geer\_4 * D\_4}}{\texttt{J\_4}}\right) \texttt{* momentum e431 p\_0 t +}$$
$$\texttt{Cx}\left(\frac{\texttt{-- Geer\_4}}{\texttt{K\_2}}\right) \texttt{* displacement f441 q\_0 t + Cx}\left(\frac{\texttt{Motor\_4 * S\_e}}{\texttt{Ra\_4}}\right)\texttt{;}$$

```
f422 t = inertance_f Jm_4 e421 p_0 t;
```

$$\texttt{momentum\_der e431 p\_0 t = Cx}\left(\frac{\texttt{Geer\_4 * D\_4}}{\texttt{Jm\_4}}\right) \texttt{* momentum e421 p\_0 t +}$$
$$\texttt{Cx}\left(\frac{\texttt{-- D\_4}}{\texttt{J\_4}}\right) \texttt{* momentum e431 p\_0 t + Cx}\left(\frac{\texttt{\&1}}{\texttt{K\_2}}\right) \texttt{*}$$

```
                     displacement f441 q_0 t);
```

```
displacement_der f441 q_0 t = Cx (Geer_4 / Jm_4) * momentum e421 p_0 t +
                            Cx (- &1 / J_4) * momentum e_431 p_0 t);
f443 t = (-- inertance_f J_4 e431 p_0 t +
          Cx Geer_4 * inertance_f Jm_4 e421 t)]
```

5.3 State-Space Representation

A state-space model accepts system matrix $\mathtt{A}:\mathbb{C}^{N\times N}$, input matrix $\mathtt{B}:\mathbb{C}^{P\times N}$, state vector $\mathtt{x}:\mathbb{C}^{N}$, state vector with derivative entries $\mathtt{x\_der}:\mathbb{C}^{N}$ and input vector $\mathtt{u}:\mathbb{C}^{P}$. The state-space representation of index finger movements (Flex. & Ext) in HOL Light is as follows:

Theorem 7 State-Space Model

$$\vdash \forall \mathtt{e}_{421}\ \mathtt{e}_{431}\ \mathtt{f}_{441}\ \mathtt{Ra}_4\ \mathtt{Motor}_4\ \mathtt{Jm}_4\ \mathtt{Dm}_4\ \mathtt{Geer}_4\ \mathtt{J}_4\ \mathtt{D}_4\ \mathtt{K}_2\ \mathtt{S}_e\ \mathtt{t}.$$

$$\mathtt{ss\_model} \begin{bmatrix} \mathtt{A} & \mathtt{B} & \mathtt{C} \\ \mathtt{D} & \mathtt{E} & \mathtt{F} \\ \mathtt{G} & \mathtt{H} & \mathtt{I} \end{bmatrix} \begin{bmatrix} \mathtt{Cx}\left(\frac{\mathtt{Motor}_4}{\mathtt{Ra}_4}\right) \\ \mathtt{Cx\ (\&0)} \\ \mathtt{Cx\ (\&0)} \end{bmatrix} \begin{bmatrix} \mathtt{p}_{421}(\mathtt{t}) \\ \mathtt{p}_{431}(\mathtt{t}) \\ \mathtt{q}_{441}(\mathtt{t}) \end{bmatrix} \begin{bmatrix} \mathtt{p}'_{421}(\mathtt{t}) \\ \mathtt{p}'_{431}(\mathtt{t}) \\ \mathtt{q}'_{441}(\mathtt{t}) \end{bmatrix} \begin{bmatrix} \mathtt{S}_e \end{bmatrix} =$$

$$\begin{bmatrix} \mathtt{p}'_{421}(\mathtt{t}) \\ \mathtt{p}'_{431}(\mathtt{t}) \\ \mathtt{q}'_{441}(\mathtt{t}) \end{bmatrix} = \begin{bmatrix} \mathtt{A} & \mathtt{B} & \mathtt{C} \\ \mathtt{D} & \mathtt{E} & \mathtt{F} \\ \mathtt{G} & \mathtt{H} & \mathtt{I} \end{bmatrix} ** \begin{bmatrix} \mathtt{p}_{421}(\mathtt{t}) \\ \mathtt{p}_{431}(\mathtt{t}) \\ \mathtt{q}_{441}(\mathtt{t}) \end{bmatrix} + \begin{bmatrix} \mathtt{Cx}\left(\frac{\mathtt{Motor}_4}{\mathtt{Ra}_4}\right) \\ \mathtt{Cx\ (\&0)} \\ \mathtt{Cx\ (\&0)} \end{bmatrix} ** \begin{bmatrix} \mathtt{S}_e \end{bmatrix}$$

