ISA Transactions 128 (2022) 453–469

Contents lists available at ScienceDirect

ISA Transactions

journal homepage: www.elsevier.com/locate/isatrans

Research article

Formalization of bond graph using higher-order-logic theorem proving

Ujala Qasim [a], Adnan Rashid [b,*], Osman Hasan [b]

[a] Research Center for Modeling and Simulations (RCMS), National University of Sciences and Technology (NUST), Islamabad, Pakistan
[b] School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan

ARTICLE INFO

Article history:
Received 7 November 2020
Received in revised form 11 October 2021
Accepted 23 November 2021
Available online 11 December 2021

Keywords:
Bond graphs
State–space models
Theorem proving
Higher-order logic
HOL Light

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 domains, such as, electrical, mechanical, medical, thermal and fluid mechanics. Traditionally, these dynamics are analyzed using paper-and-pencil 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 enormous computational requirements. 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, higher-order-logic theorem proving, can overcome the shortcomings of these traditional methods and provide an accurate analysis of these systems. It has been widely used for analyzing the dynamics of engineering and physical systems. In this paper, we propose to use higher-order-logic theorem proving for performing the bond graph based analysis of the physical systems. In particular, we provide formalization of bond graph, which mainly includes functions that allow conversion of a bond graph to its corresponding mathematical model (state–space model) and the 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 help non-experts in HOL, we encode our formally verified stability theorems in MATLAB to perform the stability analysis of an anthropomorphic prosthetic mechatronic hand.

1. Introduction

Bond Graph (BG) is a linear, labeled, directed and domain-independent graphical approach for modeling 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 BG was introduced by H.M. Paynter [1], of Massachusetts Institute of Technology (MIT) in 1961, to describe the dynamics of different systems belonging to multidisciplinary domains and exhibiting analogous dynamical behavior. Signal-flow graphs and the block diagram representations are the other graphical approaches used for capturing the dynamics of the physical systems. However, BG models are generally preferred due to their ability to communicate seamlessly between different components belonging to multidisciplinary domains based on the bi-directional flow of information and thus provide a deep insight to the computational structure of the physical systems. Due to these distinguishing features, BG have been widely used in automobiles [2], biological systems [3], aerospace [4,5] and transportation systems [6]. The first step in BG based dynamical analysis of a system is to apply the 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 transformed into state–space representations by applying various properties of vectors and matrices. The final step is based on analyzing various properties, such as, stability, of the state–space model of the underlying system.

Traditionally, the BG based analysis is performed using paper-and-pencil proof method. However, the analysis is prone to error due to the highly involved human manipulation for analyzing the complex physical systems and thus could not ensure absolute accuracy of the analysis. Similarly, computer based symbolic and numerical techniques have been widely used for analyzing BG based models of the systems. Some of the widely used tools are N-PORT (ENPORT) [7,8], DEscription and SImulation Systems (DESIS) [9], Model Transformation Tools (MTT) [10], Computer Aided Modeling Program with Graphical Input (CAMP-G) [11] and Twente Simulator (20-sim) [12]. However, these numerical methods involve the approximation of the mathematical

* Corresponding author.
E-mail addresses: uqasim.mscse16@rcms.nust.edu.pk (U. Qasim), adnan.rashid@seecs.nust.edu.pk (A. Rashid), osman.hasan@seecs.nust.edu.pk (O. Hasan).

https://doi.org/10.1016/j.isatra.2021.11.042
0019-0578/

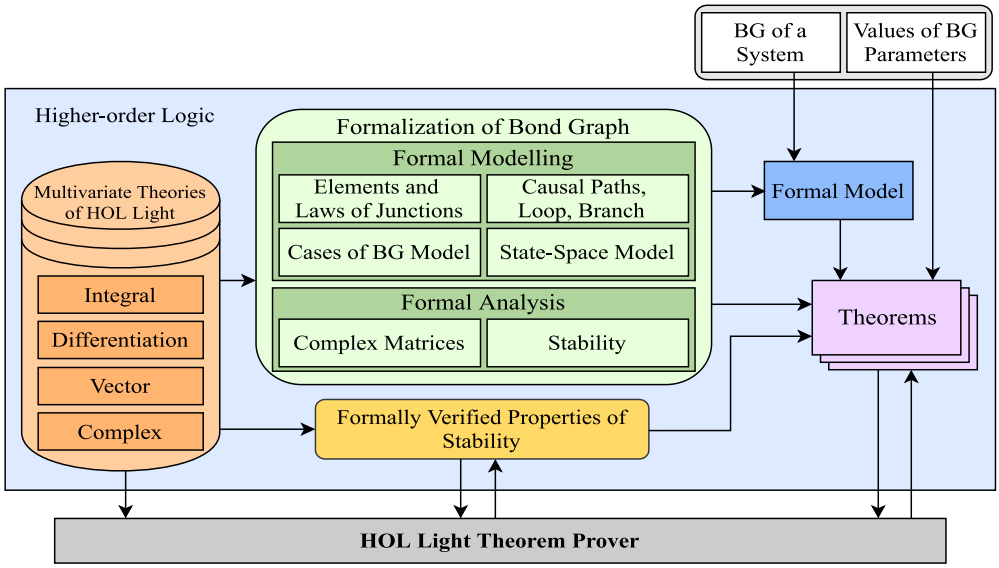


Fig. 1. Proposed framework.

expressions and results due to the finite precision of computer arithmetic. Moreover, they involve 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 BG 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 method for performing an accurate analysis of the engineering and physical systems. In this paper, we propose a higher-order-logic based framework, as depicted in Fig. 1, for the BG based analysis of the physical systems. It mainly involves the formalization of the BG models and the formal verification of their various properties using multivariate calculus theories of HOL Light theorem prover [13]. To illustrate the practical effectiveness of our proposed framework, we formally analyze the dynamics of an anthropomorphic mechatronic prosthetic hand [14] based on the formalization of BG.

Our proposed framework accepts a BG representation of a system and its corresponding parameters, such as, the values of its different components from a user. The first step is the development of the mathematical model (state–space representation), which involves a conversion of the BG representation to a set of differential equations. It is based on the application of the laws of junctions and the 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 the complex-valued vectors and matrices, which are developed as a part of our 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 a practical illustration of our proposed formalization, we present a formal stability analysis of an anthropomorphic mechatronic prosthetic hand [14]. Moreover, to facilitate a non-expert HOL user, we encode our formally verified stability theorems in MATLAB to perform stability analysis of an anthropomorphic prosthetic mechatronic hand.

The rest of the paper is organized as follows: We provide related work in Section 2 about the formal verification of engineering systems involving complex dynamics. Section 3 provides preliminaries that include a brief introduction of the HOL Light theorem prover, multivariable calculus theories of HOL Light and BGs. Section 4 describes the proposed algorithm for the formal analysis of BGs and some set of assumptions/rules for the formal modeling of BGs. Section 5 presents the formalization of the fundamental components of BG representations of different systems. We provide the definition of stability, formally verified general stability theorem and present various properties of stability analysis in Section 6 of the paper. Section 7 provides the stability analysis of an anthropomorphic mechatronic prosthetic hand. Finally, Section 8 concludes the paper.

2. Related work

Traditionally, the BG based analysis of the engineering and physical systems is performed using paper-and-pencil proofs, and computer based symbolic and numerical techniques [2,4–6]. However, these methods suffer from their inherent limitations, such as, human error process, approximation of the mathematical expressions and limited computational resources. Formal methods, in particular, higher-order-logic theorem proving has been widely used for performing an accurate analysis of the engineering and physical and systems. BG based formal analysis using a theorem prover is mainly based on some foundational libraries, such as, vectors and matrices. These vectors and matrices libraries have been formalized in various higher-order-logic theorem provers, such as HOL4, HOL Light, Coq and Isabelle [15–21]. Among these formal libraries, the one available in HOL Light is quite comprehensive, especially in the reasoning support for vectors and matrices, and is thus suitable for the proposed formalization of BG.

