

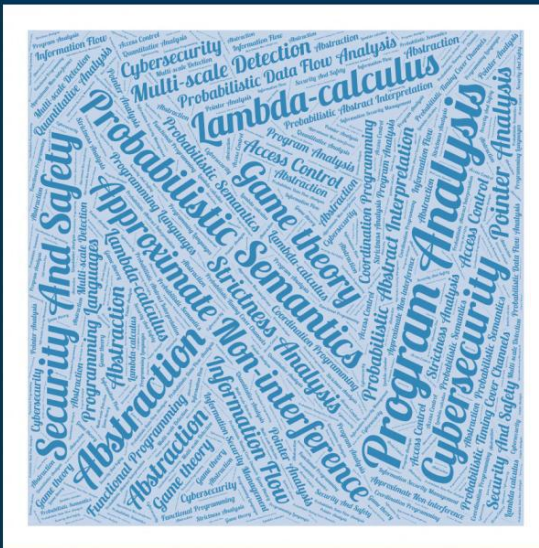
Festschrift

LNCS 12065


Alessandra Di Pierro
Pasquale Malacaria
Rajagopal Nagarajan (Eds.)

From Lambda Calculus to Cybersecurity Through Program Analysis

Essays Dedicated to Chris Hankin
on the Occasion of His Retirement



Editors

Alessandra Di Piero 
University of Verona
Verona, Italy

Rajagopal Nagarajan
Department of Computer Science
Middlesex University
London, UK

Pasquale Malacaria
School of Electronic Engineering
and Computer Science
Queen Mary University of London
London, UK

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-030-41102-2 ISBN 978-3-030-41103-9 (eBook)
<https://doi.org/10.1007/978-3-030-41103-9>

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Cover illustration: Chris Hankin

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

14. Fielder, A., Panaousis, E.A., Malacaria, P., Hankin, C., Smeraldi, F.: Decision support approaches for cyber security investment. *Decision Support Systems* **86**, 13–23 (2016)
15. Garasi, C.J., Drake, R.R., Collins, J., Picco, R., Hankin, B.E.: The MEDEA experiment - can you accelerate simulation-based learning by combining information visualization and interaction design principles? In: VISIGRAPP (3: IVAPP). pp. 299–304. SciTePress (2017)
16. Hankin, C.: An Introduction to Lambda Calculi for Computer Scientists. Texts in computing, Kings College (2004), <https://books.google.co.uk/books?id=kzdmQgAACAAJ>
17. Hankin, C.: Abstract interpretation of term graph rewriting systems. In: Functional Programming. pp. 54–65. Workshops in Computing, Springer (1990)
18. Hankin, C.: Lambda Calculi: A Guide, Handbook of Philosophical Logic. Handbook of Philosophical Logic, vol. 15, pp. 1–66. Springer, Dordrecht (2011)
19. Hankin, C.: Game theory and industrial control systems. In: Semantics, Logics, and Calculi. Lecture Notes in Computer Science, vol. 9560, pp. 178–190. Springer (2016)
20. Hankin, C., Burn, G.L., Peyton Jones, S.L.: A safe approach to parallel combinator reduction (extended abstract). In: ESOP. Lecture Notes in Computer Science, vol. 213, pp. 99–110. Springer (1986)
21. Hankin, C., Burn, G.L., Peyton Jones, S.L.: A safe approach to parallel combinator reduction. *Theor. Comput. Sci.* **56**, 17–36 (1988)
22. Hankin, C., Métayer, D.L.: Deriving algorithms from type inference systems: Application to strictness analysis. In: POPL. pp. 202–212. ACM Press (1994)
23. Hankin, C., Métayer, D.L.: Lazy type inference for the strictness analysis of lists. In: ESOP. Lecture Notes in Computer Science, vol. 788, pp. 257–271. Springer (1994)
24. Hankin, C., Métayer, D.L.: A type-based framework for program analysis. In: SAS. Lecture Notes in Computer Science, vol. 864, pp. 380–394. Springer (1994)
25. Hunt, S., Hankin, C.: Fixed points and frontiers: A new perspective. *J. Funct. Program.* **1**(1), 91–120 (1991)
26. Khouzani, M.H.R., Malacaria, P., Hankin, C., Fielder, A., Smeraldi, F.: Efficient numerical frameworks for multi-objective cyber security planning. In: ESORICS (2). Lecture Notes in Computer Science, vol. 9879, pp. 179–197. Springer (2016)
27. Li, T., Hankin, C.: Effective defence against zero-day exploits using bayesian networks. In: CRITIS. Lecture Notes in Computer Science, vol. 10242, pp. 123–136. Springer (2016)
28. Malacaria, P., Hankin, C.: Generalised flowcharts and games. In: ICALP. Lecture Notes in Computer Science, vol. 1443, pp. 363–374. Springer (1998)
29. Malacaria, P., Hankin, C.: A new approach to control flow analysis. In: CC. Lecture Notes in Computer Science, vol. 1383, pp. 95–108. Springer (1998)
30. Martin, C., Hankin, C.: Finding fixed points in finite lattices. In: FPCA. Lecture Notes in Computer Science, vol. 274, pp. 426–445. Springer (1987)
31. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag, Berlin, Heidelberg (1999)
32. Simmie, D.S., Vigliotti, M.G., Hankin, C.: Ranking twitter influence by combining network centrality and influence observables in an evolutionary model. *J. Complex Networks* **2**(4), 495–517 (2014)
33. Thapen, N.A., Simmie, D.S., Hankin, C.: The early bird catches the term: combining twitter and news data for event detection and situational awareness. *J. Biomedical Semantics* **7**, 61 (2016)