Where

$\mathtt{p}'_{421}(\mathtt{t}) = \mathtt{momentum\_der}\ \mathtt{e}_{421}\ \mathtt{p}_0\ \mathtt{t}$

$\mathtt{p}'_{431}(\mathtt{t}) = \mathtt{momentum\_der}\ \mathtt{e}_{421}\ \mathtt{p}_0\ \mathtt{t}$

$\mathtt{q}'_{441}(\mathtt{t}) = \mathtt{displacement\_der}\ \mathtt{f}_{441}\ \mathtt{q}_0\ \mathtt{t}$

$\mathtt{p}_{421}(\mathtt{t}) = \mathtt{momentum}\ \mathtt{e}_{421}\ \mathtt{p}_0\ \mathtt{t}$

$\mathtt{p}_{431}(\mathtt{t}) = \mathtt{momentum}\ \mathtt{e}_{421}\ \mathtt{p}_0\ \mathtt{t}$

$\mathtt{q}_{441}(\mathtt{t}) = \mathtt{displacement}\ \mathtt{f}_{441}\ \mathtt{q}_0\ \mathtt{t}$

$$A = \mathtt{Cx}\ \left(\left(\frac{-\mathtt{Dm}_4}{\mathtt{Jm}_4}\right) - \left(\frac{(\mathtt{Geer}_4)\mathtt{pow2} * \mathtt{D}_4}{\mathtt{Jm}_4}\right) - \left(\frac{(\mathtt{Motor}_4)\mathtt{pow2}}{\mathtt{Jm}_4 * \mathtt{Ra}_4}\right)\right)$$
$$B = \mathtt{Cx}\ \left(\frac{\mathtt{Geer}_4 * \mathtt{D}_4}{\mathtt{J}_4}\right),\ C = \mathtt{Cx}\ \left(\frac{-\mathtt{Geer}_4}{\mathtt{K}_2}\right),\ D = \mathtt{Cx}\ \left(\frac{\mathtt{Geer}_4 * \mathtt{D}_4}{\mathtt{Jm}_4}\right)$$
$$E = \mathtt{Cx}\ \left(\frac{-\mathtt{D}_4}{\mathtt{J}_4}\right),\ F = \mathtt{Cx}\ \left(\frac{\&1}{\mathtt{K}_2}\right),\ G = \mathtt{Cx}\ \left(\frac{\mathtt{Geer}_4}{\mathtt{Jm}_4}\right)$$
$$H = \mathtt{Cx}\ \left(\frac{-\ \&1}{\mathtt{J}_4}\right),\ I = \mathtt{Cx}\ (\&0)$$

5.4 Stability Analysis in MATLAB

For the utilization of stability property, we modeled our formalized theorems of matrices in MATLAB in the form of an algorithm to verify the status (stable, marginally stable or unstable) of the given system and presented results in graphical form as shown in Figure 5.2. All of the roots of a polynomial (eigenvalues) lie in the left half of the complex plane, which means system of index finger (Flex. & Ext.) is stable. Our work of formal stability analysis is of the automatic nature, which makes it quite useful even for a beginner of formal methods.

