The 'Causality' Quagmire for Formalised 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. We extend the formalisation in preceding work to address the phenomenon of 'causality', intended to help formulate solution strategies for bond graphs, but usually presented in such vague terms that the claims made are easily shown to be false. We show that 'causality' only works as advertised in the simplest cases, where it mimics the mathematical definition of bond graph semantics. Counterexamples severely limit the applicability of the notion.

Keywords: Bond Graph · Formalisation · Bond Graph 'Causality' · Bond Graph Solution Strategy.

1 Introduction

Bond graphs were introduced in the work of Paynter in 1959 [16, 17]. These days the most authoritative presentation is [12]. From the large related literature we cite [4, 12, 14, 7, 18].

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 physical and engineering terminology feeling insecure and confused. In [3] we introduced a formalisation of bond graphs, allowing results to be presented with precision, and we discussed bond graph transformation, abstraction and refinement in that framework. In the present paper, we use the same framework to tackle so-called bond graph 'causality'. This has nothing to do with the normal physical notion of causality. Instead it is a game (in the formal mathematical sense of the word), for decorating a bond graph (usually in a locally progressive manner). The aim of this game is that its outcomes provide insight into how the equations of bond graph semantics (which are indeed equations, and thus undirected) may be transformed into a more algorithmically directed system, which can be used to actually solve for the behaviour of the system that the bond graph represents.

The bottom line is that the game only works in the most simple of cases, and then it just follows the semantics, interpreted in the simplest possible way. When the game doesn't work out that way, it indicates that something more sophisticated is needed,

though it gives little clue about *what* might be needed, without deeper (and often *ad hoc*) analysis. For such cases, more powerful generic techniques have been available for quite some time, and so, the case for the continued adherence to the 'causality' notion, with the enthusiasm seen in the textbook literature, is severely weakened.

The rest of this paper is as follows. Sections 2 and 3 recapitulate, tersely, the essentials of bond graphs as presented in [3]. Section 4 constructs bond graph 'causality', which we prefer to call dependency. Section 5 attempts to use the dependency notion to construct solutions to bond graph systems, but instead comes up with many counterexamples that show that the original idea doesn't work out. Lack of space forces many details to be omitted here. Section 6 presents two key theorems that explain why the 'nice' examples succeed, but the others don't. Section 7 indicates much more mathematically robust ways of approaching the solving of bond graphs, and Section 8 concludes.

2 Classical Physical Theories for Classical Engineering

We tersely summarise the physical theories of [3].

[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).

[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.

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.

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. PLBGs are assembled out of the following ingredients. Fig. 1 illustrates the process.

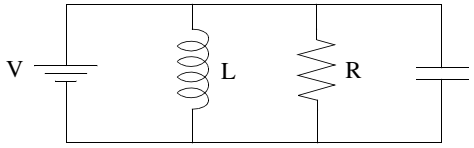
[PLBG.1] There is an alphabet $\mathcal{VL} = \mathcal{BVL} \cup \mathcal{CVL}$ of vertex labels, with basic vertex labels $\mathcal{BVL} = \{\mathbf{R}, \mathbf{C}, \mathbf{L}, \mathbf{SE}, \mathbf{SF}, \mathbf{TR}, \mathbf{GY}, \mathbf{E}, \mathbf{F}\}$, and user defined labels $\mathcal{CVL}$.

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

(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)

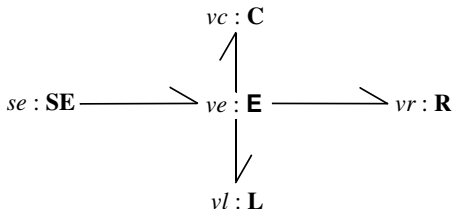


Fig. 1: Stages in bond graph construction: (a) a vertex label (for a dissipator); (b) adding a port; (c) adding a directional indicator; (d)-(f) assigning attributes (a)-(c) to a vertex vr; (g) vr's effort variable; (h) vr's flow variable; (i) vr's constitutive equation; (j) a simple elecrtical circuit 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 $\mathrm{labpt2stdio} : \mathcal{VL} \times \mathcal{PL} \nrightarrow \mathcal{IO}$. Details are formalised in [3].

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 $\mathrm{verpt2io} : V \times \mathcal{PL} \nrightarrow \mathcal{IO}$, and $\mathrm{verpt2io}(ver, pt)$ may, or may not, be the same as $\mathrm{verpt2stdio}(ver, pt)$. Details are formalised in [3].

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 $\mathrm{ver2defs} : V \rightarrow \mathrm{physdefs}$, 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. Additionally, the properties in $\mathrm{ver2defs}(ver)$ can depend on generic parameters (from a set $\mathcal{PP}$ say), so there is a map $\mathrm{ver2pars} : V \rightarrow \mathcal{PP}$. Additionally, the properties in $\mathrm{ver2defs}(ver)$ can depend on some bound variables. When necessary, we refer to such variables using $\mathbf{BV}(\mathrm{ver2defs}(ver))$. Details are formalised in [3].

[PLBG.8] There is a bijection $\mathrm{Eend2verpt} : V \times E \rightarrowtail V \times \mathcal{PL}$, from edge ends in BG, to port occurrences, and 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. Details are formalised in [3].

[PLBG.9] There is a map $\mathrm{edge2dir} : E \rightarrow \mathrm{physdir}$, where physdir is a set of equalities and antiequalities between effort and flow variables, recording whether the power flow conventions of the two devices at the ends of each edge are compatible or not, and if not, adjusting suitably. Details are formalised in [3].

[SEMANTICS] The dynamics specified by a PLBG is the family of solutions to the collection of constraints specified by ver2defs (and ver2pars, edge2dir).