HOL Light has been extensively used for formally analyzing many engineering and physical systems. Rashid et al. [22–25] formalized the Laplace and the Fourier transforms using HOL Light, which are the major mathematical techniques used for analyzing the dynamics of the continuous-time systems. Moreover, the authors used these formalizations for formally analyzing an automobile suspension system, a MEMs accelerometer and a $4 - \pi$ soft-error crosstalk model [26,27]. Similarly, Rashid

et al. [28,29] formally verified a cell injection system up to 4-DOF using HOL Light. The authors verified various coordinate systems and their interrelationship, which are vital for capturing the position and relative movement of a robotic cell injection system. Moreover, they also formally verified the solution of differential equations modeling the continuous dynamics of the system. Beillahi et al. [30] formalized the signal-flow-graph theory using HOL Light to perform the formal analysis of engineering systems namely, the PANDA Vernier resonator and the z-source impedance network. Similarly, Siddique et al. [31] formalized geometrical optics using HOL Light for analyzing optical and laser systems. The authors formally verified frequently used optical components like thin lens, thick lens and plane parallel plate and performed the stability analysis of Fabry Pérot resonator and Z-shaped resonator. Recently, Abed et al. performed the dynamical analysis of Unmanned Aerial Vehicles (UAVs) [32] and synthetic biological circuits [33,34]. Similarly, Ahmed et al. [35] performed a stability analysis of power converters using HOL Light. However, none of these contributions provide the BG based analysis of systems, which is the scope of the current paper. A more detailed account of the formal analysis of the engineering and physical systems can be found at [36].

3. Preliminaries

In this section, we provide an introduction to the HOL Light theorem prover, its multivariate calculus theories and BGs which are necessary for understanding rest of the paper.

3.1. 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 family of HOL theorem provers and is used for developing proofs in higher-order logic. HOL Light is built using the functional programming language objective CAML (OCaml). It has been extensively used for the formal verification of both hardware [28,37,38] and software systems [39] along with the formalization of mathematics.

A theorem in HOL Light is verified by applying the basic axioms and primitive inference rules or any other previously verified theorems/inference rules. Generally, a HOL Light theorem is of the form A1, A2, . . ., An $\Rightarrow$ C, where A1, A2, . . ., An model the set of assumptions and C models the conclusion and is the main goal of the HOL Light theorem. The proof of a theorem involves the concept of a tactic (an ML function), which divides the main goal into subgoals. These tactics are repeatedly used to reduce or simplify the main goal (required theorem) into intermediate subgoals until they match with the assumptions of the HOL Light theorem, concluding the proof of the required theorem.

HOL Light provides an extensive support of the formal libraries for the multivariate calculus, such as, differentiation, integration, transcendental and topology, which have been extensively used in the proposed formalization of BGs. The availability of these mathematical theories is one of the main motivations for opting HOL Light for our proposed formalization of BGs. Table 1 provides some of the frequently used HOL Light symbols and functions in our formalization.

3.2. 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 numbers ($\mathbb{R}$). All 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 as a 1-dimensional vector $\mathbb{R}^1$ or a number on a real line $\mathbb{R}$. In multivariate calculus theories of HOL Light, all theorems have been formally verified for functions with an arbitrary 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.

Table 1
HOL Light symbols and functions.

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 $x : t$
λx.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
lift x	$\mathbb{R} \rightarrow \mathbb{R}^1$	Map Reals to 1-Dimensional Vectors
drop x	$\mathbb{R}^1 \rightarrow \mathbb{R}$	Map 1-Dimensional Vectors 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)}	$\{x\|P(x)\}$	Set of all x such that $P(x)$

Definition 3.1. $\vdash_{def}$ ∀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 & typecasts a natural number to its corresponding real number.

Definition 3.2. $\vdash_{def}$ ∀z. Re z = z$1
$\vdash_{def}$ ∀x. lift x = (lambda i. x)

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

Definition 3.3. $\vdash_{def}$ ∀f i. integral i f = (@y.(f has_integral y) i)
$\vdash_{def}$ ∀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 $\mathbb{B}$ represents 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 the 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. Here, the Hilbert choice operator @:$(\alpha \rightarrow \text{bool}) \rightarrow \alpha$ returns values of the integral and differential, if they exist.

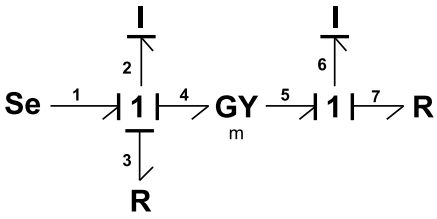


Fig. 2. Bond graph representation.

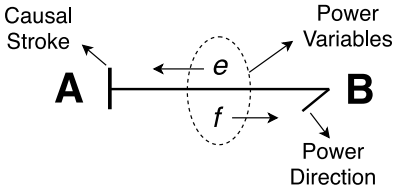


Fig. 3. Bond representation.

3.3. Bond graphs

BG [40] is a directed graph composed of bonds and components that are connected together, as shown in Fig. 2. The power bond is the most powerful component of a BG bridging any two components of BG and providing an exchanged power between them.

A power bond is represented by a half arrow whose head indicates the direction of a positive power flow, as shown in Fig. 3. BG models are based on three types of analogies namely, signal, component and the connection analogies [5]. An analogy is a mapping of the dynamical phenomena/properties from one physical system to another. In BG, systems from different domains result into analogous equations, utilizing the concept of analogies.

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 aspects of systems from a wide range of domains/areas. For example, in the electrical domain, voltage (e), current (f), flux linkage (p) and charge (q) are represented by these signals [41], respectively. Since power is a product of effort and flow signals, a power bond is composed of effort and flow signals (variables), as shown in Fig. 3.

The other two signals, i.e., the integrated effort and flow signals belong to a class of energy variables. The integrated effort also known as generalized momentum, is mathematically expressed as a time integral over the effort signal.

$$p(t) = \int^{t} e(t)dt = p_0 + \int_0^{t} e(t)dt \tag{1}$$

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.

$$q(t) = \int^{t} f(t)dt = q_0 + \int_0^{t} f(t)dt \tag{2}$$

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.

$$e(t) = \dot{p}(t) \tag{3}$$

$$f(t) = \dot{q}(t) \tag{4}$$

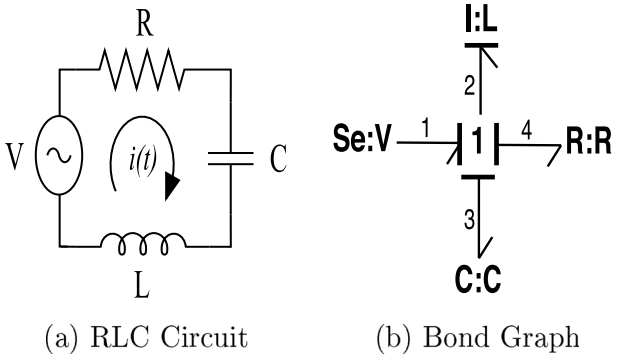


Fig. 4. RLC circuit and its corresponding bond graph representation.

The simulation of a BG is primarily based on the order of a computation of the effort and flow variables. This order is represented by a perpendicular bar (causal stroke) added to an end of a bond (head or tail of the arrow, as shown in Fig. 3), thus indicating the corresponding variables acting as an input (cause) and an 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 BG model. In Fig. 3, effort e is the input to A and the flow f is the output from A. Similarly, f acts as an input to B and e acts as an output.

A user needs to have some prior knowledge of causality, i.e., how to assign it manually, to construct the BG models of complex physical systems. Many approaches exist in the literature [40,42] for a systematic assignment of the casuality. However, Sequential Causality Assignment Procedure (SCAP) is generally preferred [40,43] due to the systematic process and provides state–space representation of a system. Causality analysis provides information about the various aspects of a system, i.e., inconsistency of design and an ill-posed model, the number of state–space variables and the presence of algebraic loops, which results in some complex Differential Algebraic Equations (DAEs), without deriving equations of a system.

There are nine basic elements/components of a BG representation, categorized into four groups according to their characteristics, as shown in Table 2. Component analogies are further categorized into three groups, namely supply, passive elements and reversible transformation. The supply group contains sources of effort and flow variables. The passive elements contain the storage elements I and C as well as the 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. The first three groups of Table 2 provide the component analogies and the last group represents the connection analogies.

Table 3 provides descriptions of various components of a BG representation, depicted in Figs. 4 and 6. Now, we illustrate a BG based dynamic analysis of a most commonly used RLC circuit [44], depicted in Fig. 4a. To perform the BG based analysis of the RLC circuit, first we formally model the given circuit using its BG representation as shown in Fig. 4b. Next, the laws of BGs, given in Table 2, are applied to obtain the corresponding set of equations. Finally, the BG components, given in Table 3, are used to derive the corresponding state–space model of the given RLC circuit. Generally, the BG representation capturing the dynamics of a system is based on transforming (mapping) system's components to their BG model counterparts and it varies according to the systems from various domains, such as, electrical, mechanical and medicine [45].

