Formalisation, Abstraction and Refinement of Bond Graphs

Richard Banach[1][0000−0002−0243−9434] and John Baugh[2][0000−0002−4999−7505]

[1] Department of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
[2] Department of Civil, Construction and Environmental Engineering,
North Carolina State University, Raleigh, North Carolina 27695-7908, U.S.A.
richard.banach@manchester.ac.uk jwb@ncsu.edu

Abstract. Bond graphs represent the structure and functionality of mechatronic systems from a power flow perspective. Unfortunately, presentations of bond graphs are replete with ambiguity, significantly impeding understanding. A formalisation of the essentials of bond graphs is given, together with a formalisation of bond graph transformation, which can directly express abstraction and refinement of bond graphs.

Keywords: Bond Graph · Formalisation · Abstraction · Refinement.

1 Introduction

Bond graphs were introduced in the work of Paynter in 1959 [8, 9]. These days the most authoritative presentation is [6]. From the large related literature we can cite [3, 6, 7].

Even the best presentations, though, are replete with ambiguity, often arising from a non-standard use of language, that leaves the reader who is more used to conventional parlance in engineering terminology, feeling insecure and confused. The topic is thus ripe for a reappraisal using the mathematical tools that bring precision to concepts in computer science. This paper introduces such a reformulation. For lack of space, only the essentials are covered, and the writing is rather terse. A fuller treatment is in [2].

The rest of the paper is as follows. Section 2 outlines the kind of physical theories that bond graphs are used for. Section 3 covers the essentials of bond graph structure. Section 4 shepherds the details in Section 3 to form the category $\mathcal{BGP}att$, which we discuss briefly. Section 5 covers the essentials of bond graph transformation, relating this to $\mathcal{BGP}att$. Section 6 shows how transformations can be used for abstraction and refinement of bond graphs. Section 7 concludes.

2 Classical Physical Theories for Classical Engineering

Bond graphs target the classical regime of physical theories well established at the end of the 19th century, and the engineering done using those theories. The usual remit of these includes mechanical, rotational, electrical, hydraulic, and thermal domains.

Somewhat remarkably, the physical theories underpinning all these domains share a large degree of similarity, a fact exploited in the creation of the bond graph formalism. In this paper, for brevity, we briefly indicate the relevance to the mechanical and electrical domains. We axiomatise the common framework as follows.

[PT.1] A system consists of interconnected devices, and operates within an environment from which it is separated by a notional boundary. A system can input or output energy from the environment through specific devices. Aside from this, the system is assumed to be isolated from the environment.

[PT.2] The classical physics relevant to bond graphs is captured, in general, by a system of second order ordinary differential equations (ODE) of the form:

$$\Phi(q'', q', q) = e \tag{1}$$

Of most interest is the case where Φ is a linear constant coefficients (LCC) ODE:

$$L\,\frac{\mathrm{d}^2 q}{\mathrm{d}t^2} + R\,\frac{\mathrm{d}q}{\mathrm{d}t} + K\,q = e \tag{2}$$

The system (1) or (2) concerns the behaviour of one (or more) generalised displacement(s), referred to as **gendis** with typical symbol q (*mech*: displacement; *elec*: charge). The gendis time derivative q' is called the **flow**, with typical symbol f (*mech*: velocity; *elec*: current). The gendis second time derivative q'' is called the generalised acceleration **genacc**, with typical symbol a (*mech*: acceleration; *elec*: induction). These all occur in the LHS of (1)-(2).

On the RHS of (1)-(2) is the **effort**, typical symbol e (*mech*: force; *elec*: voltage). The time integral (over a given time interval T) of the effort $\int_T e\,dt$ is called the generalised momentum **genmom** (accumulated over time T), with typical symbol p. The time derivative of the effort is normally of no interest.

[PT.3] Of particular importance among the variables mentioned is the product of effort and flow, because $e \times f$ is **power**, i.e., the rate at which energy is processed. The transfer and processing of power is crucial for the majority of engineered systems. According to **[PT.1]**, energy can only enter or exit a system through specific kinds of device. Therefore, all other devices conserve energy within the system.

[PT.4] Engineered systems are made by connecting relatively simple devices. We describe the most important ones now. Of particular utility are devices which are special cases of (2) that keep only one of the LHS terms.

Dissipator: R-device (*mech*: dashpot; *elec*: resistor)	$R\,f = e$	(3)
Compliant: C-device (*mech*: spring; *elec*: capacitor)	$K\,q = e$	(4)
Inertor: L-device (*mech*: mass; *elec*: inductor)	$L\,a = e$	(5)

A dissipator is a device that can output energy into the environment in the form of heat. Compliants and inertors are devices that store energy. Specifically, the power they receive is accumulated within the device as stored energy, to be released back into the rest of the system later.

Sources input power to/from the system of interest in predefined ways.

Effort source: SE-device (*mech*: force; *elec*: voltage)	$e = \Phi_E(t)$	(6)

Flow source: SF-device (*mech*: velocity; *elec*: current) $f = \Phi_F(t)$ (7)

Note that the power input and output to/from each of these cases is not determined by equations (6)-(7) alone (since the other variable is not specified), but by the behaviour of the rest of the system that they are connected to.

All the above devices are connected to the rest of the system via a single power connection, i.e., there is only one effort variable and one flow variable. Transformers and gyrators are devices that are connected to two power connections (two efforts and two flows), and allow non-trivial tradeoffs between the effort and the flow in the two connections.

Transformer: TR-device (*mech*: lever; *elec*: transformer)

$$e_1 = h\, e_2 \quad \text{and} \quad h\, f_1 = f_2 \tag{8}$$

Gyrator: GY-device (*mech*: gyroscope; *elec*: transducer)

$$e_1 = g\, f_2 \quad \text{and} \quad g\, f_1 = e_2 \tag{9}$$

Junctions are devices that distribute power among several power connections $1 \ldots n$ (each with its own effort and flow), while neither storing nor dissipating energy. Aside from transformers and gyrators just discussed, the only remaining cases that arise are the common effort and common flow cases.

Common effort: E-device

(*mech*: common force; *elec*: common voltage, Kirchoff's Current Law)

$$e_1 = e_2 = \ldots = e_n \quad \text{and} \quad f_1 + f_2 + f_3 + \ldots + f_n = 0 \tag{10}$$

Common flow: F-device

(*mech*: common velocity; *elec*: common current, Kirchoff's Voltage Law)

$$e_1 + e_2 + e_3 + \ldots + e_n = 0 \quad \text{and} \quad f_1 = f_2 = \ldots = f_n \tag{11}$$

Noting that n is not fixed, **E** and **F** devices for different n are different devices.

[PT.5] From the bond graph perspective, the individual power connections to a device are conceptualised as power **port**s, through which power flows into or out of the device. Dissipators, compliants and inertors are therefore **one port** devices. Power sources are also one port devices. Transformers and gyrators are **two port** devices, while junctions are **three (or more) port** devices. For each category of device, all of its ports are individually labelled.

[PT.6] Since power is the product of an effort variable and a flow variable, each port is associated with an (effort, flow) variable pair whose values at any point in time define the power flowing through it.