Organization

Additional Reviewers

Martin Berger	University of Sussex, UK
Chiara Bodei	Università di Pisa, Italy
Marco Comini	Università di Udine, Italy
Andrew Fielder	Imperial College, UK
Simon Gay	University of Glasgow, UK
Sebastian Hunt	City University, UK
Martin Lester	University of Reading, UK
Andrea Masini	Università di Verona, Italy
Flemming Nielson	Technical University of Denmark, Denmark
Federica Paci	Univresità di Verona, Italy
Helmut Seidl	Technische Universität München, Germany
David Sands	Chalmers University of Technology, Sweden
Nikos Tzevelekos	Queen Mary University of London, UK
Herbert Wiklicki	Imperial College, UK
Enea Zaffanella	Università di Parma, Italy

Contents

Logic

Cables, Trains and Types	3
<i>Simon J. Gay</i>	
Cathoristic Logic: A Logic for Capturing Inferences Between Atomic Sentences	17
<i>Richard Evans and Martin Berger</i>	
A Type Theory for Probabilistic λ -calculus	86
<i>Alessandra Di Pierro</i>	

Program Analysis

Galois Connections for Recursive Types	105
<i>Ahmad Salim Al-Sibahi, Thomas Jensen, Rasmus Ejlers Møgelberg, and Andrzej Wąsowski</i>	
Incremental Abstract Interpretation	132
<i>Helmut Seidl, Julian Erhard, and Ralf Vogler</i>	
Correctly Slicing Extended Finite State Machines	149
<i>Torben Amtoft, Kelly Androutsopoulos, and David Clark</i>	

Security

Secure Guarded Commands	201
<i>Flemming Nielson and Hanne Riis Nielson</i>	
Modelling the Impact of Threat Intelligence on Advanced Persistent Threat Using Games	216
<i>Andrew Fielder</i>	
Security Metrics at Work on the Things in IoT Systems	233
<i>Chiara Bodei, Pierpaolo Degano, Gian-Luigi Ferrari, and Letterio Galletta</i>	
New Program Abstractions for Privacy	256
<i>Sebastian Hunt and David Sands</i>	

Optimizing Investments in Cyber Hygiene for Protecting Healthcare Users. . . 268
*Sakshyam Panda, Emmanouil Panaousis, George Loukas,
and Christos Laoudias*

Author Index 293

Logic



Cables, Trains and Types

Simon J. Gay^(✉)

School of Computing Science, University of Glasgow, Glasgow, UK
Simon.Gay@glasgow.ac.uk

Abstract. Many concepts of computing science can be illustrated in ways that do not require programming. *CS Unplugged* is a well-known resource for that purpose. However, the examples in *CS Unplugged* and elsewhere focus on topics such as algorithmics, cryptography, logic and data representation, to the neglect of topics in programming language foundations, such as semantics and type theory.

This paper begins to redress the balance by illustrating the principles of static type systems in two non-programming scenarios where there are physical constraints on forming connections between components. The first scenario involves serial cables and the ways in which they can be connected. The second example involves model railway layouts and the ways in which they can be constructed from individual pieces of track. In both cases, the physical constraints can be viewed as a type system, such that typable systems satisfy desirable semantic properties.