Table 2
Basic components of bond graph.

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

For the case of the given RLC circuit (Fig. 4a), the voltage V is modeled by the effort source S_e and the inductor L is modeled by the inductor I. Moreover, the capacitor (C) and the resistor (R) are mapped to the energy storing element C and passive element R, respectively, as shown in Fig. 4b [40].

In the considered RLC circuit, same current flows from all components due to their series configuration. However, every component exhibits different voltages. The voltage and current are mapped to the effort and flow variables, respectively. The presence of 1_*junction* shows that the value of flow variable (current) through all connected bonds is the same (by the Kirchhoff's Current Law or KCL [40]), and the summation of effort variables (voltages) with power direction of bonds is equal to zero (by the Kirchhoff's Voltage Law or KVL [44]) as shown in Table 2. Each bond is labeled by a number and a causality is assigned on every bond by following the SCAP approach. Thus, BG presented in Fig. 4b is known as a causal BG. 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 Fig. 4b. 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. The mathematical equations of various components are expressed as follows:

$$e_1(t) = V(t) \tag{5}$$

$$f_2(t) = \frac{1}{L} * (p_0 + \int_0^t e_2(t)\,d(t))$$

By using Eq. (1), the inductor's equation can be written as:

$$f_2(t) = \frac{p_2(t)}{L} \tag{6}$$

Similarly, an equation for the capacitor can be represented in terms of energy variable (q) by using Eq. (2).

$$e_3(t) = \frac{1}{C} * (q_0 + \int_0^t f_3(t)\,d(t))$$

$$e_3(t) = \frac{q_3(t)}{C} \tag{7}$$

The output of the dissipative component (R) depends algebraically on the input as:

$$e_4(t) = R\,.\,f_4(t) \tag{8}$$

Now, we apply both laws of 1_*junction*, i.e., KCL and KVL for the case of RLC circuit. Since there is only one bond, i.e., Bond 1, which has a positive power direction, as shown in Fig. 4b. Therefore, all of the effort variables except e_1 have a negative sign in the application of the summation law, i.e., KVL.

$$e_1 - e_2 - e_3 - e_4 = 0 \tag{9}$$

Similarly, since Bond 2 is a strong bond, as shown in Fig. 4b, so the equality law, i.e., KCL for 1_*junction*, is mathematically expressed as follows:

$$f_2 = f_1 = f_3 = f_4 \tag{10}$$

Table 3
Components of a bond graph representation.

Name	Description
Strong bond	A single bond that causes effort in the 0_*junction* and flow in the 1_*junction*
Passive element	A one port element that stores input power as potential energy (C-element), as kinetic energy (I-element) or transforms it into dissipative power (R-element)
Causal BG	A BG is called causally completed or causal if the causal stroke known as causality is added on one 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 or 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 switches the direction of efforts/flows on one of its side [43]. 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 connected with a common bond or 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

Next, to obtain the state equations, i.e., equations of the energy storing components, we use equations of the components, i.e., Eqs. (5), (7), (8) in Eq. (9).

$$V(t) - e_2(t) - \frac{1}{C} \,.\, q_3(t) - R \,.\, f_4(t) = 0$$

The goal is to obtain the state equation for inductor (Bond 2) but 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 Fig. 4b, 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 the above equation using Eq. (10) as follows:

$$V(t) - e_2(t) - \frac{1}{C} \,.\, q_3(t) - R \,.\, f_2(t) = 0$$

By using Eqs. (3) and (6), the above equation can be rewritten as:

$$\dot{p}_2(t) = e_2(t) = -\frac{R}{L} \,.\, p_2(t) - \frac{1}{C} \,.\, q_3(t) + V(t) \tag{11}$$

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

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

By using Eqs. (4) and (6) in the above equation, we obtain the final form of the state equation for component C (Bond 3) as follows:

$$\dot{q}_3(t) = f_3(t) = \frac{1}{L} \,.\, p_2(t) \tag{12}$$

Finally, by applying various properties of vectors and matrices, Eqs. (11) and (12) can be transformed to their corresponding state–space models as follows:

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

$$\begin{bmatrix} \dot{p}_2(t) \\ \dot{q}_3(t) \end{bmatrix} = \begin{bmatrix} -\frac{R}{L} & -\frac{1}{C} \\ \frac{1}{L} & 0 \end{bmatrix} \begin{bmatrix} p_2(t) \\ q_3(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} V(t) \end{bmatrix} \tag{13}$$

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

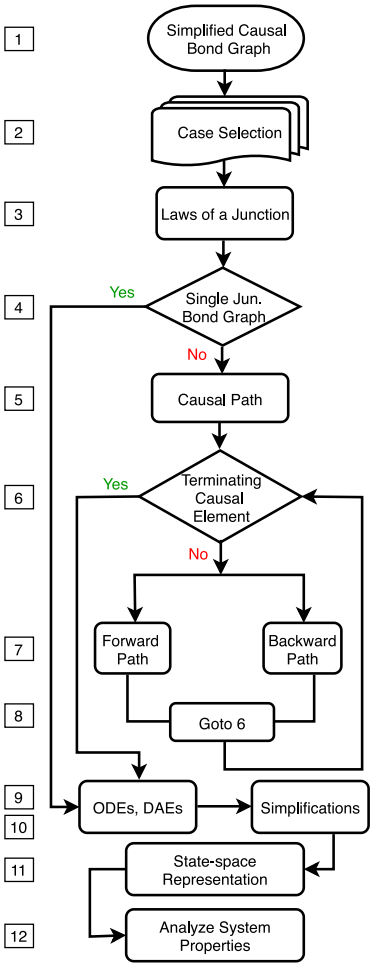


Fig. 5. Algorithm for bond graph based analysis of systems.

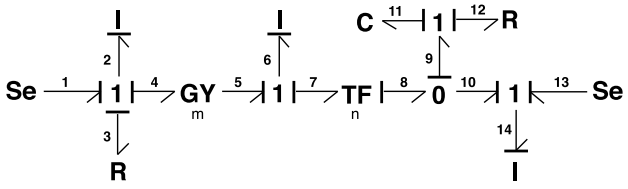
4. Proposed methodology for the bond graph based formal analysis

This section provides the proposed methodology, depicted in Fig. 5 for performing the BG based formal analysis of the dynamics of systems.

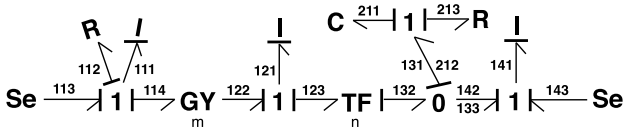
Step 1: Take a BG representation as an input from the user. The input BG should be:

(a) Causally completed

(b) Simplified after applying the simplification rules if necessary



(a) Bond Graph Representation



(b) Relabelling of Bond Graph Using Rules

Fig. 6. Representations of a bond graph.

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 within the same junction, 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: Repeat the above process by going 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.

Now, we present the rules that are followed, to obtain a relabeled BG representation, during the formalization of the BGs in HOL Light. Fig. 6 provides two different representations of an arbitrary BG, i.e., the given BG representation and the relabeled BG representation after applying the rules/assumptions to facilitate the corresponding formalization of the BGs. The rules for the formalization of the BG 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 an anti-clockwise direction as shown in Fig. 6b.

2. A bond $\mathbf{B}_{pqr}$ is labeled 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 Fig. 6a is relabeled as Bond 113, as shown in Fig. 6b. Bond 113 is the third bond of the first junction, which is an element of the first (parent) branch. Similarly, Bond 11 of Fig. 6a is relabeled as Bond 211 representing the first bond of the first junction that lies in the second (child) branch, as shown in Fig. 6b.

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 Fig. 6b, 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 the 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.

6. The modulus of a two-port component is represented in the form of a list containing the modulus of effort and flow variables. A two-port component (GY, TF) consists of two equations as described in Table 2. For example, in Fig. 6a, the list of the transformer component for the Bond 7 and 8 is in the form of $\left[\frac{1}{n}, n\right]$ and $\left[n, \frac{1}{n}\right]$, 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 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 with 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.