[PT.7] All the variables involved in the description of a system are typed using a consistent system of dimensions and units. It is assumed that this typing is sufficiently finegrained that variables from different physical domains cannot have the same type. We do not have space to elaborate details, but since the only property of dimensions and units that we use is whether two instances are the same or not, it is sufficient to assume a set $\mathcal{DT} \times \mathcal{UT}$ of (dimension, unit) terms, that type the variables we need.

[PT.8] We refer to the elements of a system using a hierarchical naming convention. Thus, if Z-devices have ports p, then Z.p names the p ports of Z-devices. And if the

effort variables of those ports are called e, then Z.$p.e$ names those effort variables. Analogously, Z.$p.f$ would name the flow variables corresponding to Z.$p.e$. Z.$p.e$.DU names the dimensions and units of Z.$p.e$, while Z.$p.f$.DU names the dimensions and units of Z.$p.f$.

[PT.9] For every (effort, flow) variable pair in a system (belonging to a port p of device Z say), for example (Z.$p.e$, Z.$p.f$), there is a **directional indication** (determined by the physics of the device in question and the equations used to quantify its behaviour). This indicates whether the power given by the product Z.$p.e\times$Z.$p.f$ is flowing **in**to or **out**of the port when the value of the product is ***positive***.

For the devices spoken of in **[PT.4]**, there is a standard assignment of **in**/**out** indicators to its ports. Thus, for **R**, **C**, **L** devices, the standard assignment to their single port is **in**. For **SE**, **SF** devices, the standard assignment to their single port is **out**. For **TR**, **GY** devices, the standard assignment is **in** for one port and **out** for the other, depicting positive power flowing through the device. For the **E** and **F** devices, we standardise on a symmetric **in** assignment to all the ports.

3 Bond Graph Basics

Bond graphs are graphs which codify the physical considerations listed above.

[UNDGR] An **undirected graph** is a pair (V, E) where V is a set of vertices, and E is a set of edges. There is a map $\mathrm{ends} : E \to \mathbb{P}(V)$, where $(\forall edg \in E \bullet \mathrm{card}(\mathrm{ends}(edg)) = 2)$ holds, identifying the pair of *distinct* elements of V that any edge edg connects. When necessary, we identify the individual ends of an edge edg, where $\mathrm{ends}(edg) = \{a, b\}$ using (a, edg) and (b, edg). If $\mathrm{ends}(edg) = \{a, b\}$, then we say that edg is incident on a and b.

Our formulation of conventional power level bond graphs (DPLBGs, directed power level bond graphs) starts with PLBGs, which are undirected labelled graphs. It is important in the following to remember that the mathematical details are intended to follow, as closely as reasonable, the constraints that apply in the physical world. This prevents many appealing ways of formulating things mathematically from being applied, because they naturally force aspects of the model to be free, and decoupled from one another, whereas in the physical world, such freedom is not possible. PLBGs are assembled out of the following ingredients. Fig. 1 illustrates the process.

[PLBG.1] There is an alphabet $\mathcal{VL} = \mathcal{BVL} \cup \mathcal{CVL}$ of vertex labels, with basic vertex labels $\mathcal{BVL} = \{\mathbf{R}, \mathbf{C}, \mathbf{L}, \mathbf{SE}, \mathbf{SF}, \mathbf{TR}, \mathbf{GY}, \mathbf{E}, \mathbf{F}\}$, and user defined labels $\mathcal{CVL}$.

[PLBG.2] There is an alphabet $\mathcal{PL}$ of port labels and a map $\mathrm{lab2pts} : \mathcal{VL} \to \mathbb{P}(\mathcal{PL})$, which maps each vertex element label to a set of port labels. (Below, we just say port, instead of port label, for brevity).

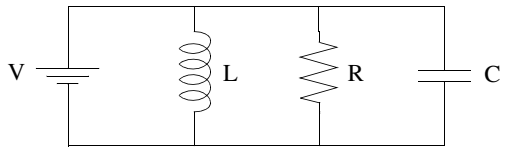
[PLBG.3] There are partial maps $\mathrm{labpt2effDU}$, $\mathrm{labpt2floDU} : \mathcal{VL}\times\mathcal{PL}\pfun\mathcal{DT}\times\mathcal{UT}$ mapping each (vertex label, port) pair to the dimensions and units (not elaborated here) of the (forthcoming) effort and flow variables, and satisfying:

$$(lab, pt) \in \mathrm{dom}(\mathrm{labpt2effDU}) \Leftrightarrow pt \in \mathrm{lab2pts}(lab) \tag{12}$$

$$(lab, pt) \in \mathrm{dom}(\mathrm{labpt2floDU}) \Leftrightarrow pt \in \mathrm{lab2pts}(lab) \tag{13}$$

(a) **R** (b) **R.**R (c) **R.**R **in**
(d) *vr*.**R** (e) *vr*.**R**.R (f) *vr*.**R**.R **in**
(g) *vr*.**R**.R.*e* (h) *vr*.**R**.R.*f* (i) *vr*.**R**.R.*e* = *R* × *vr*.**R**.R.*f*

(j)



(k)

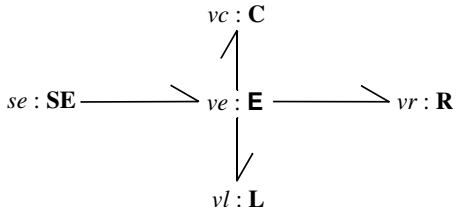


Fig. 1. Stages in bond graph construction: (a) a vertex label (for a dissipator); (b) adding a port; (c) adding a directional indicator; (d)-(f) assigning attributes (a)-(c) to a vertex vr; (g) vr's effort variable; (h) vr's flow variable; (i) vr's constitutive equation; (j) a simple elecrtical circuit embodying a dissipator (among other components); (k) a bond graph of the circuit in (j). Dimensions and units are not shown.

[PLBG.4] There is an alphabet $\mathcal{IO} = \{\textsf{in}, \textsf{out}\}$ of standard directional indicators, and a partial map $\mathrm{labpt2stdio} : \mathcal{VL} \times \mathcal{PL} \nrightarrow \mathcal{IO}$ satisfying:

$$(lab, pt) \in \mathrm{dom}(\mathrm{labpt2stdio}) \Leftrightarrow pt \in \mathrm{lab2pts}(lab) \tag{14}$$

The above clauses capture properties of PLBGs that are common to all vertices sharing the same label. Other properties are defined per vertex. PLBGs can now be constructed.

[PLBG.5] A power level bond graph PLBG is based on an undirected graph $BG = (V, E)$ as in **[UNDGR]**, together with additional machinery as follows.

[PLBG.6] There is a map $\mathrm{ver2lab} : V \to \mathcal{VL}$, assigning each vertex a label.

When map ver2lab is composed with lab2pts, yielding map $\mathrm{ver2pts} = \mathrm{ver2lab} \fatsemi \mathrm{lab2pts} : V \to \mathbb{P}(\mathcal{PL})$, each vertex acquires a set of port labels.