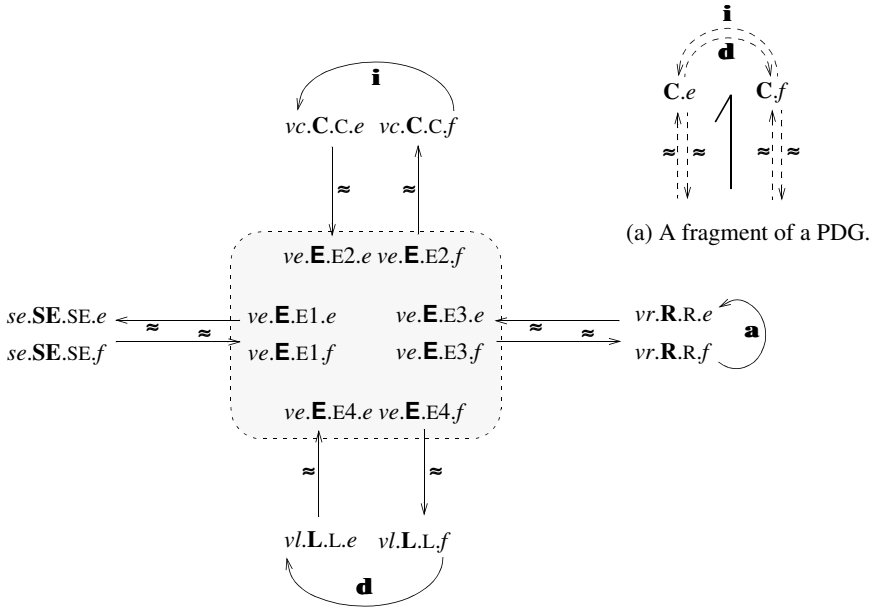
[PLBG.10] A PLBG is a DPLBG (directed PLBG, as in the literature) iff for each $edg \in E$, the power flow conventions are compatible. In such cases, edges become harpoons (half-arrows), showing the direction of positive power flow. In any case, the edges are called **bonds**.

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 Bond Graph 'Causality' / Dependency, and its Assignment

In the bond graph literature the word/s 'causality'/'causal' appear a lot. It is a most unfortunate choice of terminology, and the quotes we use indicate our dissatisfaction.

In the context of physical systems, the concept of causality has a well understood meaning. It is connected with the idea that certain physical phenomena or situations force others to be the case. We do not have the space to discuss how this works properly, but briefly, the 'motive force' that causes a physical state of affairs to persist, or that causes one physical state of affairs to change to another, is primarily thermodynamic. In the physical context (and physical systems of a particular kind, are after all, what bond graphs are for), the firmly based concept of causality is a topic for discussion in thermodynamics and statistical physics.



(a) A fragment of a PDG.

(b) One possible CDG.

Fig. 2: A PDG fragment and CDG option for Fig. 1.

By contrast, bond graph 'causality' is much more concerned with strategies for solving the system of equations that a bond graph defines. Worse, the connection between it and the necessities of actually solving these equation systems is tenuous, often to the point of invisibility. In this paper we prefer the word 'dependency', and in this section, we construct dependency, postponing thought about how it might be used till later.

Fig. 2a shows a fragment of a **pre-dependency graph** (**PDG**), for the bond graph of Fig. 1. Dependency concerns the variables of the bond graph, thus is easiest to depict using different, *directed* graphs, that follow the structure of the underlying bond graph. The idea is that the variable at the head of the arc *depends* (in some way not yet specified) on the variable at the tail of the arc. Since arcs are included in both directions for each variable, no commitment has yet been made as to what depends on what; thus it is a 'pre-' graph, indicated by making the arcs dashed. The $\approx$ label refers to the (anti-) equalities between device variables at the two ends of a bond. The **i** indicates that if the effort variable $\mathbf{C}.e$ depends on the flow variable $\mathbf{C}.f$, then **i**ntegration of $\mathbf{C}.f$ is needed (according to the constitutive equation (4)). If dependency is the other way round, then **d**ifferentiation of $\mathbf{C}.e$ is needed (again by (4)). Algebraic dependency (for dissipators) is indicated by **a**.

In a PDG, any two variables that are connected by an arc, are connected by two arcs, one in each direction, representing potential for, but indifference to, dependency between them. The indifference is removed in a **candidate dependency graph** (**CDG**), in which, for each such pair, one arc is deleted provided: (1) for each bond, the surviving arcs for effort and flow variables are oppositely directed; (2) the surviving arc for any

physical device is co-aligned with the effort and flow arcs on the bond that joins it to the rest of the bond graph. Fig. 2b shows a (somewhat perversely chosen) CDG for the Fig. 1 example. Solid arcs indicate commitment to a choice. All variables are shown in full detail. The only things missing are the device arcs for the **E** junction (all efforts strongly connected, similarly for flows) which would massively clutter the figure.

Note that we have not said what dependency *means*. The rest of this paper shows that this is not an easy question to answer.

Since all bond arcs come in oppositely directed pairs, any CDG can be represented more efficiently, by decorating the bonds. We use decorations ▶, ▷, ≻, which *always* point in the direction of the *effort arcs* of the underlying CDG, and work as follows.