5. Formalization of bond graph representation

This section provides formal definitions of the building blocks of BG that can be used to formalize a wide range of systems in higher-order logic. We have developed our formalization of BG using the foundational definitions presented in Section 3.2.

5.1. Formal model of a BG representation

We model a bond as a 6-tuple using the type abbreviation in HOL Light as follows:

Definition 5.1.
```
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 of values of the complex data type as described in Section 4, respectively. The last element of the 6-tuple is itself a pair of a 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.

5.2. Formalization of a single bond

As discussed in Section 3.3, *e* and *f* are power variables while momentum and displacement represent energy variables. The formalized functions of these energy variables in HOL Light are as follows:

Definition 5.2. ⊢ ∀e f. power e f = (λt. e t ∗ f t)
⊢ ∀e. momentum e p_0 = (λt. p_0 + integral (interval [lift(&0), t]) e)
⊢ ∀f. displacement f q_0 = (λt. q_0 + integral (interval [lift(&0), t]) f)
⊢ ∀e. momentum_der e p_0 =
(λt. vector_derivative (momentum e p_0) (at t))
⊢ ∀f. displacement_der f q_0 =
(λt. vector_derivative (displacement f q_0) (at t))

Now, we formalize the causality of a bond in HOL Light as the following recursive functions.

Definition 5.3. ⊢ ∀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 causality$_f$ 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 4. Similarly, the function causality$_t$ returns true (T) if the direction of a bond k is towards a junction i.

A power bond carries information of causality as well as its power direction as shown in Fig. 3. Its formalized function is as follows:

Definition 5.4. ⊢ ∀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 junctions 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 from the 6-tuple capturing a BG representation. Moreover, the function bond_direction_cond assigns the complex numbers 1 and −1 to the Boolean values T and F, respectively.

5.3. Formalization of basic components of a single bond and its associated laws

In BGs, sources are called active elements with one variable equal to zero. src$_e$ represents supply effort to the system and src$_f$ represents supply flow to the system and are defined as follows:

Definition 5.5. ⊢ ∀e. src$_e$ e = (λt. Cx e)
⊢ ∀f. src$_f$ f = (λt. Cx f)

These are the passive elements of BG as described in Table 2. Both integral and differential causal elements are formalized. compliance$_e$ represents integrally causal C-element and compliance$_f$ has differential causality.

Definition 5.6. ⊢ ∀R f. res$_e$ R f = (λt. Cx R ∗ f t)
⊢ ∀R e. res$_f$ R e = (λt. Cx $\left(\frac{\&1}{R}\right)$ ∗ e t)
⊢ ∀C f. compliance$_e$ C f q_0 =
(λt. Cx $\left(\frac{\&1}{C}\right)$ ∗ displacement f q_0 t)
⊢ ∀C e. compliance$_f$ C e =
(λt. Cx C ∗ vector_derivative e (at t))
⊢ ∀L f. inductor$_e$ L f = (λt. Cx L ∗ vector_derivative f (at t))
⊢ ∀L e. inductor$_f$ L e p_0 = (λt. Cx $(\frac{\&1}{L})$ ∗ momentum e p_0 t)

Next, we provide the equality laws for the junctions 0 and 1 (presented as Eq. (10) for RLC circuit example in Section 3.3) of a BG representation in HOL Light as follows:

Definition 5.7. ⊢ ∀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 represent 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*), two different bonds k1 and k2 of the junction i, and models law of equality as an equation of effort. It uses the function bond_effort to capture the effort (e) variable. Similarly, the function jun_1_f models 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 BG representation in HOL Light as follows:

Definition 5.8. ⊢ ∀j i t. jun_1_e j i t =
(vsum 0. . .bonds_length j i - 3) (λk. bond_effort_wd j i k t)
⊢ ∀j i t. jun_0_f j i t =
(vsum 0. . .bonds_length 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 represents summation law of 0_*junction* by skipping the last two bonds. Moreover, any two junctions are connected by a common bond, For example, junctions 0 and 1 are connected by Bond 10 as shown in Fig. 6a by the laws of summation and equality. Their flow variables for both junctions are mathematically expressed as:

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

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

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

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

Definition 5.9. ⊢ ∀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))

A two-port element consists of two equations with different modulus for different variables (effort and flow), as shown in Table 2. The function modulus_cond verifies the causality of a

bond towards a junction and provides the 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 the effort variable is returned.

5.4. Formalization of the components of a BG

We formalize the strong bond in HOL Light as follows:

Definition 5.10. ⊢ ∀j i. strong_bond_f j i = causality$_f$ j i (bonds_length j i - 1)
⊢ ∀j i. strong_bond_t j i = causality$_t$ j i (bonds_length j i - 1)

The functions strong_bond_f and strong_bond_t use the functions of Definition 5.3 namely causality$_f$ and causality$_t$ that check all bonds of a junction to detect the strong bonds with casualty towards a junction and in the opposite direction of a junction, respectively. Here, the function bonds_length provides the total number of bonds of a single junction.

Next, we model the two skipped bonds of the last junction as follows:

Definition 5.11. ⊢ ∀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 5.8, we skipped the 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 we need to 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 5.12. ⊢ ∀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 the second last and last bonds of the junction i, following the backward and forward paths.

Definition 5.13. ⊢ ∀j i t. (backwrd_path j 0 t = Cx(&0)) ∧
(backwrd_path j (SUC i) t =
if P1 then P2 else if P3 then P4 else Cx (&0))

To find a causal path, we need to know the starting and the upcoming junction of the path, type of the common bond between two junctions and a strong bond. The function backward_path is made up of four different parts whose explanation is provided in Table 4. The first part, i.e., P1 of the function backward_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 keeps following the 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 in just like 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. The detailed definition can be found in our HOL Light proof script [46].

Table 4
Formalized parts of Definition 5.13.

Part	Formalized form
P1	(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) = F) ∧ (snd_last_bond_type j (SUC i) = 6) ∧ (type_of_jun j i) = F)
P2	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 last_bond_modulus_cond j i ∗ bond_effort j i (strong_bond_t j i) t
P3	(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) = F) ∧ (snd_last_bond_type j (SUC i) = 6) ∧ (type_of_jun j i) = T)
P4	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 last_bond_modulus_cond j i ∗ bond_flow j i (strong_bond_f j i) t)

Definition 5.14. ⊢ ∀j i t. (fwrd_path (j:jun_list) 0 t = Cx(&0)) ∧
(fwrd_path j (SUC i) t =
if P1 then P2 else if P3 then P4 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. The detailed formalized form of the parts P1, P2, P3 and P4 is given in Table 5.

Table 5
Formalized parts of Definition 5.14.

Part	Formalized form
P1	(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) = F) ∧ (last_bond_type (rev j) (SUC i) = 6) ∧ (type_of_jun (rev j) i = F)
P2	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 snd_last_bond_modulus_cond (rev j) i ∗ bond_effort (rev j) i (strong_bond_t (rev j) i) t
P3	(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) = F) ∧ (last_bond_type (rev j) (SUC i) = 6) ∧ (type_of_jun (rev j) i) = T)