When map ver2lab, with a choice of port, is composed with maps labpt2effDU and labpt2floDU, yielding maps $\mathrm{verpt2effDU} = \mathrm{ver2lab} \times \mathbf{Id} \fatsemi \mathrm{labpt2effDU} : V \times \mathcal{PL} \nrightarrow \mathcal{DT} \times \mathcal{UT}$ and $\mathrm{verpt2floDU} = \mathrm{ver2lab} \times \mathbf{Id} \fatsemi \mathrm{labpt2floDU} : V \times \mathcal{PL} \nrightarrow \mathcal{DT} \times \mathcal{UT}$, each (vertex, port) pair acquires dimensions and units for its effort and flow variables.

When map ver2lab, with a choice of port, is composed with map labpt2stdio, yielding partial map $\mathrm{verpt2stdio} = \mathrm{ver2lab} \times \mathbf{Id} \fatsemi \mathrm{labpt2io} : V \times \mathcal{PL} \nrightarrow \mathcal{IO}$, each (vertex, port) pair acquires its *standard* directional indicator.

[PLBG.7] In practice, and especially for **E**, **F** devices, directional indicators are often assigned per (vertex, port) pair rather than generically per (vertex label, port).

Thus there is a partial map $\mathrm{verpt2io} : V \times \mathcal{PL} \nrightarrow \mathcal{IO}$ satisfying

$$(ver, pt) \in \mathrm{dom}(\mathrm{verpt2io}) \Leftrightarrow pt \in \mathrm{ver2pts}(ver) \tag{15}$$

and $\mathrm{verpt2io}(ver, pt)$ may, or may not, be the same as $\mathrm{verpt2stdio}(ver, pt)$.

There is a partial injective map $\mathrm{verpt2eff} : V \times \mathcal{PL} \rightarrowtail \mathcal{PV}$ giving each (vertex, port) pair (ver, pt) where $pt \in \mathrm{ver2pts}(ver)$, an effort variable with dimensions and units $\mathrm{verpt2effDU}(ver, pt)$. Similarly, $\mathrm{verpt2flo} : V \times \mathcal{PL} \rightarrowtail \mathcal{PV}$ gives each (ver, pt) a flow variable with dimensions and units $\mathrm{verpt2floDU}(ver, pt)$. Also, we must have $\mathrm{ran}(\mathrm{verpt2eff}) \cap \mathrm{ran}(\mathrm{verpt2flo}) = \varnothing$. These variables are referred to by extending the hierarchical convention of **[PT.8]**. Thus $v.\mathrm{Z}.pt.e$ refers to vertex v, with label Z, having port pt, and so $v.\mathrm{Z}.pt.e$ is the relevant effort variable, etc.

There is a map $\mathrm{ver2defs} : V \to \mathrm{physdefs}$, which yields, for each vertex ver, a set of constitutive equations and/or other properties, that define the physical behaviour of the device corresponding to the vertex ver. The free variables of the properties in $\mathrm{ver2defs}$ satisfy:

$$\mathbf{FV}(\mathrm{ver2defs}(ver)) \subseteq \bigcup_{pt \in \mathrm{ver2pt}(ver)} (\mathrm{verpt2eff}(ver, pt) \cup \mathrm{verpt2flo}(ver, pt)) \tag{16}$$

Additionally, the properties in $\mathrm{ver2defs}(ver)$ can depend on generic parameters (from a set $\mathcal{PP}$ say), so there is a map $\mathrm{ver2pars} : V \to \mathcal{PP}$ which satisfies:

$$\mathrm{ver2pars}(ver) = \mathbf{Pars}(\mathrm{ver2defs}(ver)) \tag{17}$$

Additionally, the properties in $\mathrm{ver2defs}(ver)$ can depend on some bound variables. When necessary, we refer to such variables using $\mathbf{BV}(\mathrm{ver2defs}(ver))$.

[PLBG.8] There is a bijection $\mathrm{Eend2verpt} : V \times E \rightarrowtail V \times \mathcal{PL}$, from edge ends in BG, to port occurrences:

$$(ver, edg) \in \mathrm{dom}(\mathrm{Eend2verpt}) \Leftrightarrow edg \text{ is incident on } ver \tag{18}$$

$$(ver, pt) \in \mathrm{ran}(\mathrm{Eend2verpt}) \Leftrightarrow pt \in \mathrm{ver2pts}(ver) \tag{19}$$

$$(\forall ver \bullet ver \in \mathrm{ends}(edg_1) \wedge ver \in \mathrm{ends}(edg_2) \wedge edg_1 \neq edg_2 \Rightarrow \mathrm{Eend2verpt}(ver, edg_1) \neq \mathrm{Eend2verpt}(ver, edg_2)) \tag{20}$$

For each edge $edg \in E$, where $\mathrm{ends}(edg) = \{a, b\}$, the effort and flow variables at the ends of edg, have the same dimensions and units:

$$\mathrm{verpt2effDU}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2effDU}(\mathrm{Eend2verpt}(b, edg)) \tag{21}$$

$$\mathrm{verpt2floDU}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2floDU}(\mathrm{Eend2verpt}(b, edg)) \tag{22}$$

[PLBG.9] There is a map $\mathrm{edge2dir} : E \to \mathrm{physdir}$, where physdir is a set of equalities and antiequalities between effort and flow variables, and such that for all edges $edg \in E$ (where $\mathrm{ends}(edg) = \{a, b\}$):

$$\mathrm{verpt2io}(\mathrm{Eend2verpt}(a, edg)) \neq \mathrm{verpt2io}(\mathrm{Eend2verpt}(b, edg)) \Rightarrow$$

$$\begin{aligned}&\mathrm{edge2dir}(edg) = \{\\ &\quad \mathrm{verpt2eff}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2eff}(\mathrm{Eend2verpt}(b, edg)),\\ &\quad \mathrm{verpt2flo}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2flo}(\mathrm{Eend2verpt}(b, edg))\ \}\end{aligned} \tag{23}$$

and

$$\begin{aligned}&\mathrm{verpt2io}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2io}(\mathrm{Eend2verpt}(b, edg)) \Rightarrow\\ &\quad \mathrm{edge2dir}(edg) = \{\\ &\quad\quad \mathrm{verpt2eff}(\mathrm{Eend2verpt}(a, edg)) = -\,\mathrm{verpt2eff}(\mathrm{Eend2verpt}(b, edg)),\\ &\quad\quad \mathrm{verpt2flo}(\mathrm{Eend2verpt}(a, edg)) = \mathrm{verpt2flo}(\mathrm{Eend2verpt}(b, edg))\ \}\\ &\qquad\qquad \text{or, } \textit{the same with the minus sign between the flow variables}\end{aligned} \tag{24}$$

The dynamics specified by a PLBG is the family of solutions to the collection of constraints specified by $\mathrm{ver2defs}$ (and $\mathrm{ver2pars}$, $\mathrm{edge2dir}$).

[PLBG.10] A PLBG is a DPLBG (directed PLBG, as in the literature) iff for each $edg \in E$, only case (23) is relevant. In such cases, edges become harpoons (half-arrows), showing the direction of positive power flow. In any case, the edges are called **bonds**.