Firstly, the decorations are determined by the devices at the ends of a bond, so there can be clashes. These are resolved by ▶ overriding ▷ overriding ≻, as needed. Then: **SE** has ▶ on its bond, pointing away from it; **SF** has ▶ on its bond, pointing towards it; **E** has one bond with ▷ pointing towards it (the **dominant bond**) and all other bonds (the **dependent bonds**) have ≻ pointing away from it; **F** has one bond with ▷ pointing away from it (the **dominant bond**) and all other bonds (the **dependent bonds**) have ≻ pointing towards it; **TR** has ≻ on its bonds, one towards and one away from it; **GY** has ≻ on its bonds, either both towards or both away from it; **L**, **C**, **R**, can have any decoration on their bond, but **L** *prefers* towards and **C** *prefers* away from (and **R** is indifferent). A CDG whose dependencies conform to the restrictions on direction implicit in the preceding is called a **normal dependency graph** (**NDG**), and a bond graph that encodes these dependency decorations on its bonds in the manner just described is called a **dependency bond graph** (**DBG**). It is clear that the CDG in Fig. 2b breaks all the rules that we have been imposing for an NDG. Fig. 3 shows the (only possible) DBG corresponding to Fig. 1. To make Fig. 2b into an NDG, we would have to reverse all the dependency arcs shown in the figure.

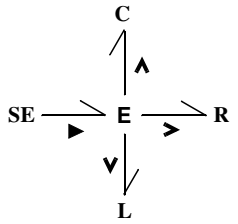


Fig. 3: A DBG for the example in Fig. 1.

The DBG definition just defined specifies how the DBG decorations should look. The traditional bond graph literature (e.g. [4, 12, 14, 7, 18]) presents what is termed a sequential 'causality' assignment procedure (SCAP) to achieve an acceptable decoration. In reality, this is a greedy algorithm executing a rule based system involving priorities for the assignment, that features, potentially, a lot of concurrency that is implicit in the original formulation. Fig. 4 summarises the rule based form.

It is clear that Fig. 4 is a greedy algorithm that depends on making choices at various points, and thus, for bond graphs with highly tangled junction structures, can involve a superpolynomial number of choice configurations to try, in order to determine whether a DBG set of decorations for a bond graph exists. In [15] the authors undertake a more incisive analysis, which shows that the job can be done in polynomial time.

5 Dependency Bond Graph Solution Strategies and Problems

The basic hypothesis of the traditional 'causality' approach of the bond graph literature, is that by decorating the bond graph with suitable markings, a strategy can be developed

Fig. 4: **Traditional Dependency Assignment Procedure**

Input: An undecorated bond graph
Output: A fully decorated bond graph, or **abort**

Pri := 1
Assign DBF decorations in order of priority Pri ;
If inconsistency found at any point **Then abort Fi**
1. Assign to *all* **SE** and **SF** bonds ; Pri := 2
2. Assign to *all* **E** and **F** *not fully assigned but with already assigned dominant bond* ; Pri := 3
3. Assign to *some* **L** or **C** *able to take its preferred direction* ;
If successful **Then** Pri := 2 **Else** Pri := 4 **Fi**
4. Assign to *some* **R**, **E**, **F**, **TR**, **GY** arbitrarily (but consistent with its definition) ; Pri := 2

that reduces the solution of system behaviour to a set of oriented explicit ODEs and/or assignments, involving a minimal set of variables.

It has to be said that the hypothesis fails, and the stated objective is unachievable.

The rule based process outlined in Fig. 4 was the main step of the first attempt to achieve the stated goal, and on benign examples it succeeds. But it was soon noticed that it does not succeed on all bond graphs. The rather extensive literature on the potential foundations of bond graphs that was produced in the '70s, '80s and '90s (surveyed comprehensively in [4]) does not result in establishing a procedure that always produces the desired outcome. Despite this, all the standard texts unfailingly discuss 'causality' in terms that raise no qualms about the relevance and validity of the notion, or about what caveats might need to hold for its validity.

The first thing to do is to show how the ideal outcome fails, which happens in many ways. Let us assume, for now, that dependency means that once the thing at the tail of a dependency arc is 'known', the thing at the head becomes 'knowable' too. To start with, we consider Fig. 2b, which is out of scope really, since it breaks all the rules. The most 'known' thing there is **SE**.f, since it depends on nothing else. But it is *not* known, since only **SE**.e is defined for an effort source, so we have to guess the **SE**.f value. Once guessed, we would have to guess how that flow is partitioned between **L**, **C**, **R**. Once guessed, we would solve algebraically for **R**, differentiate for **L**, integrate for **C**. Miraculously, we would come up with the same effort in each case at **E**, which agreed with the predefined value of **SE**.e. Of course it is crazy to try to solve the system this way, but it is not logically inconsistent, being simply an angelically derived solution.

Within the rules is Fig. 3, which reverses the dependencies. Now we start with the truly known **SE**.e, shared via **E** to each of **L**, **C**, **R**. This is integrated to solve for **L**, differentiated for **C**, scaled for **R**. Thus, sticking to the rules is some help here as the solution process is deterministic. There is one niggle though. While integration is used for **L**, in line with **L**'s preferred dependency, differentiation is used for **C**, in line with **C**'s *unpreferred* dependency. While this is no problem mathematically, differentiation is highly deprecated in engineering terms as it generates jolts and spikes in variables' behaviours. The 'wrong' dependency for **C** heralds

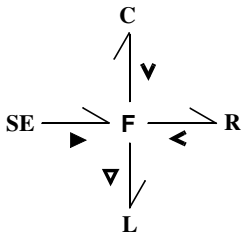


Fig. 5: A DBG for Fig. 1 with **F** instead of **E**.

this kind of issue, but we see that merely having a DBG for Fig. 1 does not guarantee the desired explicit ODE solution.

Fig. 5 replaces **E** with **F** in Fig. 1. This makes things more complicated. There are now three DBGs for Fig. 5 depending on which bond of **F** is chosen dominant. What is shown is the one that gives both **L** and **C** their preferred dependency. The traditional solution approach to more complex examples is **tracking back**, to derive the ODEs that solve the system. We start with the variable of an **L** or **C** that would appear in the LHS of an explicit ODE, equate it to the other side of the constitutive equation, and successively substitute for the variable there using the equations of the junction structure until source or physical variables are encountered. In **optimised tracking back** we do not stop at dissipators, but, recognising that their constitutive equations are linear algebraic, continue through those as if they were junctions. This causes **layer switching**, changing focus between efforts and flows, recognising that junction structures only couple efforts to efforts and flows to flows (**GY**s also layer switch).