P4	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 snd_last_bond_modulus_cond (rev j) i ∗ bond_flow (rev j) i (strong_bond_f (rev j) i) t

Definition 5.15. ⊢ ∀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, the function final_fwrd_path reverses 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 5.1, which remains unchanged in the reversion process. 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 unique number. 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 a BG representation with only one junction as shown in Fig. 4. 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:

Definition 5.16. ⊢ ∀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 check 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, the functions 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 5.17. ⊢ ∀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 Fig. 6a, to obtain the summation equation (for flow variable) of Bond 9, we follow both forward and backward paths by eliminating the last two bonds of 0_*junction*. It is important to note that the power direction of the skipped bonds is important in the law of summation. Therefore, we multiply the power direction of the last bond (last_bond_dir) with the forward path (final_fwrd_path) and add it to the backward path (backwrd_path) multiplied with the power direction of the second last bond (snd_last_bond_dir).

Definition 5.18. ⊢ ∀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 5.19. ⊢ ∀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, as provided in Table 2. The function path_select_t selects a path by matching a strong bond (strong_bond_t) with the last or second last bond of the junction i and its position in the list of junctions j. Similarly, the path_select_f chooses 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.

We have formalized the law of equality for a junction (0,1) in HOL Light, as presented in Definition 5.7. Now, we formalize the equality law for the case, when a junction is following a causal path as follows:

Definition 5.20. ⊢ ∀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 as follows:

Definition 5.21. ⊢ ∀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. The function path_selection_der models equality law by taking derivative on both sides of the equality law's equation, obtained using the type of junction and the causality of the strong bond.

Now, we formalize the summation laws along with a causal path depending on the causality, type and position of a junction as follows:

Definition 5.22. ⊢ ∀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 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 Fig. 6a, 0_*junction* is representing a middle junction, while 1_*junctions* on both sides of the BG representation are side junctions.

Definition 5.23. ⊢ ∀j i t. jun_sum j i t =
then side_jun_sum j i t else middle_jun_sum j i t)

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

Definition 5.24. ⊢ ∀j i t. jun_sum_final j i t =
(jun_sum j i t = Cx(&0))

According to the summation law of a junction, the sum of all the effort or flow variables is equal to zero. Thus, the function jun_sum_final models the final form (summation equal to zero) of the summation law.

Definition 5.25. ⊢ ∀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.

Definition 5.26. ⊢ ∀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 BG 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 returns both summation jun_sum_final and the equality law based on the causality of a resistive element and the type of a junction.

Definition 5.27. ⊢ ∀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 (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 (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 F) else F)
⊢ ∀ j j1 t. branch_main j j1 t = (if j = [ ] ∨ j1 = [ ] then F
else (branch_jun j j1 (LENGTH j - 1) t))

The function branch_presence accepts two different lists 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 set 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 branches, respectively. The presence of a branch 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 between them, we have assigned a number branch_num to them as described in Section 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 a Boolean value **T**, if the connecting bond is a gyrator (GY) and provides the 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 Fig. 6b, 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 a boolean **F**, if the lists j or j1 are empty. Otherwise, the function branch_jun checks all the junctions of list j for branch presence.

Definition 5.28. ⊢ ∀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 (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 (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 F) else F)

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 set 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, matches the branch numbers of both junctions i and i1 (to check that they are connected together or not) and finally checks the presence of the gyrator component (integer value 6). If the branch numbers of both junctions do not match or there is no branch then the returned value is **F**.

Definition 5.29. ⊢ ∀j i k i1 n t. loop_bonds j i k i1 0 t =
(if 0 = k then F else loop_presence j i k i1 0 t) ∧
loop_bonds j i k i1 (SUC n) t = (if SUC n = k then F
else (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 junction 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 5.30. ⊢ ∀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 = (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 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 the connecting bond of the loop is attached.

Definition 5.31. ⊢ ∀j i k t. causal_loop j i k t =
(if (branch j i k = T) ∧ ∼(branch_num j i k = 0)
then (loop_jun_lst j i k t) else F

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

5.5. Formalization of the case selection

Here, we define some cases of a BG representation based on our formalization till now.

Definition 5.32. ⊢ ∀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 =
(jun_sum_final j i t) ∨ (causal_loop j i k t)
⊢ ∀j i k t. branch_type_b j i k t =
(path_selection j i k t) ∨ (causal_loop j i k t)

The functions 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 the summation and equality laws are deducted, respectively. For example, in Fig. 6b, the bond of the 1_*junctions* with a component I (having integral causality) results into summation law and the bond of 1_*junction* connected with component C (having integral causality) results into equality law by following causal paths. The function zero_ordr_ele models junctions with the resistive component (R). The function diff_causal_ele_a is similar to the first order case except the elements with differential causality. These functions provide differential equations of the BG representation. Lastly, the function branch_type_a is also similar to first order case except the presence of a causal loop. The detailed definition can be found in our HOL Light proof script [46] for detailed cases of a BG representation.

Definition 5.33. ⊢ ∀j i k t. case_selection j i k t =
(if P1 then frst_ordr_ele_a j i t
(else if P2 then frst_ordr_ele_b j i k t
else if P3 then zero_ordr_ele j i k t
else if P4 then diff_causal_ele_a j i t
else if P5 then branch_type_a j i k t
else if P6 then T else F)

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. The detailed formalized form of the parts P1, P2, P3 and P4 is given in Table 6.

Definition 5.34. ⊢ ∀j i k t. (bond_selection j i 0 t =
case_selection j i 0 t) ∧
bond_selection j i (SUC k) t = (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 the function case_selection to the bonds of a junction. Similarly, the function bond_selection_lst applies case selection to all the bonds of a junction.

Definition 5.35. ⊢ ∀j i t. jun_selection j i t =