1 Introduction

There is increasing interest in introducing key concepts of computing science in a way that does not require writing programs. A good example is *CS Unplugged* [2], which provides resources for paper-based classroom activities that illustrate topics such as algorithmics, cryptography, digital logic and data representation. However, most initiatives of this kind focus on “Theoretical Computer Science Track A” [4] topics (algorithms and complexity), rather than “Track B” topics (logic, semantics and theory of programming). To the extent that logic is covered, the focus is on gates and circuits rather than deduction and proof.

In the present paper, we tackle Track B by describing two non-programming scenarios illustrating the principles of static type systems. The first scenario involves serial cables, and defines a type system in which the type of a cable corresponds to the nature of its connectors. The physical design of the connectors enforces the type system, and this guarantees that the semantics (electrical connectivity) of a composite cable is determined by its type.

The second scenario is based on model railway layouts, where there is a desirable runtime safety property that if trains start running in the same direction, there can never be a head-on collision. Again, the physical design of the pieces

Supported by the UK EPSRC grant EP/K034413/1, “From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)”.

© Springer Nature Switzerland AG 2020
A. Di Pierro et al. (Eds.): Festschrift Hankin, LNCS 12065, pp. 3–16, 2020.
https://doi.org/10.1007/978-3-030-41103-9_1

of track enforces a type system that guarantees runtime safety. The situation here is more complicated than for serial cables, and we can also discuss the way in which typability is only an approximation of runtime safety.

We partially formalise the cables example, in order to define a denotational semantics of cables and prove a theorem about the correspondence between types and semantics. A fully formal treatment would require more machinery, of the kind that is familiar from the literature on semantics and type systems, but including it all here would distract from the key ideas. We treat the railway example even less formally; again, it would be possible to develop a more formal account.

I am only aware of one other non-technical illustration of concepts from programming language foundations, which is Victor’s *Alligator Eggs* [12] presentation of untyped λ -calculus. When I have presented the cables and trains material in seminars, audiences have found it novel and enjoyable. I hope that these examples might encourage other such scenarios to be observed—and there may be a possibility of developing them into activities along the lines of *CS Unplugged*.



Fig. 1. A serial cable with 25-pin female (left) and male (right) connectors.

2 Cables and Types

The first example involves serial cables. These were widely used to connect computers to peripherals or other computers, typically using the RS-232 protocol, until the emergence of the USB standard in the late 1990s. Figure 1 shows a serial cable with 25-pin connectors, and illustrates the key point that there are two polarities of connector, conventionally called *male* and *female*. Figure 2 shows a serial cable with 9-pin connectors, both female. The physical design is such

that two connectors can be plugged together if and only if they are of different male/female polarity and have the same number of pins. From now on we will ignore the distinction between 9-pin and 25-pin connectors, and assume that we are working with a particular choice of size of connector.

For our purposes, the interesting aspect of a serial cable is that it contains two wires for data transmission. These run between the *send* (SND) and *receive* (RCV) pins of the connectors. There are other wires for various power and control signals, but we will ignore them.

There are two ways of connecting the send/receive wires. If SND is connected to SND and RCV is connected to RCV, then the cable is called a *straight through* cable (Fig. 3). This is just an extension cable. Alternatively, if SND is connected to RCV and RCV is connected to SND, then the cable enables two devices to communicate because the SND of one is connected to the RCV of the other. This is called a *null modem* cable (Fig. 4).



Fig. 2. A serial cable with 9-pin female connectors.

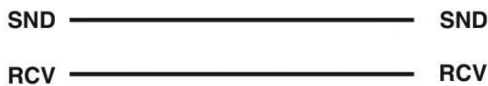


Fig. 3. A straight through cable.



Fig. 4. A null modem cable.

With two ways of wiring SND/RCV, and three possible pairs of polarities for the connectors, there are six possible structures for a serial cable. They have

different properties in terms of their electrical connectivity and their physical pluggability. When choosing a cable with which to connect two devices, clearly it is important to have the correct connectors and the correct wiring. Because the wiring of a cable is invisible, there is a conventional correspondence between the choice of connectors and the choice of wiring.

- A straight through cable has different connectors at its two ends: one male, one female.
- A null modem cable has the same connectors at its two ends: both male, or both female.

It is easy to convince oneself that this convention is preserved when cables are plugged together to form longer cables. By thinking of the electrical connectivity of a cable as its semantics, and the nature of its connectors as its type, we can see the wiring convention as an example of a type system that guarantees a semantic property. In the rest of this section, we will sketch a formalisation of this observation.