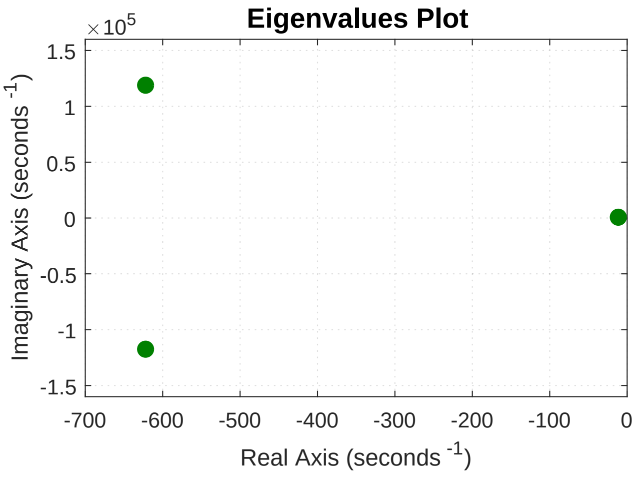


Figure 5.2: Stable Index Finger (Flex. & Ext.)

The distinguishing features of our proposed formalization as compared to the traditional analysis techniques are that, all of the parameters of BG are clearly defined along with a data type, all of the variables and functions are of a generic nature, i.e., universally quantified, soundness is assured in functions and theorems as every new theorem can be verified by applying basic axioms and inference rules or any other previously verified theorems/inference rules, and the algoritmic approach for extracting equations. In traditional analysis techniques, like computer-based simulations, we model each case of BG individually and there is a chance of error due to the misinterpretation of the parameters. The verification and the formal analysis of a BG representation ensure the accurate results and stability of a dynamic system (Anthropomorphic Robotic Hand).

Chapter 6

Conclusions

6.1 Summary

In this thesis, we presented the formal analysis of Bond graph representations of engineering and physical systems by utilizing higher-order-logic theorem proving. In particular, this thesis mainly presents two important contributions. Firstly, it provides the generic formalization of Bond graph, which includes formal definitions of bonds, components, junctions, paths, branches, loops and the state-space representation. It also presents the conversion of a Bond graph model to its corresponding state-space model and the verification of stability and its various properties. Secondly, it presents the results of the stability analysis in a graphical form. For this purpose, we encoded our formally verified stability theorems in MATLAB. We illustrated the practical effectiveness of our proposed framework by utilizing the above mentioned steps to present the formal analysis of an Anthropomorphic Mechatronic Hand.
Our proposed framework, in particular the formalization of Bond graph involves significant user interaction. Thus, in order to facilitate the HOL Light users of our framework, we developed some important tactics which are useful for conducting the proofs of the physical systems using our formalized library of Bond graph.

6.2 Future Work

The formalization of Bond graph provided in this thesis can be further extended in order to strengthen the analysis of the current framework. The presence of algebraic loops, namely, implicit algebraic equations in Bond graph is of great importance, thus extending our formalization to cope up with algebraic loops will be a good contribution. Moreover, our proposed formalization deals only with Bond graphs that results into linear equations, thus, the formalization of Bond graph can be extended to non-linear case.

References

[1] Simulator 20-sim. `https://www.20sim.com/features/simulator/`, 08 2019. (Online; Acessed 28/7/2020 20:2).

[2] Asad Ahmed, Osman Hasan, and Falah Awwad. Formal stability analysis of control systems. In *International Workshop on Formal Techniques for Safety-Critical Systems*, pages 3–17. Springer, 2018.

[3] Donald J Ballance, Geraint P Bevan, Peter J Gawthrop, and Dominic J Diston. Model transformation tools (mtt): the open source bond graph project. 2005.

[4] Wolfgang Borutzky. Derivation of mathematical models from bond graphs. *Bond Graph Methodology: Development and Analysis of Multidisciplinary Dynamic System Models*, pages 89–128, 2010.

[5] Wolfgang Borutzky. *Bond graph modelling of engineering systems*, volume 103. Springer, 2011.

[6] Jan F Broenink. Introduction to physical systems modelling with bond graphs. *SiE whitebook on simulation methodologies*, 31:2, 1999.

[7] M Delgado and C Brie. Desis-a modeling and simulation package based on bond graphs. *IFAC Proceedings Volumes*, 24(4):215–220, 1991.

[8] Vanessa Díaz-Zuccarini and César Pichardo-Almarza. On the formalization of multi-scale and multi-science processes for integrative biology. *Interface focus*, 1(3):426–437, 2011.