bond_selection_lst j 0 t ∧
jun_selection j (SUC i) t = (bond_selection_lst j (SUC i) t) ∧
(jun_selection j i t)
⊢ ∀j t. bg_main j t = (if j = [ ] then F
else (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 that 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.

5.6. Formalization of the state–space model

Finally, the state–space model for a BG representation is formalized in HOL Light as:

Definition 5.36. ⊢ ∀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 provides a state–space model for a BG representation.

Table 6
Formalized parts of Definition 5.33.

Part	Formalized form
P1	(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)
P2	(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)
P3	(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)
P4	(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)
P5	((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)
P6	(type_of_ele j i k = 0 ∨ type_of_ele j i k = 1 ∨ type_of_ele j i k = 5 ∨ type_of_ele j i k = 6)

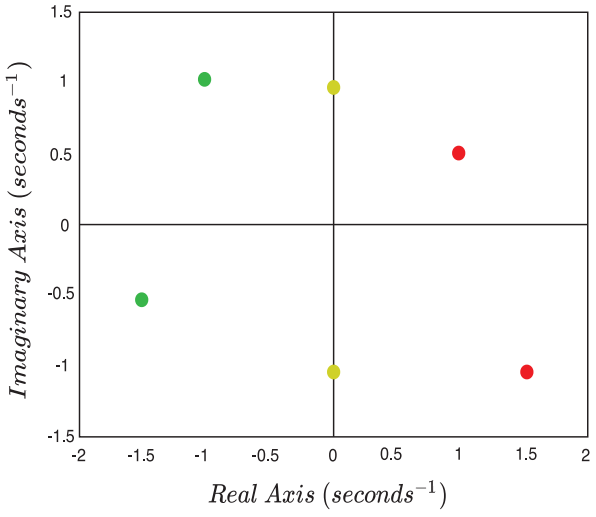


Fig. 7. Stability regions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6. Formal verification of the bond graphs properties

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 [47]. Thus, a stable system provides a stable response/output to a bounded input. The BG representation of a system is expressed as state–space models, which are based on vectors and matrices, in particular the 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 system, the eigenvalues of the system's matrix lie in the left half of the complex plane.

In Fig. 7, green dots represent eigenvalues of a stable system. Similarly, for an unstable system, the eigenvalues of the system's matrix lie in the right half of the complex plane, which are represented as red dots in Fig. 7. Moreover, 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 Fig. 7.

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 the eigenvalues of the system matrix A by solving the following equations involving the 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 [35]. 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 6.1. ⊢ ∀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 function 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 thus ensures a stable system. Here, cdet models the 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 model an unstable and a marginally stable system, respectively.

In the state–space representation, we deal with matrices, vectors and their corresponding operations. For the stability of a system, entries of the matrix A are analyzed. Since, there are different arithmetic operations (+, ∗) involved and the parameters of the system matrix contain round-off values in our case [14] so, rather than taking particular values of the parameters, we took general values and verified the theorems for (stable, unstable, marginally stable) square matrices of dimension 2 and 3.

Theorem 6.1. ⊢ ∀a11 a12 a13 a21 a22 a23 a31 a32 a33 b1 c1 d1 r. Cx (-- a33 - a22 - a11) = Cx b1 + Cx r ∧
Cx (-- a13 ∗ a31 - a12 ∗ a21 - a23 ∗ a32 + a22 ∗ a33 + a11 ∗ a33 + a11 ∗ a22) = Cx c1 + Cx (b1 ∗ r)∧
Cx (-- a11 ∗ a22 ∗ a33 - a12 ∗ a31 ∗ a23 - a13 ∗ a21 ∗ a32 + a11 ∗ a23 ∗ a32 + a12 ∗ a21 ∗ a33 + a13 ∗ a31 ∗ a22)= Cx (c1 ∗ r) ∧
(&0 < r ∨ (&0 < b1 ∧ (b1 pow 2 - &4 ∗ c1 < &0 ∨ b1 pow 2 - &4 ∗c1 = &0) ∨
(&0 < b1 pow 2 - &4 ∗ c1 ∧ ((b1 pow 2 - &4 ∗ c1) < b1 ∨ − b1 < $\sqrt{(b1 \text{ pow } 2 - \&4 * c1)}$))))
⇒ stable_sys $\begin{bmatrix} Cx\ a11 & Cx\ a12 & Cx\ a13 \\ Cx\ a21 & Cx\ a22 & Cx\ a23 \\ Cx\ a31 & Cx\ a32 & Cx\ a33 \end{bmatrix}$

Table 7
Properties of complex matrices.

Name	Formalized form
`Stable matrix property`	$\vdash_{thm}$ ∀A. **[A1]** (diagonal_cmatrix A ∨ lt_cmatrix A ∨ ut_cmatrix A) ∧ **[A2]** (stable_diagonal_ele_cond A) ⇒ stable_sys A
`Unstable Matrix Property`	$\vdash_{thm}$ ∀A. **[A1]** (diagonal_cmatrix A ∨ lt_cmatrix A ∨ ut_cmatrix A) ∧ **[A2]** (unstable_diagonal_ele_cond A) ⇒ unstable_sys A
`Marginally Stable Matrix Property`	$\vdash_{thm}$ ∀A. **[A1]** (diagonal_cmatrix A ∨ lt_cmatrix A ∨ ut_cmatrix A) ∧ **[A2]** (marg_stable_diagonal_ele_cond A) ⇒ marginally_stable_sys A

The above theorem provides all possible conditions on the entries of a matrix with dimension 3 × 3 to be stable. This theorem is formally verified using complex matrix theory, which is developed as part of this work [46] and multivariate complex and real analysis theories available in the library of the HOL Light theorem prover. We used lemmas in the formalization of matrix stability to solve characteristics polynomial of the given matrix. These lemmas can be found in our HOL Light proof script [46].

Next, we verify some important properties of the state–space models of the given BG representations, when they represent some specific matrices, such as, upper triangular, lower triangular and diagonal matrices [48,49]. We formalize these matrices as follows:

Definition 6.2. ⊢ ∀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 its entries below the main diagonal equal to zero. Similarly, a square matrix whose entries above the main diagonal are zero is called the lower triangular matrix and is formalized in HOL Light as a function lt_matrix. The function diagonal_cmatrix provides a diagonal matrix, which is a special case of triangular matrices and has all its entries, other than diagonal, equal to zero.

Table 7 presents formally verified properties of the complex matrices, providing stability, unstability and marginal stability under some conditions and the verification of these theorems are mainly based on the properties of complex-valued matrices along with some complex arithmetic reasoning.

7. Application: Anthropomorphic mechatronic prosthetic hand

Anthropomorphic Mechatronic Hand [14] is a robotic hand consisting of electrical and mechanical components. It is capable of conducting movements of a human hand and is widely used in robotics, such as industrial, service, surgical robots [50–52], 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 as shown in Fig. 8. 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 as shown in Fig. 8, 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 [53].

A robotic hand replicates the structure of human hand consisting of bones, joints and muscles by using frames, pulley-string mechanisms and electrical actuators, respectively. We formally analyzed all the movements of fingers of an anthropomorphic prosthetic hand [14] by applying our formalization of BG, as described in Section 5. In this paper, we only present formal analysis of flexion and extension movements of the index finger.

A BG representation of an anthropomorphic mechatronic prosthetic hand is presented in Fig. 9. It consists of BG models of index finger for the case of abduction and adduction, and flexion and extension movements. It also contains BG models of thumb for the flexion and extension, abduction and adduction, and up and down movements. We have formalized each of these index finger and thumb movements using our formalization of the BG representation, presented in Section 5.

In this paper, we only present the formalization of flexion and extension movements of the index finger. In this regard, we model the index finger movements namely flexion (flex.) and extension (ext.) in HOL Light as follows:

Definition 7.1. ⊢ ∀e411 e412 f412 e413 f413 e421 f422 e423 f423 e424 f424 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 $\left(\frac{\&1}{\text{Motor\_4}}\right)$; 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 $\left(\frac{\&1}{\text{Motor\_4}}\right)$; Cx Motor_4],(e423,f423); T,F,(F,0),5,[Cx $\left(\frac{\&1}{\text{Geer\_4}}\right)$; 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 $\left(\frac{\&1}{\text{Geer\_4}}\right)$],(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 complex-valued constants Jm_4,J_4 represents inertial mass of motor and frames, pulleys and strings, respectively. Motor_4,Geer_4 is the ratio of the gyrator and gear, respectively [14,46].