Figure 5 gives the definitions that we need. Syntactically, a *Cable* is either one of the fundamental cables or is formed by plugging two cables together via the \cdot operator. The fundamental cables are the straight through cable, **straight**, and two forms of null modem cable, **null₁** and **null₂**. Recalling that a null modem cable has the same type of connector at both ends, the forms **null₁** and **null₂** represent cables with two male connectors and two female connectors. It doesn't matter which cable is male-male and which one is female-female.

To define the type system, we use the notation of classical linear logic [8]. Specifically, we use *linear negation* $(-)^{\perp}$ to represent complementarity of connectors, and we use *par* (\wp) as the connective that combines the types of connectors into a type for a cable. This is a special case of a more general approach to using classical linear logic to specify typed connections between components [6]. We use \mathbb{B} to represent one type of connector, and then \mathbb{B}^{\perp} represents the other type. As usual, negation is involutive, so that $(\mathbb{B}^{\perp})^{\perp} = \mathbb{B}$. The notation \mathbb{B} is natural because we will use boolean values as the corresponding semantic domain. It doesn't matter whether \mathbb{B} is male or female, as long as we treat it consistently with our interpretation of **null₁** and **null₂**. The typing rule **PLUG**, which is a special case of the *cut* rule from classical linear logic, specifies that cables can be plugged together on complementary connectors. In this rule, A , B and C can each be either \mathbb{B} or \mathbb{B}^{\perp} .

Example 1. The cable **straight** \cdot **straight** represents two straight through cables connected together. It is typable by

$$\frac{\text{straight} : \mathbb{B} \wp \mathbb{B}^{\perp} \quad \text{straight} : \mathbb{B} \wp \mathbb{B}^{\perp}}{\text{straight} \cdot \text{straight} : \mathbb{B} \wp \mathbb{B}^{\perp}} \text{ PLUG}$$

This composite cable has the same type as a single straight through cable, and we will see that it also has the same semantics.

Syntax	
$Cable ::= \text{straight} \mid \text{null}_1 \mid \text{null}_2 \mid Cable \cdot Cable$	cables
$A, B, C ::= \mathbb{B} \mid \mathbb{B}^\perp$	types
Type equivalence	
$(\mathbb{B}^\perp)^\perp = \mathbb{B}$	
Typing rules	
$\text{straight} : \mathbb{B} \wp \mathbb{B}^\perp$	$\text{null}_1 : \mathbb{B} \wp \mathbb{B}$
$\text{null}_2 : \mathbb{B}^\perp \wp \mathbb{B}^\perp$	
$\frac{c : A \wp B \quad d : B^\perp \wp C}{c \cdot d : A \wp C} \text{ PLUG}$	
Semantics	
Writing $\mathbb{B}^{[\perp]}$ to represent either \mathbb{B} or \mathbb{B}^\perp , the denotational semantics of $c : \mathbb{B}^{[\perp]} \wp \mathbb{B}^{[\perp]}$ is	
$\llbracket c \rrbracket \subseteq \{\text{true}, \text{false}\} \times \{\text{true}, \text{false}\}$	
defined inductively on the syntactic construction of c by:	
$\llbracket \text{straight} \rrbracket = \{(\text{false}, \text{false}), (\text{true}, \text{true})\}$	identity, id
$\llbracket \text{null}_1 \rrbracket = \{(\text{false}, \text{true}), (\text{true}, \text{false})\}$	inversion, inv
$\llbracket \text{null}_2 \rrbracket = \{(\text{false}, \text{true}), (\text{true}, \text{false})\}$	inversion, inv
$\llbracket c \cdot d \rrbracket = \llbracket c \rrbracket \circ \llbracket d \rrbracket$	relational composition

Fig. 5. Formalisation of cables.

Example 2. The cable $\text{null}_1 \cdot \text{null}_2$ is two null modem cables connected together, which will also be semantically equivalent to a straight through cable. It is typable by

$$\frac{\text{null}_1 : \mathbb{B} \wp \mathbb{B} \quad \text{null}_2 : \mathbb{B}^\perp \wp \mathbb{B}^\perp}{\text{null}_1 \cdot \text{null}_2 : \mathbb{B} \wp \mathbb{B}^\perp} \text{ PLUG}$$

Example 3. The cable $\text{straight} \cdot \text{null}_1$ is a null modem cable extended by plugging it into a straight through cable. Semantically it is still a null modem cable. It is typable by

$$\frac{\text{straight} : \mathbb{B} \wp \mathbb{B}^\perp \quad \text{null}_1 : \mathbb{B} \wp \mathbb{B}}{\text{straight} \cdot \text{null}_1 : \mathbb{B} \wp \mathbb{B}} \text{ PLUG}$$

To complete the formalisation of the syntax and type system, we would need some additional assumptions, at least including commutativity of \wp so that we

can flip a straight through cable end-to-end to give $\text{straight} : \mathbb{B}^\perp \wp \mathbb{B}$. However, the present level of detail is enough for our current purposes.