[9] Immanuel Gaiser, Stefan Schulz, Artem Kargov, Heinrich Klosek, Alexander Bierbaum, Christian Pylatiuk, Reinhold Oberle, Tino Werner, Tamim Asfour,

Georg Bretthauer, et al. A new anthropomorphic robotic hand. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 418–422. IEEE, 2008.

[10] Peter J Gawthrop and Geraint P Bevan. Bond-graph modeling. *IEEE Control Systems Magazine*, 27(2):24–45, 2007.

[11] Jose Granda and Raymond Montgomery. Automated modeling and simulation using the bond graph method for the aerospace industry. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 5527, 2003.

[12] Jose J Granda and Jim Reus. New developments in bond graph modeling software tools: the computer aided modeling program camp-g and matlab. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 2, pages 1542–1547. IEEE, 1997.

[13] Michael K Hales and Ronald C Rosenberg. Enport model builder: An improved tool for multiport modeling of mechatronic systems. *SIMULATION SERIES*, 33(1):152–157, 2001.

[14] J. Harrison. HOL Light: A Tutorial Introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

[15] Mohammadali Honarpardaz, Mehdi Tarkian, Johan Ölvander, and Xiaolong Feng. Finger design automation for industrial robot grippers: A review. *Robotics and Autonomous Systems*, 87:104–119, 2017.

[16] Violina Iordanova, Hassane Abouaissa, and Daniel Jolly. Bond-graphs traffic flow modelling and feedback control. *IFAC Proceedings Volumes*, 39(3):345–350, 2006.

[17] BJ Joseph and HR Martens. The method of relaxed causality in the bond graph analysis of nonlinear systems. 1974.

[18] Javier Kypuros. *System Dynamics and Control with Bond Graph Modeling.* CRC Press, 2013.

[19] Norman S Nise. *CONTROL SYSTEMS ENGINEERING, (With CD).* John Wiley & Sons, 2007.

[20] Henry M Paynter. *Analysis and Design of Engineering Systems.* MIT press, 1961.

[21] Changju Rhee, Woojin Chung, Munsang Kim, Youngbo Shim, and Hyungjin Lee. Door opening control using the multi-fingered robotic hand for the indoor service robot. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 4011–4016. IEEE, 2004.

[22] Henry Ricardo. *A modern introduction to linear algebra.* CRC Press, 2009.

[23] R. C. Rosenberg, W. Feurzeig, and P. Wexelblat. Bond graphs and enport in elementary physics instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):170–174, 1970.

[24] Muhammad Tallal Saeed, Sardor Khaydarov, Biniam Legesse Ashagre, and MS Zafar. Comprehensive bond graph modeling and optimal control of an anthropomorphic mechatronic prosthetic hand. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 2006–2011. IEEE, 2019.

[25] Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.

[26] Jean U Thoma. *Introduction to Bond Graphs and their Applications.* Elsevier, 2016.

[27] Lefan Wang, Turgut Meydan, and Paul Ieuan Williams. A two-axis goniometric sensor for tracking finger motion. *Sensors*, 17(4):770, 2017.

[28] Lin-an Wang, Qiang Li, and Xiao-juan Liang. Modeling and dynamic simulation of electric power steering system of automobile using bond graph technique. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, pages 744–747. IEEE, 2010.

Appendices

Appendix A

Proof Tactics

Tactic	**Description**
BOND_DIRECTION_TAC	It simplifies the function that calculates power direction of a bond
BOND_MODULUS_TAC	It simplifies the modulus of two-port component function
BOND_EFFORT_TAC	It simplifies the function that calculates effort variable of a bond
BOND_FLOW_TAC	It simplifies function which calculates flow variable of a bond
BACKWRD_PATH_TAC	It mainly simplifies the functions of backward path which specifies path and strong bond
FWRD_PATH_TAC	It simplifies the function of forward path which specifies path direction and strong bond
PATHS_SELECTION_TAC	Simplifies the functions which are responsible for selecting a path
BRANCH_TAC	Simplifies the functions that check the presence of a branch on a bond of a junction
CAUSAL_LOOP_TAC	It simplifies the functions of causal loop that are resposible for detecting the presence of a loop and finding both connecting bonds of a loop
CASE_SELECTION_TAC	It simplifies the functions which are responsible for choosing a case