A consequence of a unidirectional convention for variables along edges is that it permits the use of directed (rather than undirected) graphs as the underlying formalism. Although this makes the handling of edge ends a little easier, the impediments to bottom up bond graph construction that it imposes dissuaded us from following this approach.

4 The Category of Bond Graph Patterns $\mathcal{BGP}att$

The formulation of abstract bond graphs in Section 3 was extremely operational. In this section we show how these details may be shepherded into a structure within which we can discern a category, $\mathcal{BGP}att$, of bond graph patterns and morphisms. The extent to which this can be used as a basis for bond graph transformation will be discussed at the end of this section and in the next. We start with a familiar caveat on graph isomorphism.

[GRISO] Combinatorial graphs (e.g. bond graphs) have vertices and edges. The vertices and edges, in themselves, have no properties save their own identity, unless endowed with properties using, e.g., maps, such as appear in **[PLBG.6]**-**[PLBG.10]**. Starting with a graph G, and then manipulating it in different ways, may result in technically non-identical graphs, even when the intention is to arrive at 'the same' result. The different results will be isomorphic, though not identical. Writing the needed isomorphisms explicitly gets very tedious, so we will use the phrase 'the same' below, to indicate that we are suppressing these details. Similar observations apply to 'a copy' of part of the RHS of a transformation rule during rule application.

[BGP.1] A vertex label **Any** is introduced. It is a one port label, with an anonymous port '$-$'.

Any does not correspond to any physical device, but serves to label the vertex at the end of an edge at the periphery of a pattern (defined below) that is to be matched to a vertex (of a bond graph which is to be transformed).

Since **Any**-labelled vertices do not correspond to physical devices, they do not need all the attributes of normal vertices. They just need attributes for dimensions and units, and directional indication. These are given by extending the domains of the maps $\mathrm{verpt2effDU}$, $\mathrm{verpt2floDU}$, $\mathrm{verpt2io}$, as needed, to the **Any**-labelled vertices.

[BGP.2] A **pattern** is a bond graph that contains zero or more **Any**-labelled vertices (along with their reduced set of attributes, as given in **[BGP.1]**).

Thus a bond graph is always a pattern, but a pattern is not a bond graph (i.e. a PLBG or a DPLBG) if it has one or more **Any**-labelled vertices.

[$\mathcal{BGP}att$.OBJ] The objects of the category $\mathcal{BGP}att$ are the patterns of **[BGP.2]**.

Let P be a pattern. We define $Anys_P$ to be the set of all **Any**-labelled vertices of P, and $nonAnys_P$ to be the set of all vertices of P other than **Any**-labelled vertices. We define AAs_P to be the set of all edges of P between two $Anys_P$ vertices, NAs_P to be the set of all edges of P between a $nonAnys_P$ vertex and an $Anys_P$ vertex, and NNs_P to be the set of all edges of P between two $nonAnys_P$ vertices.

[BGP.3] Let A and B be patterns. We use A and B subscripts to label the individual technical ingredients from Section 3 belonging to A and B. A **matching** m of A to B, written $m : A \to B$ consists of: a map $m_V : V_A \to V_B$ on vertices; and an injective map $m_E : E_A \rightarrowtail E_B$ on edges. From m_E, a further injective map m_{ends} on edge ends is derived. These are all required to satisfy injective and homomorphic conditions:

$$m_E, m_{\mathrm{ends}} \text{ are 1-1}, m_V \text{ need not be 1-1} \tag{25}$$

$$\mathrm{ends}(edg) = \{a, b\} \Rightarrow \mathrm{ends}(m_E(edg)) = \{m_V(a), m_V(b)\} \tag{26}$$

$$edg \text{ is incident on } a \Rightarrow m_{\mathrm{ends}}(a, edg) = (m_V(a), m_E(edg)) \tag{27}$$

The injectivity on edges reflects the fact that physical devices have fixed numbers of connections, which each need to be connected to something for the device to function.

We further require that the bond graph properties of $nonAnys_A$ vertices and edges of A are mirrored in B (i.e., labels, definitions (which we assume to include identity of free variables and parameters), ports (and their effort and flow variables, their dimensions and units, and their directional indicators)):

$$\mathrm{ver2lab}(a) \neq \textbf{Any} \Rightarrow$$

$$[\ \mathrm{ver2lab}_A(a) = \mathrm{ver2lab}_B(m_V(a))\ ] \wedge \tag{28}$$

$$[\ \mathrm{ver2defs}_A(a) = \mathrm{ver2defs}_B(m_V(a)\ ] \wedge \tag{29}$$

$$[\ \mathrm{ver2pts}_A(a)) = \mathrm{ver2pts}_B(m_V(a))\ ] \wedge \tag{30}$$

$$[\ pt \in \mathrm{ver2pts}_A(a) \Rightarrow$$

$$[\ \mathrm{verpt2effDU}_A(a, pt) = \mathrm{verpt2effDU}_B(m_V(a), pt)\ ] \wedge \tag{31}$$

$$[\ \mathrm{verpt2eff}_A(a, pt) = \mathrm{verpt2eff}_B(m_V(a), pt)\ ] \wedge \tag{32}$$

$$[\ \mathrm{verpt2floDU}_A(a, pt) = \mathrm{verpt2floDU}_B(m_V(a), pt)\ ] \wedge \tag{33}$$

$$[\ \mathrm{verpt2flo}_A(a, pt) = \mathrm{verpt2flo}_B(m_V(a), pt)\ ] \wedge \tag{34}$$

$$[\ \mathrm{verpt2io}_A(a, pt) = \mathrm{verpt2io}_B(m_V(a), pt)\ ]\ ] \tag{35}$$

We also require that the bond graph properties of $Anys_A$ vertices and edges of A are mirrored in B in line with their reduced attributes:

$$\begin{aligned}
&\mathrm{ver2lab}(a) = \textbf{Any} \Rightarrow \\
&\quad [\ \mathrm{verpt2effDU}_A(a,-) = \mathrm{verpt2effDU}_B(m_V(a), pt)\ ] \wedge && (36)\\
&\quad [\ \mathrm{verpt2floDU}_A(a,-) = \mathrm{verpt2floDU}_B(m_V(a), pt)\ ] \wedge && (37)\\
&\quad [\ \mathrm{verpt2io}_A(a,-) = \mathrm{verpt2io}_B(m_V(a), pt)\ ] && (38)\\
&\quad \text{where: } edg \text{ is incident on } a, \text{ and } \mathrm{Eend2verpt}(m_V(a), m_E(edg)) = (m_V(a), pt)
\end{aligned}$$

[$\mathcal{BGP}att$.MOR] The morphisms of the category $\mathcal{BGP}att$ are the matchings of **[BGP.3]**.