We define a denotational semantics of cables, to capture the electrical connectivity. We interpret both \mathbb{B} and \mathbb{B}^\perp as $\{\text{true}, \text{false}\}$ so that we can interpret a straight through cable as the identity function and a null modem cable as logical inversion. Following the framework of classical linear logic, we work with relations rather than functions. Plugging cables corresponds to relational composition.

Example 4. Calculating the semantics of the cables in Examples 1–3 (for clarity, including the type within $\llbracket - \rrbracket$) gives

$$\llbracket \text{straight} \cdot \text{straight} : \mathbb{B} \wp \mathbb{B}^\perp \rrbracket = \text{id} \circ \text{id} = \text{id} = \llbracket \text{straight} \rrbracket$$

$$\llbracket \text{null}_1 \cdot \text{null}_2 : \mathbb{B} \wp \mathbb{B}^\perp \rrbracket = \text{inv} \circ \text{inv} = \text{id} = \llbracket \text{straight} \rrbracket$$

$$\llbracket \text{straight} \cdot \text{null}_1 : \mathbb{B} \wp \mathbb{B} \rrbracket = \text{id} \circ \text{inv} = \text{inv} = \llbracket \text{null}_1 \rrbracket$$

This illustrates the correspondence between the type of a cable and its semantics.

The following result is straightforward to prove.

Theorem 1. *Let A be either \mathbb{B} or \mathbb{B}^\perp and let c be a cable.*

1. *If $c : A \wp A$ then $\llbracket c \rrbracket = \text{inv}$.*
2. *If $c : A \wp A^\perp$ then $\llbracket c \rrbracket = \text{id}$.*

Proof. By induction on the typing derivation, using the fact that $\text{inv} \circ \text{inv} = \text{id}$. \square

This analysis of cables and their connectors has several features of the use of static type systems in programming languages. The semantics of a cable is its electrical connectivity, which determines how it behaves when used to connect devices. The type of a cable is a combination of the polarities of its connectors. There are some basic cables, which are assigned types in a way that establishes a relationship between typing and semantics. The physical properties of connectors enforce a simple local rule for plugging cables together. The result of obeying this rule is a global correctness property: for *every* cable, the semantics is characterised by the type.

It is possible, physically, to construct a cable that doesn't obey the typing rules, by removing a connector and soldering on a complementary one. For example, connecting $\text{straight} : \mathbb{B} \wp \mathbb{B}^\perp$ and $\text{straight} : \mathbb{B}^\perp \wp \mathbb{B}$, by illegally joining \mathbb{B}^\perp to \mathbb{B}^\perp , gives a straight through cable with connectors $\mathbb{B} \wp \mathbb{B}$. Such cables are available as manufactured components, called *gender changers*. Usually they are very short straight through cables, essentially two connectors directly connected back to back, with male-male or female-female connections. They are like type casts: sometimes useful, but dangerous in general. If we have a cable that has been constructed from fundamental cables and gender changers, and if we can't

see exactly which components have been used, then the only way to verify that its connectors match its semantics is to do an electrical connectivity test—i.e. a runtime type check.

Typically, a programming language type system gives a *safe approximation* to correctness. Every typable program should be safe, but usually the converse is not true: there are safe but untypable programs. Cable gender changers are not typable, so the following typing derivation is not valid.

$$\frac{\frac{\text{untypable}}{\text{changer}_1 : \mathbb{B} \wp \mathbb{B}} \quad \frac{\text{untypable}}{\text{changer}_2 : \mathbb{B}^\perp \wp \mathbb{B}^\perp}}{\text{changer}_1 \cdot \text{changer}_2 : \mathbb{B} \wp \mathbb{B}^\perp} \text{ PLUG}$$

However, the semantics is defined independently of typing, and

$$\begin{aligned} \llbracket \text{changer}_1 \cdot \text{changer}_2 \rrbracket &= \llbracket \text{changer}_1 \rrbracket \circ \llbracket \text{changer}_2 \rrbracket \\ &= \text{id} \circ \text{id} \\ &= \text{id} \end{aligned}$$

so that the typing $\text{changer}_1 \cdot \text{changer}_2 : \mathbb{B} \wp \mathbb{B}^\perp$ is consistent with Theorem 1.