Tracking back from $\mathbf{C}.e$ to $\mathbf{C}.f$ and then through the **equality layer** of **F** leads to $\mathbf{L}.f$ and to $\mathbf{C}.e' = \frac{1}{C}\,\mathbf{L}.f$. Tracking back from $\mathbf{L}.f$ to $\mathbf{L}.e$ and then through the **sum layer** of **F** is more complicated as it branches out to **R**, **C** and **SE**. Eventually we derive $\mathbf{L}.f' = \frac{1}{L}\,(\mathbf{SE}.e - \mathbf{C}.e - R\,\mathbf{L}.f)$. We have a coupled pair of first order ODEs, just as promised by traditional 'causality'. Solving them does however require the derivative of $\mathbf{SE}.e$ — **C** is 'too close' to an effort source for any jolt (e.g. switching on a constant non-zero effort) to be smoothed out by the system.

A surprising number of examples of awkward behaviour in bond graphs do not require the involvement of inertors or compliants — dissipators are quite capable of generating them without help. Fig. 6 shows a simple example of several dissipators connected to an effort source via a common effort junction. Using electrical language, we have resistors, composed in parallel. If we track back from the flow of the i'th dissipator, we get: $\mathbf{R}_i.f \times R_i = I_i R_i = V = \mathbf{R}_i.e = \mathbf{E}.e_i = \mathbf{E}.e_0 = \mathbf{SE}.e$, where $\mathbf{E}.e_0$ is the effort variable of **E** on the bond to **SE**. The choice of flow variable to track back from was crucial to arriving at the *known* variable of **SE** rather than the unknown one. Assuming the natural current orientations for the dependent variable $\mathbf{SE}.f$, given by: $\mathbf{SE}.f = \mathbf{E}.f_0 = \sum_{i=1}^{n} \mathbf{R}_i.f$ quickly leads to the familiar law $R^{-1} = \sum_{i=1}^{n} R_i^{-1}$, where R is the effective resistance of the parallel composition. Note that paying attention to the source unknown variable is indispensable for this derivation. Surprisingly, it is often stated in the bond graph literature that the unknown variables of sources are of no interest at all.

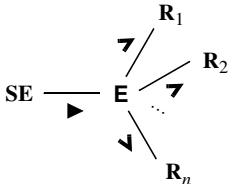


Fig. 6: Dissipators, composed in parallel.

Fig. 7 is the series counterpart of Fig. 6, i.e. several dissipators connected to an effort source via a common flow junction. In electrical language, we have resistors in series. The most immediate thing we notice is that the obvious symmetry of the physical system has been destroyed, in the bond graph representation, by the DBG dependency decorations. The most sensible arrangement would have been to have the dominant bond of **F** pointing towards **SE**. But since the decoration on the **SE** bond *must* be away from **SE**, and the **F** junction *must*

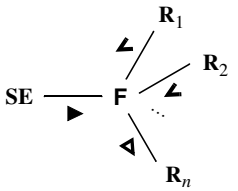


Fig. 7: Dissipators, composed in series.

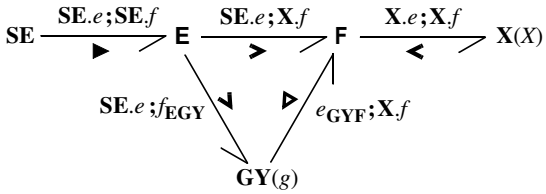


Fig. 8: A DBG showing tracking back to both known and unknown source variables.

have a dominant bond pointing outwards, one dissipator must acquire the dominant decoration of the **F** junction, and $\mathbf{R}_n$ has been chosen randomly. From a computational perspective, although the effort is assumed known from **SE**, the unique value of the flow shared by the whole system has to be derived by aggregating the relevant constitutive equations for the dissipators. The source of the difficulty is that, in the sum layer of a junction, even if the value on the dominant bond is known precisely, this only supplies partial information to the dependent bonds.

In more detail, tracking the dependencies back to the source from the flow on the n'th dissipator via its effort, we find: $\mathbf{R}_n.e = \mathbf{SE}.e - \sum_{i=1}^{n-1} \mathbf{R}_i.e$. Rearranging, we get: $\mathbf{SE}.e = \sum_{i=1}^{n} \mathbf{R}_i.e = \sum_{i=1}^{n} \mathbf{R}_i.f \times R_i = \mathbf{SE}.f \times \sum_{i=1}^{n} R_i$. After this, the value of the **SE** flow, and the familiar law for series composition $R = \sum_{i=1}^{n} R_i$, can be derived. For $i \neq n$, the DBG dependencies point in the opposite direction, but we have little option but to pursue the same calculational route. Similar issues arise if we replace parallel or series compositions of dissipators with compositions of inertors or of compliants, except that we have some integration or differentiation to do.

Fig. 8 shows an example in which the tracking back ansatz leads to both known and unknown variables of sources appearing as inhomogeneous terms in the differential equations that are generated. It is a given, though invariably unstated in the bond graph literature, that when we derive the requirement that a source variable is needed for the value of some physical variable, i.e. it appears as an inhomogeneous term in an explicit ODE for the variable, it is the *known* variable that appears and not the unknown one. But bond graphs do not guarantee that that is necessarily the case.