Theorem 1. *$\mathcal{BGP}att$ is a category, as claimed.*

Proof sketch: The morphisms of $\mathcal{BGP}att$ are conventional homomorphisms of labelled graphs, but restricted in a number of ways. In particular, edges and edge ends are mapped injectively. This means that only $Anys$ vertices may map many-1 (provided their edge ends map to distinct edge ends of the target). The various labelling attributes of vertices map identically for $nonAnys$ vertices, and for $Anys$ vertices the much smaller set of labelling attributes that matter, also map identically. So morphisms are injective in all respects save the $Anys$ vertex map.

Isomorphisms can thus be bijective homomorphisms of labelled graphs that preserve all the attributes, with obvious identities. Since morphisms are constructed from functions (on vertices and edges (and their ends)) and equalities of attributes, associativity follows immediately. □

We observe that a large number of the conditions (25)-(38) are actually independent from one another. This creates scope for defining many minor variants on the notion of morphism, by removing one or more of these conditions, provided that none of the conditions which remain, are dependent on the removed ones. Every such variant gives rise to a different category, even if they all share the same objects.

We further observe that while $\mathcal{BGP}att$ is a category of concrete graphs, we can easily create an analogous category of abstract graphs by taking its objects as those of $\mathcal{BGP}att$, *up to isomorphism* (where the isomorphisms are understood to be those of $\mathcal{BGP}att$), with the analogous adaptation of the morphisms.

4.1 Commentary

Having covered the essentials of the $\mathcal{BGP}att$ category above, we reflect on the 'design choices' made, with an eye to using these insights in the construction of a notion of rule based bond graph transformation in the next section.

In particular, the high degree of injectivity demanded of the morphisms deserves comment. It arises from the strong coupling between a vertex's label and its permitted set of edges, via the functional dependence of the number and characteristics of the ports associated with that label. Such strong constraints are needed because the vertices represent actual physical devices and the edges represent actual physical connections. Physical devices cannot acquire new physical connections or lose existing ones on a

whim, which would be possible if there were no coupling. This coupling between a vertex and its permitted set of edges (via its label) is in stark contrast to the usual situation in graph transformation formalisms, in which the properties of the two are decoupled, this being exploited during transformation. We see the impact of this below.

5 Bond Graph Transformation

In practice, bond graphs are often simplified, transformed, or rewritten, in various ways. In the existing literature, this activity is always described informally. In this section, we address the transformation of bond graphs from a more formal perspective, benefiting from the categorical formulation of the previous section.

5.1 Rule Based Bond Graph Transformation

In Section 3, **[PLBG.1]**-**[PLBG.10]** provided the mechanics for constructing bond graphs. This was achieved using maps that took each vertex to its various components, e.g. label, ports, etc. In this section, bond graphs are transformed by applying transformation **rule**s, which are templates for how an actual bond graph may be changed in the region of a **matching of the rule** to a **match**, or **redex**. This draws on the machinery of $\mathcal{BGP}att$ from Section 4.

Rule based graph transformation has an extensive literature. The procedure we will describe is adapted from the so-called double pushout and single pushout constructions comprehensively presented in [4] and [5] and in work cited therein, as well as in more recent publications. However, our approach will be considerably simpler, so we will not need most of the machinery discussed there. Partly, this is because we can work at a lower level of abstract technically, and partly it is because there are fundamental obstacles to applying the standard approaches *verbatim* in this application domain. We axiomatise what we need as follows.

[BGTR.1] A bond graph transformation rule (L, p, R) (**rule** for short) is given, firstly, by a pair of patterns L and R, where L is called the left hand side (LHS) and R is called the right hand side (RHS). Secondly, there is a bijection $p : Anys_L \rightarrowtail\!\!\!\!\!\rightarrow Anys_R$, for which (36)-(38) hold (in which the matching m is p and the port pt is the anonymous port of $p_V(a)$, as needed).

Lemma 1. *Let (L, p, R) be a rule. Then:*

1. *p extends to a bijection between edge ends incident on $Anys$ vertices.*
2. *For every AAs_L edge in L, either there is a unique AAs_R edge (where the respective ends correspond via p), or there are two NAs_R edges (where the $Anys_L$ and $Anys_R$ ends correspond via p).*
3. *For every AAs_R edge in R, either there is a unique AAs_L edge (where the respective ends correspond via p), or there are two NAs_L edges (where the $Anys_L$ and $Anys_R$ ends correspond via p).*
4. *There is a bijection between the $Anys_L$ vertices (and their incident edges/ends) that are not in the scope of either 1 or 2, and the $Anys_R$ vertices (and their incident edges/ends) that are not in the scope of either 1 or 2.*

Proof: This is a straightforward consequence of the bijective nature of p, and of the fact that $Anys$ vertices have a single edge end, extending p to a bijection between $Anys$ edge ends. The rest follows from the fact that each edge has exactly two ends. □

[BGTR.2] Let (L, p, R) be a rule. Let G be a bond graph, and $m : L \to G$ be a matching. The **application** of the rule (L, p, R) at the matching m is the result of the following steps:

1. Remove from G, all $m_E(E_L)$ edges, and all $m_V(nonAnys_L)$ vertices. This creates D (which will not be a valid bond graph, since, in general, it will have ports (belonging to $m_V(Anys_L)$ vertices) that do not correspond to edge ends in D).
2. Add to D:
   (a) a copy[2] of the vertices in $nonAnys_R$;
   (b) a copy of the edges in R;
   (c) If $edgC$ is the copy of an edge edg of R, and if edg has edge end (v, edg) in R, let $(vC, edgC)$ be the edge end corresponding to (v, edg) where:
      (i) if $v \in nonAnys_R$ then vC is its copy;
      (ii) if $v \in Anys_R$ then $vC = m_V(p^{-1}(v))$.

   Call the resulting graph H.
3. Endow H with all the needed attributes to make it a PLBG (labels, ports, variables, definitions, directional indicators) by: (a) retaining the existing attributes inherited via $m(L)$ for D; (b) replicating the attributes from the edges and $nonAnys_R$ vertices to their copies in H.

Below, to shortcut the rather inconvenient language of 2.(c), we will say that 'the NAs_L edges have their $nonAnys_L$ end **redirected**' (to their destinations in R), and 'the NAs_L *matched* edges have their $nonAnys_L$ *matched* end **redirected**' (to their destinations in H), or similar language.

Note that, by Lemma 1, the 'other end' of an AAs_L edge is also an $Anys_L$ incident edge. Therefore, unless such an edge is mapped by p to a similar AAs_R edge, such an edge is redirected to two edges in R, which is reflected in H. Analogously, two NAs_L edges may both be redirected to the same AAs_R edge, similarly reflected in H.

Remark We observe that the rule application described in **[BGTR.2]** definitely does not fall under the canonical framework of the double pushout approach (despite superficial appearances), since the entity D created in step 1 is not an object of $\mathcal{BGP}att$. There are two relatively self-evident reactions to this state of affairs. (1) Simply accept things as they are, namely, accept that in this case, stepping out of the category $\mathcal{BGP}att$ is necessary in order to define the required transformations. (2) Attempt to modify the definition of $\mathcal{BGP}att$ so that it can accommodate the intermediate entities D, thereby placing the construction back in the legitimate double pushout approach.