3 Trains and Types

The second example of a static type system is based on model railway layouts. Specifically, the simple kind that are aimed at young children [1, 5], rather than the elaborate kind for railway enthusiasts [3]. The examples in this paper were constructed using a “Thomas the Tank Engine” [7] set.

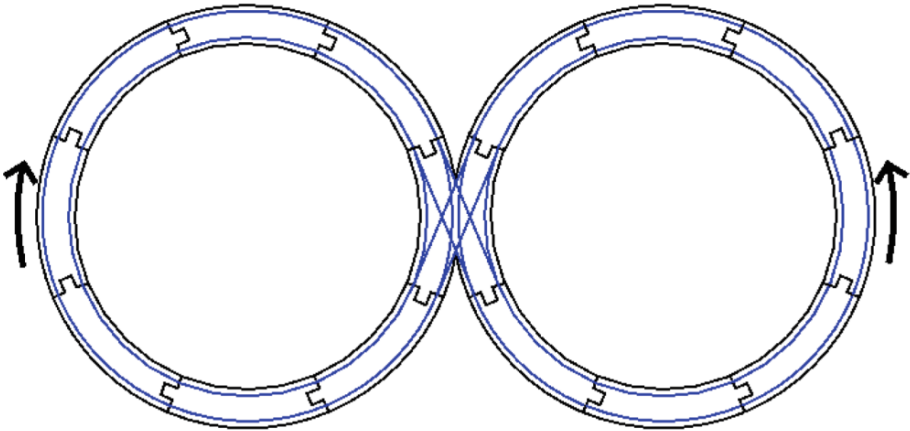


Fig. 6. A figure eight layout. (Color figure online)

Figure 6 shows a simple figure eight layout consisting of two circles linked by a crossover piece. The blue lines (coloured in the electronic version of the paper) show the guides for the train wheels—in these simple sets, they are grooves rather than raised rails. Notice that there are multiple pathways through the crossover piece. It would be possible for a train to run continuously around one of the circles, but in practice the tendency to follow a straight path means that it always transfers through the crossover piece to the other circle.

It’s clear from the diagram that when a train runs on this layout, it runs along each section of track in a consistent direction. If it runs clockwise in the left circle, then it runs anticlockwise in the right circle, and this never changes. Consequently, if two trains run simultaneously on the track, both of them in the correct consistent direction, there can never be a head-on collision. For example, if one train starts clockwise in the left circle, and the other train starts anticlockwise in the right circle, they can never move in opposite directions within the same circle. They might side-swipe each other by entering the crossover section with bad timing, or a faster train might rear-end a slower train, but we will ignore these possibilities and focus on the absence of head-on collisions as the safety property that we want to guarantee.

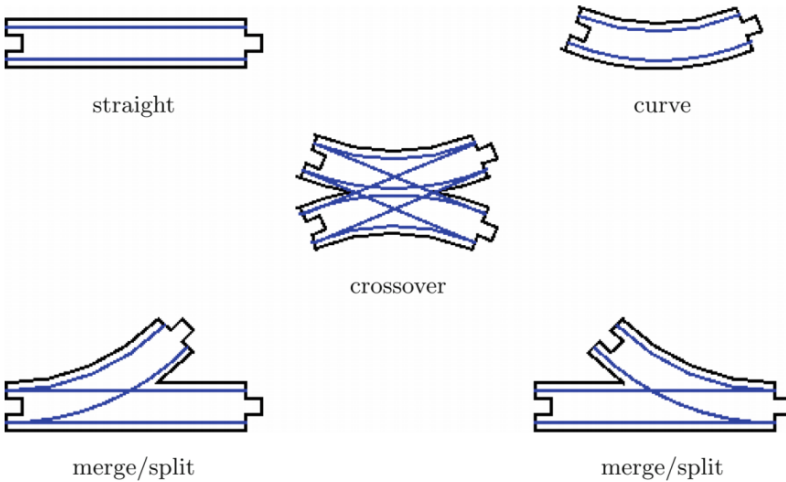


Fig. 7. Basic track pieces.

Figure 7 shows a collection of basic track pieces. They can be rotated and reflected (the pieces are double-sided, with grooves on the top and bottom), which equivalently means that the merge/split pieces (bottom row) can be used with inverted connectors. When a merge/split piece is used as a split (i.e. a train enters at the single endpoint and can take either the straight or curved branch), there is a lever that can be set to determine the choice of branch. We will ignore this feature, because we are interested in the safety of layouts under the assumption that any physically possible route can be taken.