Similarly, we formalized thumb up and down movements of the prosthetic hand [46]. The next step is to formalize the state–space model of the index finger (flex. and ext.) that requires system and input matrices, state and input vectors, and derivative of the state vector, as given in Definition 5.36. We only provide the formal model of the system matrix here. The formalization of the other vectors and matrices can be found at our project webpage [46]. The system matrix required for the state–space representation of index finger movements (flex.and ext) is defined in HOL Light as follows:

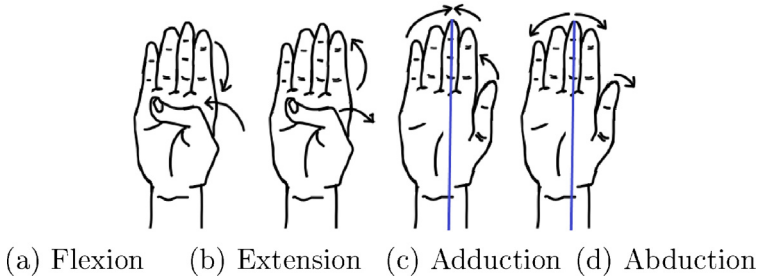


Fig. 8. Movements of a human hand's fingers.

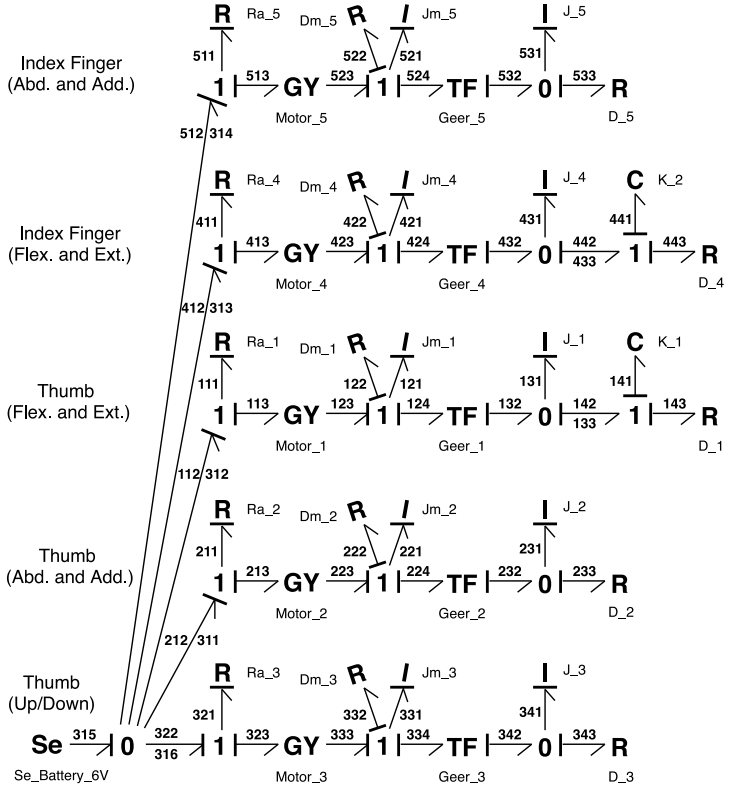


Fig. 9. Bond Graph of anthropomorphic mechatronic prosthetic hand.

Definition 7.2. $\vdash \forall Ra_4\ Motor_4\ Jm_4\ Dm_4\ Geer_4\ J_4\ D_4\ K_2.$ index_finger_flex_ext_system_mat

$$Ra_4\ Motor_4\ Jm_4\ Dm_4\ Geer_4\ J_4\ D_4\ K_2 = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix}$$

where

$$a11 = Cx\left(\left(\frac{-Dm_4}{Jm_4}\right) - \left(\frac{(Geer_4)\ pow\ 2 * D_4}{Jm_4}\right) - \left(\frac{(Motor_4)\ pow\ 2}{Jm_4 * Ra_4}\right)\right)$$

$$a12 = Cx\left(\frac{Geer_4 * D_4}{J_4}\right),$$

$$a13 = Cx\left(\frac{-Geer_4}{K_2}\right),$$

$$a21 = Cx\left(\frac{Geer_4 * D_4}{Jm_4}\right),$$

$$a22 = Cx\left(\frac{-D_4}{J_4}\right),$$

$$a23 = Cx\left(\frac{\&1}{K_2}\right),$$

$$a31 = Cx\left(\frac{Geer_4}{Jm_4}\right),$$

$$a32 = Cx\left(\frac{-\&1}{J_4}\right),$$

$$a33 = Cx\ (\&0)$$

Next, we formally verify the implementation of the index finger for the flexion and extension movements, connection between index finger and thumb, simplification of the index finger equations and state–space representation of the index finger. These verification results can be found at our project webpage.

Finally, we verify the stability of the index finger (flex. and ext.) by utilizing the formally verified system matrix index_finger _flex_ext_system_mat in HOL Light as the following theorem:

Theorem 7.1. $\vdash \forall Ra\_4\ Motor\_4\ Jm\_4\ Dm\_4\ Geer\_4\ J\_4\ D\_4\ K\_2.$ [**A1**] $Cx\left(\frac{D_4}{J_4} - \left(\frac{-Dm_4}{Jm_4} - \frac{(Geer_4)^2 * D_4}{Jm_4} - \frac{(Motor_4)^2}{Jm_4 * Ra_4}\right)\right) = Cx\ b1 + Cx\ r \wedge$

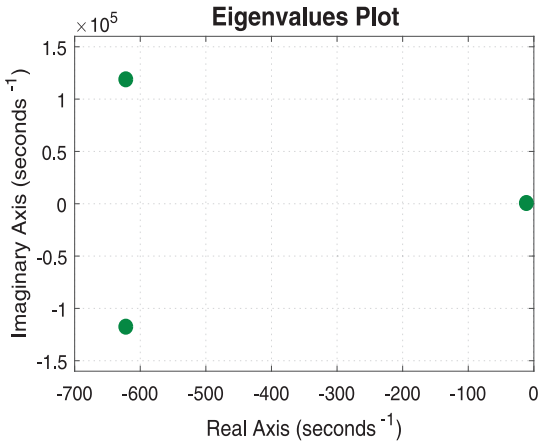


Fig. 10. Stable Index Finger (Flex. and Ext.).

[**A2**] $Cx\left(\frac{Geer_4}{K_2} * \frac{Geer_4}{Jm_4} - \frac{Geer_4 * D_4}{J_4} * \frac{Geer_4 * D_4}{Jm_4} - \frac{\&1}{K_2} * \frac{-\&1}{J_4} + \left(\frac{-Dm_4}{Jm_4} - \frac{(Geer_4)^2 * D_4}{Jm_4} - \frac{(Motor_4)^2}{Jm_4 * Ra_4}\right) * \frac{-D_4}{J_4}\right) = Cx\ c1 + Cx\ (b1 * r) \wedge$

[**A3**] $Cx\left(- \frac{Geer_4 * D_4}{J_4} * \frac{Geer_4}{Jm_4} * \frac{\&1}{K_2} - \frac{-Geer_4}{K_2} * \frac{Geer_4 * D_4}{Jm_4} * \frac{-\&1}{J_4} + \left(\frac{-Dm_4}{Jm_4} - \frac{(Geer_4)^2 * D_4}{Jm_4} - \frac{(Motor_4)^2}{Jm_4 * Ra_4}\right) * \frac{\&1}{K_2} * \frac{-\&1}{J_4} + \frac{-Geer_4}{K_2} * \frac{Geer_4}{Jm_4} * \frac{-D_4}{J_4}\right) = Cx\ (c1 * r) \wedge$

[**A4**] $(\&0 < r \vee (\&0 < b1 \wedge (b1\ pow\ 2 - \&4 * c1 < \&0 \vee b1\ pow\ 2 - \&4 * c1 = \&0) \vee (\&0 < b1\ pow\ 2 - \&4 * c1 \wedge ((b1\ pow\ 2 - \&4 * c1) < b1 \vee -b1 < \sqrt{(b1\ pow\ 2 - \&4 * c1)}))))$
$\Rightarrow$ stable_sys (index_finger_flex_ext_system_mat Ra_4 Motor_4 Jm_4 Dm_4 Geer_4 J_4 D_4 K_2)

Assumptions A1-A3 capture the necessary conditions for the stability of the index finger (flex. and ext.). Finally, the conclusion provides the stable index finger. The verification of Theorem 7.1 is mainly based on Theorem 6.1 alongwith some complex arithmetic reasoning.

To verify the assumptions for all possible values of the parameters, we encoded our formalized Theorem 6.1 in MATLAB in the form of an algorithm to verify the status (stable, marginally stable or unstable) of the given system and it provides results in the graphical form as shown in Fig. 10. We provided the values of the parameters such as Ra_4, Motor_4, Jm_4, Dm_4, Geer_4, J_4, D_4 and K_2 as an input to the MATLAB algorithm and it provides a graph of the eigenvalues as an output. All of the roots of a polynomial (eigenvalues) lie in the left half of the complex plane in Fig. 10, which indicates the system of index finger (flex. and ext.) is stable.

The distinguishing features of our proposed framework, as compared to the traditional analysis techniques, include (1) all of the parameters of BG are formally (clearly) defined along with a data type, (2) all of the variables and functions are of generic nature, i.e., universally quantified and (3) the utilization of the formally verified algorithm of BG. Soundness is assured in functions and theorems as every new theorem is verified by applying basic axioms and inference rules or any other previously verified theorems/inference rules due to the usage of an interactive theorem prover. In the traditional analysis techniques, like computer-based simulations, there are chances of errors due to the misinterpretation of the parameters and their sampling based nature. We have demonstrated the effectiveness of our formal stability analysis approach by analyzing an anthropomorphic prosthetic mechatronic hand using HOL Light. It can be seen in our final results that all of the verified theorems are of generic nature, i.e., all of the functions and variables are universally quantified and thus can be specialized based on the requirement of the analysis of the anthropomorphic prosthetic mechatronic hand. Whereas, in the case of computer based simulations, i.e., the MATLAB based analysis, we need to model each case individually. Also, the inherent soundness of the theorem proving technique ensures that all the required assumptions are explicitly present along with the theorem, which are generally ignore in the corresponding MATLAB based analysis and their absence may affect the accuracy of the corresponding analysis. Therefore, we encoded our formally verified stability theorems alongside all of its assumptions in MATLAB, which certify the analysis performed in MATLAB. Moreover, due to the undecidable nature of the higher-order logic, the proposed formalization involves a significant user involvement. However, we have developed simplifiers, such as, PATHS_SELECTION_TAC, BACKWRD_PATH_TAC, FWRD_PATH_TAC, CASE_SELECTION_TAC that significantly reduce the user guidance in the proof reasoning process. Finally, the verification and the formal analysis of a BG representation ensures the accurate results and stability of a dynamic system (Anthropomorphic Robotic Hand).

8. Conclusion

In this paper, we proposed a framework for formally analyzing BG representations of engineering and physical systems in the HOL Light theorem prover. Firstly, we proposed an analysis methodology of BG that contains components, laws, causal paths, causal loop, branch and state–space model etc. Secondly, we formally verified all the concepts/steps of BG presented in its analysis methodology. Thirdly, the theorems of generic nature are verified for the stability analysis of matrices. This formal stability analysis requires the formalization of the complex matrices, which are also developed as a part of the proposed framework. Moreover, we encoded these formally verified theorems of matrices in MATLAB to perform the formal stability analysis. Finally, we performed the stability analysis of an anthropomorphic mechatronic prosthetic hand to obtain the accurate state–space model and stable movements performed by different fingers of a robotic hand. In future, our aim is to extend our current formalization of BGs to incorporate algebraic loops and non-linear systems. In addition, we plan to apply our formalization of BGs to some other complex applications involving robotics and biological systems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Paynter HM. Analysis and design of engineering systems. MIT Press; 1961.