In Fig. 8, the first gyrator equation is $\mathbf{SE}.e = g\,\mathbf{X}.f$, which coincides with tracking back from variable $\mathbf{X}.f$ to known source $\mathbf{SE}.e$. Tracking back from $\mathbf{X}.e$ however, requires both gyrator equations and leads to $\mathbf{X}.e = \mathbf{SE}.e(1 - \frac{1}{g}) + g\,\mathbf{SE}.f$, in which both known $\mathbf{SE}.e$ and unknown $\mathbf{SE}.f$ appear.

What happens next depends on what **X** is. If **X** is a dissipator, then $\mathbf{X}.e = X\mathbf{X}.f = \frac{X}{g}\mathbf{SE}.e$, solving for the physical device by simple algebra. If **X** is a compliant, we have $\mathbf{X}.e' = \frac{1}{C}\mathbf{X}.f = \frac{X}{g}\mathbf{SE}.e$ so we just have to integrate $\mathbf{SE}.e$ to solve for the physical device. But if **X** is an inertor, we get:

$$\mathbf{X}.f' = \tfrac{1}{L}\mathbf{X}.e = \tfrac{1}{L}\big(\mathbf{SE}.e(1 - \tfrac{1}{g}) + g\,\mathbf{SE}.f\big) \tag{12}$$

which cannot be solved immediately, given the unknown $\mathbf{SE}.f$. This has to be reconciled with the first gyrator equation $\mathbf{SE}.e = g\,\mathbf{X}.f$, which yields $\mathbf{SE}.e' = g\,\mathbf{X}.f'$. $\mathbf{X}.f'$ can now be eliminated to solve for $\mathbf{SE}.f$ algebraically. What we have is a simple example of a non-trivial differential algebraic system.

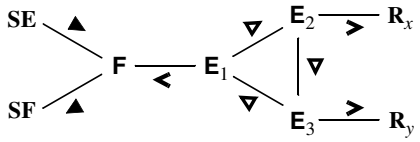


Fig. 9: A simple DBG showing dependency capture.

In some of the bond graph literature, tracking back can be interpreted as uncritically following the dependency decorations when there is a choice (in the equality layer of a junction). Fig. 9 shows the inherent risks. If we track the effort variable back from dissipator $\mathbf{R}_x$, we encounter $\mathbf{E}_2$, and following the dependency decorations, $\mathbf{E}_1$, $\mathbf{E}_3$, $\mathbf{E}_2$, and so on indefinitely — a case of dependency capture. This is particularly bad considering that the $\mathbf{E}_1$-$\mathbf{E}_2$-$\mathbf{E}_3$ triangle can be reduced to a single $\mathbf{E}$ junction, using the bond graph transformation theory of [3], which eliminates the endless tracking back loop, and does not change the physical behaviour of the system. Moreover, if we consider the $\mathbf{E}_1$-$\mathbf{E}_2$-$\mathbf{E}_3$ triangle as the base of a tetrahedron, adding another $\mathbf{E}$ junction as apex, connected to the base in the obvious way, we cannot even assign consistent NDG/DBG decorations to the resulting bond graph. This shows that existence and properties of dependency assignments are not invariant under bond graph transformations that are correct according to the standard correctness notion.

Fig. 10 shows a problem inherent in naively tracking back from physical variables on an individual basis. There are two dissipators, $\mathbf{R}_1$ and $\mathbf{R}_2$, both joined to an $\mathbf{E}$ junction, which is also joined to two further portions of the bond graph. The $\mathbf{E}$ junction is assumed to be a cut vertex of the graph. Tracking back from $\mathbf{R}_1$ may choose to go into 'the rest of the bond graph A'. This forces the equality of the two occurrences of e_1 shown. Tracking back from $\mathbf{R}_2$ may choose to go into 'the rest of the bond graph B'. This forces the equality of the two occurrences of e_2 shown. But the tracking back process itself does not force $e_1 = e_2$, which is required by the semantics, and will not be detected if the two tracking back instances terminate in A and B respectively. So naive tracking back can easily be unsound.

It is therefore important to supplement naive tracking back with a policy that ensures that *all* equalities implied by junction equality layer semantics are taken care of. The easiest way of ensuring this is to nominate a distinguished bond of each junction that is *always* tracked back through at each tracking back instance involving that junc-

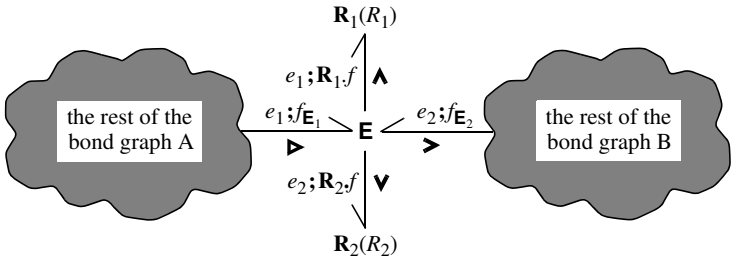


Fig. 10: A bond graph for which naive tracking back yields an incorrect solution.

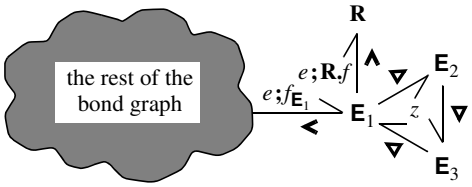


Fig. 11: An example for which tracking back yields equations different from those of the original bond graph, and following dominant bonds leads to nontermination.

tion's equality layer, *and also* insisting that *all* bonds of each junction get involved in equality layer tracking back. The obvious candidate for the distinguished bond role is the junction's dominant bond.

Fig. 11 shows an example, involving the earlier $\mathbf{E}_1$-$\mathbf{E}_2$-$\mathbf{E}_3$ triangle, in which the preceding considerations have some impact. Naive tracking back of efforts from **R** may explore 'the rest of the bond graph' and terminate there, and thereby miss catching some of the effort equalities of $\mathbf{E}_1$, and thereby those of $\mathbf{E}_2$ and $\mathbf{E}_3$. Exploring flows reaches further into the triangle and reveals that an unconstrained flow of arbitrary value may circulate round the triangle, but the details of the solution can differ from the standard semantics of the system.