The pieces in Fig. 7 can be used to construct the figure eight layout (Fig. 6) as well as more elaborate layouts such as the one in Fig. 8. It is easy to see that the layout of Fig. 8 has the same “no head-on collisions” property as the figure eight layout.

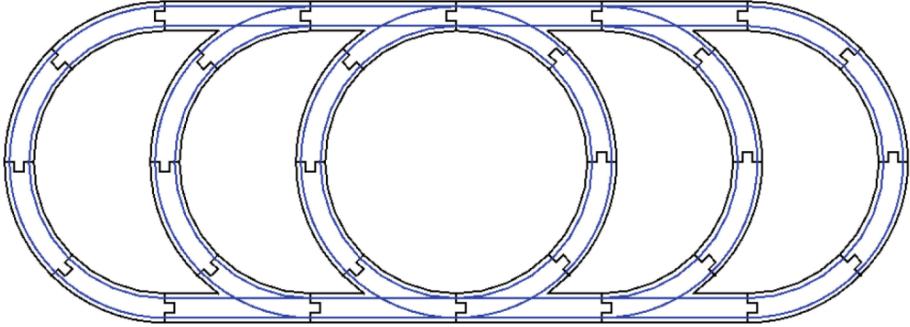


Fig. 8. A layout with multiple paths.

Each track piece has a number of endpoints, where it can be connected to other pieces. We will refer to each endpoint as either positive (the protruding connector) or negative (the hole). The pieces in Fig. 7 have the property that if a train enters from a negative endpoint, it must leave from a positive endpoint. This property is preserved inductively when track pieces are joined together, and also when a closed (no unconnected endpoints) layout is formed. This inductively-preserved invariant is the essence of reasoning with a type system, if we consider the type of a track piece or layout to be the collection of polarities of its endpoints. If we imagine an arrow from negative to positive endpoints in each piece, the whole layout is oriented so that there are never two arrowheads pointing towards each other. This is exactly the “no head-on collisions” property. It is possible to use the same argument in the opposite direction, with trains running from positive to negative endpoints, to safely orient the layout in the opposite sense.

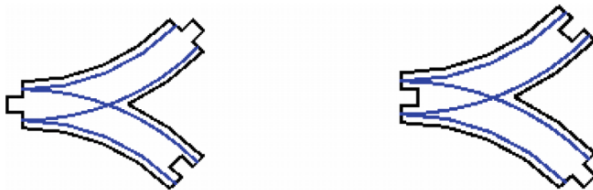


Fig. 9. The Y pieces.

This argument could be formalised by defining a syntax for track layouts in the language of traced monoidal categories [9, 11] or compact closed categories [6, 10] and associating a directed graph with every track piece and layout.

The track pieces in Fig. 7 are not the only ones. Figure 9 shows the Y pieces, which violate the property that trains run consistently from negative to positive endpoints or *vice versa*. They can be used to construct layouts in which head-on collisions are possible. In the layout in Fig. 10, a train can run in either direction around either loop, and independently of that choice, it traverses the central straight section in both directions.

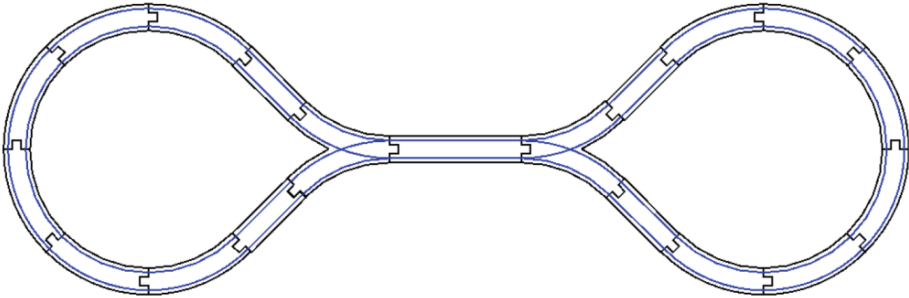


Fig. 10. An unsafe layout using Y pieces.

It is possible to build safe layouts that contain Y pieces. Joining two Y pieces as in Fig. 11 gives a structure that is similar to the crossover piece (Fig. 7) except that the polarities of the endpoints are different. This “Y crossover” can be used as the basis for a safe figure eight (Fig. 12). However, safety of this layout cannot be proved by using the type system. If a train runs clockwise in the circle on the right, following the direction from negative to positive endpoints, then its anticlockwise journey around the circle on the left goes against the polarities. To prove safety of this layout, we can introduce the concept of logical polarities, which can be different from the physical polarities. In the circle on the left, assign logical polarities so that the protruding connectors are negative and the holes are positive, and then the original proof works.