Neither option is entirely satisfactory. (1) has the evident defect that it steps outside the clean categorical framework of the double pushout approach. Nevertheless, as argued in Section 4.1, the category it uses, $\mathcal{BGP}att$, captures the right properties for

[2] When we say 'copy' of some graph theoretic concept, e.g., a set of vertices or a set of edges, we mean a distinct set of the same cardinality as the original, and endowed with attributes equivalent to the original, c.f. **[GRISO]**. In its turn, 'distinct' means having no element in common with any similar entity in the discourse.

the given application domain. (2) has the merit of remaining within the double pushout approach. But, in breaking the link between a vertex label with its set of ports on the one hand, and the set of edges incident on a vertex carrying that label on the other hand —as must happen if D is to be a legitimate category object— we fatally undermine the suitability of such a category to capture the properties needed in the application domain.

Another possibility suggests itself: (3) Attempt to modify the actual definition of graph transformation, by involving another category besides $\mathcal{BGP}att$ (e.g. a modification of $\mathcal{BGP}att$ as envisaged in (2)), thus arriving at a more complex, and novel, kind of transformation formalism. This is certainly possible, e.g. an approach via opfibrations [1], but this is well beyond the scope of the present paper.

Proposition 1. *The rule application described in* **[BGTR.2]** *yields a legitimate PLBG* H.

Proof sketch: By Lemma 1, p extends to a bijection between $Anys_L$ incident edge ends and $Anys_R$ incident edge ends. This can be composed at the L side with the injective m_{ends} and on the R side with the bijection between the corresponding $Anys_R$ incident edge ends and their copies in H. So there is a bijection between the images of the $Anys_L$ incident edge ends and the images of the $Anys_R$ incident edge ends. Since the rest of H is either pre-existing bond graph structure in D, or a copy of bond graph structure in R, the correctness of the combinatorial graph structure of H follows.

It remains to check that the dimensions, units, and directional indicators at the two ends of an edge in the PLBG H match suitably. They will do so in the part of D not affected by the removal of L. They will also do so in pattern R, a copy of which is added to D. This leaves the redirection mechanism to be checked. However, the constraints on matching, i.e. (36)-(38), and the analogues of those constraints demanded of p, ensure that like is replaced with like during the redirection, so that the needed properties hold.

The only remaining issue is the possible non-injectivity of m_V. However, all the properties we need are properties of edge ends, not of vertices, so the location of the edge ends is not germane, and any non-injectivity of m_V does not affect them. □

Beyond the preceding, there is evidently a matching $m^{\mathrm{R}} : R \to H$, together with a bijection $q : m_V(Anys_L) \rightarrowtail\!\!\!\!\!\rightarrow m^{\mathrm{R}}_V(Anys_R)$, with similar properties to p.

The bijectivity of p implies that for every rule (L, p, R), there is an inverse rule (R, p^{-1}, L). And existence of the matching m^{R} implies that for every application of (L, p, R) to G using m to yield H, to which R can be matched using m^{R}, there is an application of (R, p^{-1}, L) to H using m^{R} to yield G, to which L can be matched using m.

[SOUND] The rule application described in **[BGTR.2]** for (L, p, R) and m is **sound** iff the family of solutions to the PLBG H, restricted to the variables at the ports of $m^{\mathrm{R}}_V(Anys_R)$, is contained in the family of solutions to the PLBG G, restricted to the variables at the ports of $m_V(Anys_L)$.

[COMP] The rule application described in **[BGTR.2]** for (L, p, R) and m is **complete** iff the family of solutions to the PLBG G, restricted to the variables at the ports of $m_V(Anys_L)$, is contained in the family of solutions to the PLBG Hrestricted to the variables at the ports of $m^{\mathrm{R}}_V(Anys_R)$.

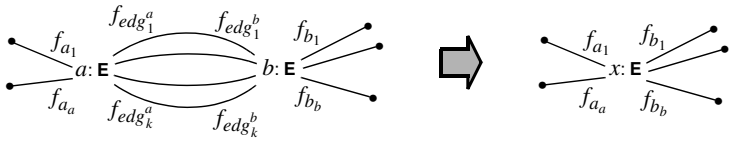


Fig. 2. A rule that shrinks a pattern whose $nonAnys$ form a two vertex connected **E**-graph. The • vertices are the **Any**-labelled vertices, with obvious bijection p.

[UPATH] A **path** in an undirected graph is a sequence of vertices, such that each consecutive pair is the pair of edge ends of an edge. If needed, the edges in question can be included in the path data.

[UCONN] A graph or pattern is **connected** iff for any two vertices, there is a path between them.

[E-GR F-GR EF-GR] A pattern is an **E**-graph iff all its $nonAnys$ vertices are **E**-labelled. Similarly for an **F**-graph. A pattern is an **EF**-graph iff each $nonAnys$ vertex is either **E**-labelled, or **F**-labelled.

Theorem 2. *Let (L, p, R) be a rule in which L is a connected* **E***-graph, and R is an* **E***-graph with a single $nonAnys$ vertex. Then any application of (L, p, R) is sound and complete. If L is not connected, then application is complete, but not necessarily sound. Similarly if L is an* **F***-graph.*

Proof: We consider the special case in which L is an **E**-graph with two $nonAnys$ vertices a, b, with one or more edges $\{edg_1 \ldots edg_k\}$ connecting them. See Fig. 2. Suppose a is connected to **Any**-labelled vertices $\{a_1 \ldots a_a\}$ via suitable edges, and b is connected to **Any**-labelled vertices $\{b_1 \ldots b_b\}$ via suitable edges. Then a will have flow variables $\{f_{edg_1^a} \ldots f_{edg_k^a}\ ,\ f_{a_1} \ldots f_{a_a}\}$ at its ports connected to its edges, and b will have flow variables $\{f_{edg_1^b} \ldots f_{edg_k^b}\ ,\ f_{b_1} \ldots f_{b_b}\}$ at its ports connected to its edges.

Assuming the standard directional indicators for **E**-devices, i.e., **in**, the behaviours at the two vertices matched to a, b in an application of (L, p, R) will be a copy of:

$$-f_{edg_1^a} - \ldots - f_{edg_k^a} = f_{a_1} + \ldots + f_{a_a} \tag{39}$$

$$-f_{edg_1^a} = f_{edg_1^b} \tag{40}$$

$$\ldots\ldots$$

$$-f_{edg_k^a} = f_{edg_k^b} \tag{41}$$

$$-f_{edg_1^b} - \ldots - f_{edg_k^b} = f_{b_1} + \ldots + f_{b_b} \tag{42}$$

Substituting (40)-(41) into (39), and adding the result to (42), yields:

$$0 = f_{a_1} + \ldots + f_{a_a} + f_{b_1} + \ldots + f_{b_b} \tag{43}$$

This, together with the constant effort condition, specifies the behaviour of G at the ports matched to the **Any** ports of L. But the same conditions result from replacing the vertices matched to a, b with a single vertex x say, and redirecting all edges incident on

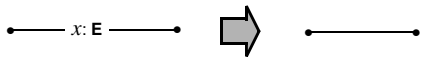


Fig. 3. A rule that shortcuts a single two port **E**-vertex.

the images of a, b to x. But this is what application of the rule (L, p, R) achieves. So application of (L, p, R) is complete.