On the other hand, if we follow the dominant bond tracking back policy, we get stuck in the triangle. Altogether, we have a choice between either bad semantics or nontermination.

6 Two Key Theorems

The preceding examples indicate that traditional bond graph 'causality' and the tracking back processes it is associated with are highly questionable notions, and placing naive faith in them is misplaced as far as the derivation of solutions to system behaviour is concerned. Nevertheless, the examples treated in the standard texts referred to in the Introduction seem to work well enough under this approach. This is a phenomenon that begs an explanation. In this section we present two theorems that go a long way to explaining the apparent dichotomy.

Theorem 1. *Let BG be a bond graph endowed with DBG dependency decorations. Suppose given a tracking back process for BG that always uses the dominant bond at junction equality layers. Then, for every physical device* **X** *for which tracking back encounters no gyrators:*

1. *If e is the effort variable of* **X** *and e is connected* either *to an* **E** *junction via a dependent bond of* **E**, or *to an* **F** *junction via the dominant bond of* **F**, *then the tracking back procedure tracks back only through bonds whose DBG dependency decorations point towards e.*
2. *If f is the flow variable of* **X** *and f is connected* either *to an* **F** *junction via a dependent bond of* **F**, or *to an* **E** *junction via the dominant bond of* **E**, *then the*

tracking back procedure tracks back only through bonds whose DBG dependency decorations point away from f.

In both of these cases, the bond mentioned may have one or more transformers interposed without affecting the result.

Theorem 1 highlights the essential reason why, under the right conditions, knowing something about a variable of device **X** permits knowledge about distant parts of the bond graph that might be of interest. This can be the only justification for entertaining notions like dependency decorations and the rules for their allocation — otherwise they serve no purpose. Theorem 2 embellishes Theorem 1 with additional contextual conditions that enable its use in deriving solution strategies for actual bond graphs.

Theorem 2. *Let BG be a bond graph endowed with DBG dependency decorations. Assume given a choice of core variables, consisting of one variable for every inertor or compliant, each with integral dependency. Assume each is tracked back through the device's constitutive equation and then through the equations of the bond graph, in an optimised and terminating process, until inertor or compliant or source variables are reached. Then, the process yields a set of explicit ODEs for the system, each of the form $v' = \Theta_{\mathbf{i}}(vs)$. The solution to these is semantically complete at peripheral variables, but need not be semantically sound at peripheral variables without the addition of algebraic equations.*

The structure of the proof of Theorem 1 reveals that the case analysis depends on: (1) whether an effort or a flow variable is being tracked; (2) whether the bond in question is dominant or not; (3) whether the bond is connected to an **E** junction or an **F** junction. So there are eight cases. Unfortunately, as we have seen, the theorem handles only four of them, leaving another four unconsidered. So the theorem far from exhausts the situations that are expressible using bond graphs, even when attention is restricted to bond graphs describing systems that are 'natural' from an engineering point of view. For bond graphs featuring these alternative cases, any connection with DBG dependency markings may appear tenuous indeed, and to a large extent, the plethora of awkward examples seen earlier can be seen to be rooted in these four cases.

In effect, Theorem 1 says that the nomenclature of 'following the causality', seen in the literature, is justified in the cases that the theorem handles, as it shows that the DBG decoration tree that is rooted at a physical device's variable, coincides with the parse tree that the bond graph's equations demand when discovering what expression the value of the relevant variable depends on, mathematically.

When tracking back enters the equality layer of a junction via a dependent bond b_{dep} and exits via the dominant bond b_{dom}, the traditional literature sometimes attempts to explain this by saying that the exit is via b_{dom} because 'b_{dep} is *caused by* b_{dom}'. Similar remarks can be seen when the sum layer is entered via the dominant bond and exited via all the dependent bonds. The limited remit of Theorem 1, and the impact of the preceding theory and counterexamples, show just how potentially unhelpful such phraseology can be.

We observe that the vast majority of textbook examples have tree shaped bond graphs, with a unique source device, and a tracking back process that overwhelmingly

uses the dominant bond wherever possible at equality layer variables. Moreover, gyrators are typically cut points of the bond graph, even in cases that are not tree shaped as a whole. So the conditions of Theorems 1 and 2 are often naturally fulfilled, thus leading to the good behaviour seen for 'natural', integral dependency examples. This all additionally aligns with the fact that some of the counterexamples explored earlier contained loops, which, in bond graphs, are always an indication that caution is required.

7 Robust Solution Strategies for Bond Graphs

The most straightforward (and negative) interpretation of the previous section is that dependency bond graphs (so called 'causal bond graphs') are a deeply questionable notion intrinsically,[2] and are of little use in determining system solution strategies in the general case. We observed that the traditional derivation of systems of equations of the typical form desired for solving bond graph systems was a surprisingly deterministic process, with any possible role for dependency decorations being confined to the choice element of tracking back at equality layers. In this section we briefly overview a system solution strategy approach that does not depend on dependency decorations at all, but rather treats the constituent equations of bond graphs that the standard semantics in **[SEMANTICS]** is defined with respect to, at face value, resulting in a system of differential algebraic equations (DAEs). It is worth observing that in general, no completely satisfactory strategy for arbitrary DAE systems is known. The following are standard references: [13, 11, 9, 10, 5, 8], of which [13] is the most pertinent for our purposes.