[2] Wang L-a, Li Q, Liang X-j. 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. IEEE; 2010, p. 744–7.
[3] Díaz-Zuccarini V, Pichardo-Almarza C. On the formalization of multi-scale and multi-science processes for integrative biology. Interface Focus 2011;1(3):426–37.
[4] Granda J, Montgomery R. Automated modeling and simulation using the bond graph method for the aerospace industry. In: AIAA modeling and simulation technologies conference and exhibit. 2003, p. 5527.
[5] Gawthrop PJ, Bevan GP. Bond-graph modeling. IEEE Control Syst Mag 2007;27(2):24–45.
[6] Iordanova V, Abouaissa H, Jolly D. Bond-graphs traffic flow modelling and feedback control. IFAC Proc Vol 2006;39(3):345–50.

[7] Rosenberg RC, Feurzeig W, Wexelblat P. Bond graphs and ENPORT in elementary physics instruction. IEEE Trans Man-Mach Syst 1970;11(4):170–4.
[8] Hales MK, Rosenberg RC. Enport model builder: An improved tool for multiport modeling of mechatronic systems. Simul Ser 2001;33(1):152–7.
[9] Delgado M, Brie C. DESIS-A modeling and simulation package based on bond graphs. IFAC Proc Vol 1991;24(4):215–20.
[10] Ballance DJ, Bevan GP, Gawthrop PJ, Diston DJ. Model transformation tools (MTT): the open source bond graph project. 2005.
[11] Granda JJ, Reus J. 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, Vol. 2. IEEE; 1997, p. 1542–7.
[12] Simulator 20-sim. 2019, https://www.20sim.com/features/simulator/. (Online; Acessed 28/8/2020 20:2).
[13] Harrison J. HOL light: A tutorial introduction. In: Srivas M, Camilleri A, editors. Proceedings of the first international conference on formal methods in computer-aided design (FMCAD'96). Lecture notes in computer science, vol. 1166, Springer-Verlag; 1996, p. 265–9.
[14] Saeed MT, Khaydarov S, Ashagre BL, Zafar M. Comprehensive bond graph modeling and optimal control of an anthropomorphic mechatronic prosthetic hand. In: 2019 IEEE international conference on mechatronics and automation (ICMA). IEEE; 2019, p. 2006–11.
[15] Shi Z, Guan Y, Li X. Formalization of complex analysis and matrix theory. Springer; 2020.
[16] Harrison J. Formalizing basic complex analysis. In: Matuszewski R, Zalewska A, editors. From insight to proof: Festschrift in honour of Andrzej Trybulec. Studies in logic, grammar and rhetoric, vol. 10(23), University of Białystok; 2007, p. 151–65, URL http://mizar.org/trybulec65/.
[17] Harrison J. Hol light multivariate theory. 2021, https://github.com/jrh13/hol-light/tree/master/Multivariate. (Online; Acessed 21/6/2021 19:48).
[18] Standard library the coq proof assistant. 2014, https://coq.inria.fr/distrib/current/stdlib/. (Online; Acessed 21/6/2021 20:11).
[19] Thiemann R, Yamada A. Algebraic numbers in Isabelle/HOL. In: International conference on interactive theorem proving. Springer; 2016, p. 391–408.
[20] Brian Huffman JH. Theory real vector spaces. 2021, https://isabelle.in.tum.de/dist/library/HOL/HOL/Real_Vector_Spaces.html. (Online; Acessed 21/6/2021 20:26).
[21] Obua S. Theory matrix. 2021, https://isabelle.in.tum.de/library/HOL/HOL-Matrix_LP/Matrix.html. (Online; Acessed 21/6/2021 20:27).
[22] Rashid A, Hasan O. On the formalization of Fourier transform in higher-order logic (rough diamond). 2016.
[23] Rashid A, Hasan O. Formal analysis of continuous-time systems using Fourier transform. J Symbolic Comput 2019;90:65–88.
[24] Rashid A, Hasan O. Formalization of transform methods using HOL light. In: International conference on intelligent computer mathematics. Springer; 2017, p. 319–32.
[25] Rashid A. Formalization of transform methods using higher-order-logic theorem proving (Ph.D. thesis), Islamabad, Pakistan: Ph.D. thesis, National University of Science & Technology; 2019.
[26] Rashid A, Hasan O. Formal analysis of the continuous dynamics of cyber–physical systems using theorem proving. J Syst Archit 2021;112:101850.
[27] Rashid A, Hasan O. Formalization of Lerch's theorem using HOL light. 2018, arXiv preprint arXiv:1806.03049.
[28] Rashid A, Hasan O. Formal analysis of robotic cell injection systems using theorem proving. In: International workshop on design, modeling, and evaluation of cyber physical systems. Springer; 2017, p. 127–41.
[29] Rashid A, Hasan O. Formal verification of robotic cell injection systems up to 4-DOF using HOL light. Form Asp Comput 2020;32(2):229–50.
[30] Beillahi SM, Siddique U, Tahar S. Formal analysis of engineering systems based on signal-flow-graph theory. In: International workshop on numerical software verification. Springer; 2016, p. 31–46.
[31] Siddique U, Aravantinos V, Tahar S. On the formal analysis of geometrical optics in HOL. In: International workshop on automated deduction in geometry. Springer; 2012, p. 161–80.
[32] Abed S, Rashid A, Hasan O. Formal analysis of unmanned aerial vehicles using higher-order-logic theorem proving. J Aerosp Inf Syst 2020;17(9):481–95.
[33] Abed S, Rashid A, Hasan O. Formal analysis of the biological circuits using higher-order-logic theorem proving. In: Proceedings of the 35th annual ACM symposium on applied computing. 2020; p. 3–7.
[34] Rashid A, Hasan O, et al. Formal reasoning about synthetic biology using higher-order-logic theorem proving. IET Syst Biol 2020;14(5):271–83.
[35] Ahmed A, Hasan O, Awwad F. Formal stability analysis of control systems. In: International workshop on formal techniques for safety-critical systems. Springer; 2018, p. 3–17.
[36] Rashid A, Siddique U, Tahar S. Formal verification of cyber-physical systems using theorem proving. In: International workshop on formal techniques for safety-critical systems. Springer; 2019, p. 3–18.
[37] Binyameen F, Osman H, Sohail I. Formal kinematic analysis of the two-link planar manipulator. In: International conference on formal engineering methods. 2013.
[38] Siddique U, Aravantinos V, Tahar S. Formal stability analysis of optical resonators. In: NASA formal methods symposium. Springer; 2013, p. 368–82.
[39] Schumann JM. Automated theorem proving in software engineering. Springer Science & Business Media; 2001.
[40] Broenink JF. Introduction to physical systems modelling with bond graphs. In: SiE whitebook on simulation methodologies, Vol. 31. 1999, p. 2.
[41] Borutzky W. Bond graph modelling of engineering systems, Vol. 103. Springer; 2011.
[42] Joseph B, Martens H. The method of relaxed causality in the bond graph analysis of nonlinear systems. 1974.
[43] Borutzky W. Derivation of mathematical models from bond graphs. In: Bond graph methodology: Development and analysis of multidisciplinary dynamic system models. Springer; 2010, p. 89–128.
[44] Kypuros J. System dynamics and control with bond graph modeling. CRC Press; 2013.
[45] Thoma JU. Introduction to bond graphs and their applications. Elsevier; 2016.
[46] Qasim U, Rashid A. Formalization of bond graph using higher-order-logic theorem proving: Project code. 2021, http://save.seecs.nust.edu.pk/fbgholtp.
[47] Nise NS. Control systems engineering, (with CD). John Wiley & Sons; 2007.
[48] Ricardo H. A modern introduction to linear algebra. CRC Press; 2009.
[49] Mirsky L. An introduction to linear algebra. Courier Corporation; 2012.
[50] Gaiser I, Schulz S, Kargov A, Klosek H, Bierbaum A, Pylatiuk C, Oberle R, Werner T, Asfour T, Bretthauer G, et al. A new anthropomorphic robotic hand. In: Humanoids 2008-8th IEEE-RAS international conference on humanoid robots. IEEE; 2008, p. 418–22.
[51] Rhee C, Chung W, Kim M, Shim Y, Lee H. 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, Vol. 4. IEEE; 2004, p. 4011–6.
[52] Honarpardaz M, Tarkian M, Ölvander J, Feng X. Finger design automation for industrial robot grippers: A review. Robot Auton Syst 2017;87:104–19.
[53] Wang L, Meydan T, Williams PI. A two-axis goniometric sensor for tracking finger motion. Sensors 2017;17(4):770.