But the procedure described can be reversed by considering the inverse rule (R, p^{-1}, L). If R is the LHS, its single $nonAnys$ vertex imposes a condition like (43). The flow variables in the right of (43) can then be partitioned into two sets, corresponding to the two $nonAnys$ vertices of L, and the incident edges can be redirected to these two vertices. Since the two vertices are connected by one or more edges in L, corresponding edges are created in the result PLBG, and a system of equations like (39)-(42) results, leading to soundness of application of (L, p, R).

If L is a connected **E**-graph with more than two $nonAnys$ vertices, then the edges between them can be contracted one at a time following the above procedure, except that some of the $\{f_{a_1} \ldots f_{a_a}\ ,\ f_{b_1} \ldots f_{b_b}\}$ flow variables will typically belong to ports to other $nonAnys$ vertices, rather than to $Anys$ vertices exclusively. This does not undermine the argument. The soundness and completeness properties of successive steps compose to give soundness and completeness for the application of (L, p, R) in its entirety.

If L is not connected, then following the above procedure will, at some point, fuse two vertices which are not connected by any edge. We will then have analogues of (39) and (42) (with 0 on the left), but none of (40)-(41). Absent $\{f_{edg_1^a} \ldots f_{edg_k^a}\ ,\ f_{edg_1^b} \ldots f_{edg_k^b}\}$, (39) and (42) imply (43), but not *vice versa*, leading to the failure of soundness. □

Corollary 1. *Let (L, p, R) be a rule where L has two* **Any***-labelled vertices and a single* **E***-labelled vertex, connected together by two edges, and where R has two* **Any***-labelled vertices only, and a single edge connecting them, i.e. a single AAs_R edge.*

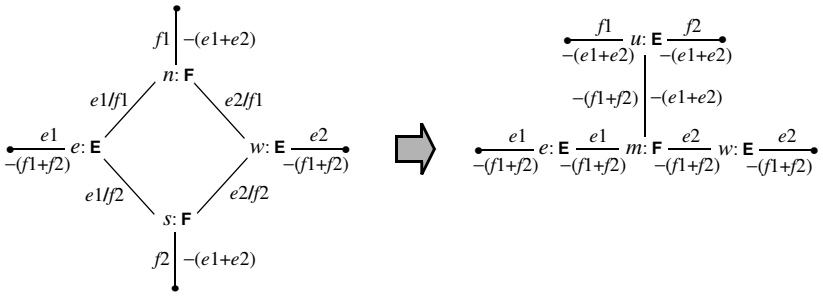


Fig. 4. A rule that optimises a shape that arises in bond graph construction from schematics.

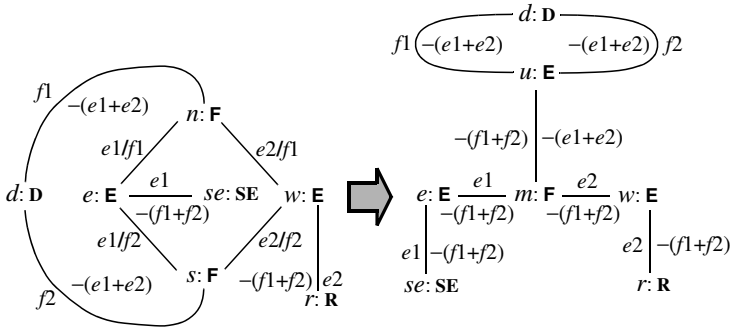


Fig. 5. An application of the rule in Fig. 4.

Then any application of (L, p, R) *is sound and complete. Similarly if* L *has a single* **F***-labelled vertex.*

Proof: Fig. 3 illustrates. This follows via a straightforward simplification of the previous arguments. □

Fig. 4 shows a rule (L, p, R) that simplifies a bond graph structure that can easily arise from the systematic construction of bond graphs from schematics. Only the values of the efforts and flows have been shown on the bonds, from which the bijection p can be easily inferred. For specific application, the rule needs to be completed with directional indicators, and appropriate (anti)equalities on the port effort and flow variables. Fig. 5 shows an application of the rule to a bond graph containing a two port device **D**. Soundness and completeness follow from applying the distributive law in various ways to a formula of the shape $(a + b)(c + d)$. Fig. 6 shows a more elaborate version of the same rule. These two examples have evident duals in which the roles of **E**-labelled vertices and **F**-labelled vertices are interchanged.

In various works in the bond graph literature there are *ad hoc* discussions of similar bond graph transformation rules (as we would term them) that can be safely applied

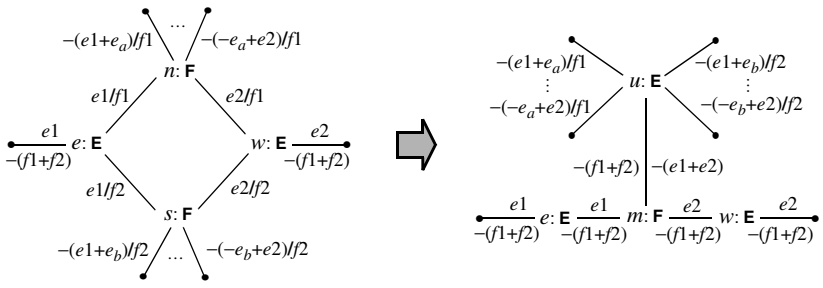


Fig. 6. A more elaborate version of the rule in Fig. 4.

to simplify bond graphs, when they have been arrived at by some systematic process starting from a schematic (which usually generates some redundancy). In [3] there is a more comprehensive list of such transformations. Reformulated as above, all of them can be proved sound and complete straightforwardly.

5.2 Bond Graph Transformation Confluence

When a number of rules are available for transforming a structure, e.g. a graph G, and more than one of their LHSs matches G, a question of some interest is whether 'the same' result will be produced if the rules are applied in one sequence or in a different sequence — or, speaking more technically, whether the set of rules is **confluent**. The following is an easy result guaranteeing confluence.

Theorem 3. *Let function* $\mathrm{labels}(X)$ *return the set of labels of the vertex set* X*. Let* $\mathcal{R} = \{Rl_1 \equiv (L_1, p_1, R_1), \ldots, Rl_n \equiv (L_n, p_n, R_n)\}$ *be a set of bond graph transformation rules, and let* G *be a bond graph. Then the application of* $\mathcal{R}$ *to* G *is confluent if:*

1. *for all* $1 \leq i \leq n$*, an* AAs *edge does not occur as a subgraph of any* L_i *or* R_i*;*
2. *for every* $1 \leq i < j \leq n$*, we have that* $\mathrm{labels}(nonAnys(L_i)) \cap \mathrm{labels}(nonAnys(L_j)) = \varnothing$*.*

Fig. 7 compactly shows two rules, $Rl1$ and $Rl2$, and a bond graph G. Rule $Rl1$'s LHS is shown, and the two dotted arrows indicate that vertex (labelled) **A1** is to be redirected to the lower **Any** vertex, and vertex **A2** is to be redirected to the upper **Any** vertex, so that the RHS of $Rl1$ is just a single AAs edge. Rule $Rl2$ is similar, but redirecting **B1** and **B2** to the **Any** vertices. If $Rl1$ is applied to G, bond graph H_1 results which is a single bond between **B1** and **B2**. There is a unique homomorphism from the LHS of $Rl2$ to H_1, but not a matching, because the edge map is not injective; both edges of the LHS of $Rl2$ have to map to the single edge in H_1. If we apply $Rl2$ first, we get H_2 which is a single bond between **A1** and **A2**, and we get no matching of $Rl1$'s LHS to it. Clearly there is no way of bringing H_1 and H_2 together with the rules we have.