The approaches we outline can be applied to the raw bond graph equations given by the device definitions. This generates a large number of trivial linear equations. We can optimise to an extent, by substituting physical device variables into the junction structure as far as possible, taking care to preserve correctness by using **unification** rather than tracking back from individual variables. If the bond graph is tree structured, this will eliminate all junction variables. If not, residual junction variables will remain.

Like ODEs, DAEs can be classified into general systems, linear systems, and LCC systems. Less like ODEs, non-CC linear systems are much harder than LCC ones. The best treatment known to us of all the DAE cases is [13].

LCC DAE Systems The general structure studied in the LCC case is of the form $\boldsymbol{E}\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{x}$ is the vector of variables of interest, $\boldsymbol{x}'$ is the vector of its derivatives, $\boldsymbol{E}$ and $\boldsymbol{A}$ are matrices, and $\boldsymbol{b}$ is an inhomogeneous vector term. Evidently this covers all the LCC bond graph cases we have covered. The focus is then on the *matrix pencil* $\mathbf{M} \equiv (\lambda\boldsymbol{E} - \boldsymbol{A})$. The general approach reduces the pencil to Kronecker canonical form via matrix equivalence arguments, which covers all the singular cases. Of more interest practically is when $\boldsymbol{E}$ and $\boldsymbol{A}$ are square, whereupon, if $\det(\mathbf{M})(\lambda) \not\equiv 0$ the pencil is *regular*, and the Jordan normal form of $(\boldsymbol{A}-\lambda_0\boldsymbol{E})^{-1}\boldsymbol{E}$, where λ_0 is a value that makes the determinant nonzero, enables the pencil to be brought into Weierstrass canonical form. This form neatly decomposes the original system into a conventional inhomogeneous ODE problem, and the application of successive powers of a nilpotent operator to correspondingly higher derivatives of an inhomogeneous term, with the inhomogeneities derived from the arbitrary (but sufficiently smooth) $\boldsymbol{b}$. The nilpotency of

[2] We are by no means the first to say it; see e.g. [6], cited in [4].

the operator guarantees that the series has only a finite number of terms. It is the nilpotent part of the solution that brings in, for example, the derivative required of **SE**.e in the discussion of Fig. 5 above. It is clear that even in the regular LCC case, somewhat nontrivial mathematics is needed to handle all the exceptional circumstances that can be thrown up, and that our discourse on bond graphs had no difficulty in displaying.

In the case of Fig. 5 just mentioned, if **SE**.e is a step function that switches on a non-zero effort at $t = 0$, then $\boldsymbol{b}$ above is non-smooth, and, strictly speaking, we are outside the theory just described. Fortunately, there is a fairly straightforward extension of the smooth theory to generalised functions of a restricted form (piecewise smooth enough functions embellished with finite order derivatives of impulses at the join points (which themselves have no accumulations)). This enables such cases to be handled properly.

Linear Non-CC (LNCC) DAE Systems The equivalence classes that yield the classification used in the LCC case are based on conventional matrix similarity notions. Unfortunately, if coefficients are not constant, linear combination does not commute with differentiation. So the classification for LNCC cases is much more complicated than for LCC, as [13] shows (in stark contrast to pure ODE systems).

Nonlinear (NL) DAE Systems Aside from special cases that can be solved analytically, and that can always be created by reverse engineering the prospective solution, the only approach that is generally applicable for NL DAEs is numerical. The best known incarnation of this is the backward differentiation formula approach. In this, the derivative of the solution vector $\boldsymbol{x}'$ at time $n+1$ is approximated by a linear expression in $\boldsymbol{x}(n+1), \boldsymbol{x}(n) \ldots \boldsymbol{x}(n-k)$. The DAE system then yields a system of non-linear algebraic equations in $\boldsymbol{x}(n+1), \boldsymbol{x}(n) \ldots \boldsymbol{x}(n-k)$, which is solved using a non-linear approach such as the Newton-Raphson method, and the time index is incremented. For much more on approaches to the NL case see [1, 8], as well as [13] and elsewhere.

8 Conclusions

In the preceding sections we recalled, rather briefly, the essentials of formalised bond graphs, as per [3] (which made precise the more straightforward elements of bond graphs, as well as their rule based transformation). We then dived into the much murkier world of bond graph dependency (eschewing the highly misleading 'causality' word). Counterexamples proved ludicrously easy to find — once one overcame the inclination to accept the vague assertions regarding dependency's utility for solution derivation found in the bond graph textbook literature. This occupied a couple of sections above.

Nevertheless, the examples of bond graph 'causality' found in the bond graph textbook literature do seem to work. Why? In Section 6 we provided the answer. There is a case analysis for which half the cases assure 'nice behaviour' and the other half don't. By and large, the nice cases explain the nice textbook examples, and the other cases account for the plethora of trivially simple counterexamples we presented. To the best of our knowledge, no comparable result about 'causality' exists in the literature. We followed this by briefly covering mathematical approaches that deal robustly and reliably with the kind of equation systems that bond graphs generate. For lack of space, the whole of this paper is rather terse. A better illustrated and more detailed account of everything we have discussed is anticipated in [2].