The reasoning above was based on graph structure alone. This, unfortunately, is not enough to show the confluence of the system of rules discussed in Theorem 2 and Corollary 1 (including the analogues for **F**-labelled vertices). There is too much overlap between the labels occurring in the rules. To get confluence, we need to appeal, additionally, to the effort and flow constraints. With some work we can deduce:

Theorem 4. *The collection of rules discussed in Theorem 2 and Corollary 1, along with their* **F***-labelled analogues, is confluent and normalising.*

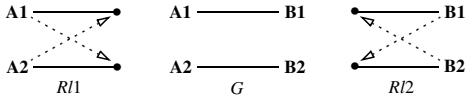


Fig. 7. An example showing the difficulties that arise when AAs edges are allowed to be either in, or created by, rule patterns.

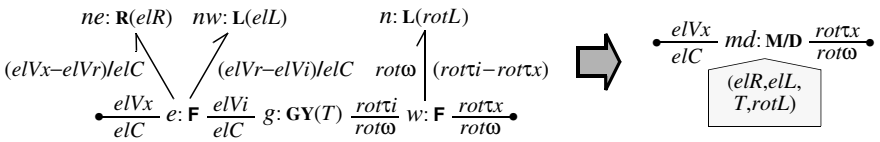


Fig. 8. An example abstraction, replacing the detail of an ideal but low level motor/dynamo with a high level abstract motor/dynamo device which conceals the details of its internal operation.

6 Bond Graph Abstraction and Refinement

The machinery we have developed for transforming bond graphs serves well for formally expressing hierarchy and abstraction. Parts of a large, low level bond graph can be formally abstracted by transformation to yield a smaller, more compact (and thus more perspicuous) one. Conversely, the reverse of the same process can formally expand a high level, abstract bond graph, refining it to a lower level of abstraction. The ingredient of our formalism, thus far unused, that enables us to do this effectively, is the user level device label.

We illustrate the process with a small idealised electric motor/generator example, adapted from [6]. Fig. 8 shows the details of an abstraction rule (L, p, R). On the left, the upward pointing harpoons in L show the positive power flow into the dissipator (electrical resistor) ne, with value elR, the positive power flow into the inertor (electrical inductance) nw with value elL, and the positive power flow into the inertor (rotational flywheel) n with value $rotL$. When the horizontal bonds are turned into right pointing harpoons, the bond graph becomes that of an ideal DC electrical motor, taking in power from the left (via external voltage $elVx$ and external current elC), with the voltage drop $elVx - elVi$ accounted for by the power absorbed by ne and nw. The remaining internal power $elVi \times elC$ goes into the gyrator g, which has transduction coefficient T. This outputs rotational power $rot\tau i \times rot\tau\omega$, according to $elVi = T \times rot\tau\omega$ and $T \times elC = rot\tau i$. There is a further loss of power due to the drop in torque $rot\tau i - rot\tau x$ caused by the flywheel, which finally results in rotational power $rot\tau x \times rot\tau\omega$ being output via the driveshaft. When the direction of the horizontal harpoons is reversed, the pattern can be reinterpreted as an ideal electrical dynamo powered from the right via the driveshaft. Formally, in L, all the port variables corresponding to the bond values shown in L are free, as are the parameters elR, elL, T, $rotL$, and the equations that govern the behaviour of the devices e, ne, nw, g, n.

In the RHS of the rule R, we see a single $nonAnys$ vertex md labelled with device **M/D**. The understanding would be that **M/D** is a user-introduced label for abstract ideal motors/dynamos in a given application context. The properties of **M/D** conceal and absorb the details of vertices e, ne, nw, g, n. This is done by aggregating the equations that govern the behaviour of the devices e, ne, nw, g, n, and existentially quantifying all their port variables, leaving the port variables corresponding to $elVx$, elC, $rot\tau x$, $rot\omega$ free. Note that the parameters elR, elL, T, $rotL$, now become parameters of **M/D**.

If, inside the quantified formula for the behaviour of an abstract device like **M/D**, the bound variables can be eliminated by some manipulation, then behaviour of **M/D**

will be given by equations in only the free port variables corresponding to $elVx$, elC, $rot\tau x$, $rot\tau\omega$ (and the parameters elR, elL, T, $rotL$). Usually though, this will not be possible.

Expressing a large complex system in terms of such high level components can bring convenience and perspicuity. Since our rules are reversible, the inverse process, refinement, is equally straightforward: we simply reverse the roles of L and R, and replace a high level abstraction with a more detailed implementation.

7 Conclusions

In the preceding sections we have presented, rather tersely, the essential elements of a formalisation of bond graphs. We covered the ingredients of the physical theories that are in scope, and the graphical structures that capture their interrelationships in a precise manner. Given these basics, the category $\mathcal{BGP}att$ could be formulated, and given that, the long history of graph transformation frameworks provided ample inspiration for formulating a suitable rule based framework for bond graph transformation. This could be immediately leveraged to give a methodology for abstraction and refinement of bond graphs.

One topic not touched on above, is the 'causality' concept of bond graphs. More than the topics we covered, this suffers from an extraordinary level of imprecision and ambiguity in the conventional bond graph literature, overwhelmingly attributable to lack of precision in the use of language. It will be disentangled and formalised in [2].

References

1. Banach, R.: Term Graph Rewriting and Garbage Collection Using Opfibrations. Theor. Comp. Sci. **131**, 29–94 (1994)
2. Banach, R., Baugh, J.: Bond Graphs: An Abstract Formulation (2023), *in preparation*.
3. Borutzky, W.: Bond Graph Methodology. Springer (2010)
4. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Lowe, M.: Alegbraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation (3 vols.). vol. 1, pp. 163–245. World Scientific (1997)
5. Ehrig, H., Heckel, R., Korff, M., Lowe, M., Ribeiro, L., Wagner, A., Corradini, A.: Alegbraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation (3 vols.). vol. 1, pp. 247–312. World Scientific (1997)
6. Karnopp, D., Margolis, D., Rosenberg, R.: System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems. Wiley, 5th edn. (2012)
7. Kypuros, J.: Dynamics and Control with Bond Graph Modeling. CRC (2013)
8. Paynter, H.: Analysis and Design of Engineering Systems. MIT Press (1961)
9. Paynter, H.: An Epistemic Prehistory of Bond Graphs. Elsevier (1992)