References

1. Ascher, U., Petzold, L.: Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM (1998)
2. Banach, R.: Bond Graphs: An Abstract Formulation (2024), *in preparation*.
3. Banach, R., Baugh, J.: Formalisation, Abstraction and Refinement of Bond Graphs. In: Proc. ICGT-23. vol. 13961, pp. 145–162. Springer, LNCS (2023)
4. Borutzky, W.: Bond Graph Methodology. Springer (2010)
5. Brenan, K., Campbell, S., Petzold, L.: Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations. SIAM (1996)
6. Cellier, F., Otter, M., Elmqvist, H.: Bond Graph Modeling of Variable Structure Systems. In: Proc. ICBGM-95. Simulation Series, vol. 27, pp. 49–55. SCS Publishing (1995)
7. Damic, V., Montgomery, J.: Mechatronics by Bond Graphs. Springer (2015)
8. Gerdts, M.: Optimal Control of ODEs and DAEs. De Gruyter (2012)
9. Hairer, E., Norsett, S., Wanner, G.: Solving Ordinary Differential Equations I Nonstiff Problems. Springer (1993)
10. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II Stiff and Differential-Algebraic Problems. Springer (1996)
11. Ilchmann, A., Reis (Eds.), T.: Surveys in Differential-Algebraic Equations (3 vols.). Springer (2013, 2015)
12. Karnopp, D., Margolis, D., Rosenberg, R.: System Dynamics: Modeling, Simulation, and Control of Mechatronic Systems. Wiley, 5th edn. (2012)
13. Kunkel, P., Mehrmann, V.: Differential-Algebraic Equations: Analysis and Numerical Solution. European Mathematical Society (2006)
14. Kypuros, J.: Dynamics and Control with Bond Graph Modeling. CRC (2013)
15. Lamb, J., Woodall, D., Asher, G.: Bond Graphs II: Causality and Singularity. Discrete Appl. Math. **73**, 143–173 (1997), following on from Bond Graphs I, in vol. **72**, 261–293 (1997).
16. Paynter, H.: Analysis and Design of Engineering Systems. MIT Press (1961)
17. Paynter, H.: An Epistemic Prehistory of Bond Graphs. Elsevier (1992)
18. Tenreiro Machado, J., Cunha, V.: An Introduction to Bond Graph Modeling with Applications. CRC (2021)

A Two Proofs

Proof of Theorem 1: To start with, we assume no transformer is encountered during tracking back (as well as no gyrator). For case 1 of the theorem, if e is connected to an **E** junction via a dependent bond $b_{\mathbf{X}}$ of **E**, then the DBG dependency decoration on $b_{\mathbf{X}}$ points towards **X**, thus towards e. Because BG has DBG decorations, the dominant bond, $b_{\mathbf{E}}$ of **E** points the same way, and the tracking back strategy selects it as the bond to track back along. The device at the other end of $b_{\mathbf{E}}$ is either a peripheral device, in which case we are done, or another **E** junction for which $b_{\mathbf{E}}$ is a dependent bond, or an **F** junction, for which $b_{\mathbf{E}}$ is the dominant bond, pointing away from **F**.

If it is an **E** junction with $b_{\mathbf{E}}$ a dependent bond, the argument just given repeats. If it is an **F** junction with $b_{\mathbf{E}}$ the dominant bond, then all other bonds of **F** are dependent bonds, and have DBG dependency decorations pointing towards **F**, and thus towards **X** again. The branching in the tracking back process that takes place at **F** is therefore all along dependent bonds. Let $b_{\mathbf{F}}$ be such a dependent bond of **F**. The device at the other end of $b_{\mathbf{F}}$ is either a peripheral device, in which case we are done, or another **F** junction

for which $b_{\mathbf{F}}$ is the dominant bond, or another **E** junction, for which $b_{\mathbf{F}}$ is a dependent bond, pointing away from the **E** junction. The latter two cases have been dealt with already, so the arguments now repeat as often as needed until the entire parse tree that *SeekDependencyExp*(e) visits has been treated.

If, instead, e is connected to an **F** junction via a dominant bond of **F**, then the argument is as in the preceding paragraph, so need not be repeated. The argument for case 2 of the theorem is dual.

The general case chooses to interpose one or more transformers into one or more of the bonds processed during tracking back. For a given bond b, say there are n transformers. The process replaces b and its DBG dependency decoration, by $n+1$ bonds, $b_0, b_1, \ldots b_n$ interleaving the n transformers, each having a DBG dependency decoration in the same direction as the one on b. Each one at the extreme ends of the sequence, b_0 and b_n, may become a dominant bond if the junction it attaches to demands it according to its direction. We are done. □

Proof of Theorem 2: It is assumed that each intertor or compliant has integral dependency. In that case, the derivative of its core variable v is proportional to its complementary variable, and the tracking back from the complementary variable is covered by one of the four cases of Theorem 1.

So tracking back from an intertor or compliant variable v will align with the dependency markings until one of four situations is encountered. (1) A source or drain is encountered. Then, because the tracking back aligns with the dependency markings, and the DBG dependency markings point away from an **SE** or **DE** device and towards an **SF** or **DF** device, the variable encountered will be the effort variable for an **SE** or **DE** and will be the flow variable for an **SF** or **DF**, i.e. it is the known variable that is encountered, which will then appear as a conventional inhomogeneous term in the RHS(s) of relevant explicit ODE(s). (2) A core variable w of an intertor or compliant device is encountered. This terminates that branch of the tracking back process. Variable w is the LHS variable of the ODE for that device, and so w will occur in the RHS of the explicit ODE equation for v. (3) A dissipator is encountered. This would terminate that branch of an unoptimised process, but an optimised process continues via the complementary layer. This reverses the direction of tracking of the DBG markings, but since the continuation happens in the complementary layer, it simply interchanges the clause 1 and clause 2 cases of Theorem 1, and tracking back again becomes aligned with the DBG markings. (4) A gyrator is encountered. This is similar to the dissipator case except that continuation is along the other bond of the gyrator, rather than reversing back along the same bond after the layer switch. Since each instance of tracking back from a core variable terminates by assumption, one of the four cases discussed applies for each tracking back path in the parse tree of v. Since the derived ODEs are clearly satisfied by any behaviour of the system described by BG, we have semantic completeness. But, and not least because the statement of the theorem is applicable to pure dissipator bond graphs, we do not necessarily have semantic soundness without considering further algebraic equations (even if we disregard unknown variables of sources). We are done. □