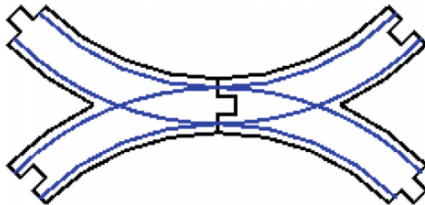


Fig. 11. Joining Y pieces to form a crossover.

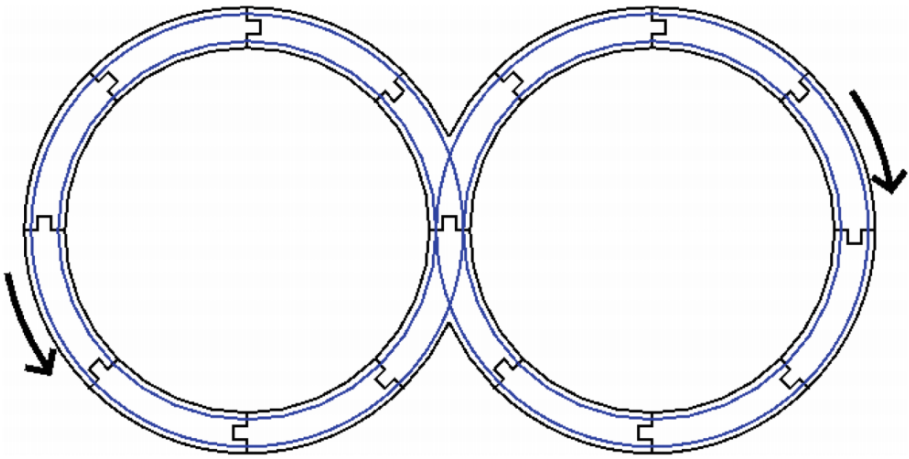


Fig. 12. A safe figure eight using Y pieces. In the circle on the right, the direction of travel follows the physical polarity, but in the circle on the left, the direction of travel is against the physical polarity. To prove safety, assign logical polarities in the circle on the left, which are opposite to the physical polarities.

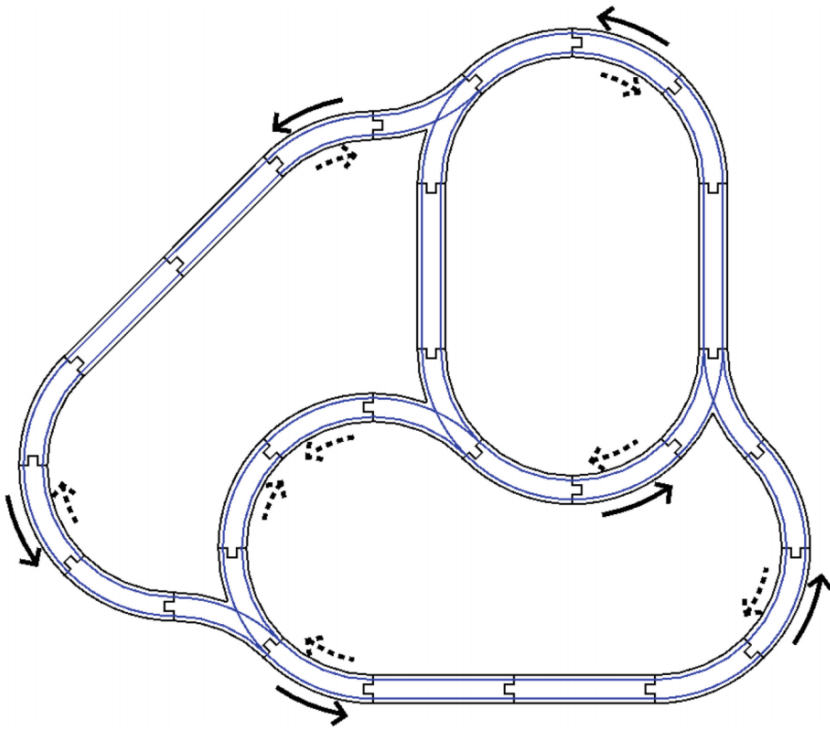


Fig. 13. A layout using Y pieces that is safe in one direction (solid arrows) but not the other (dashed arrows).

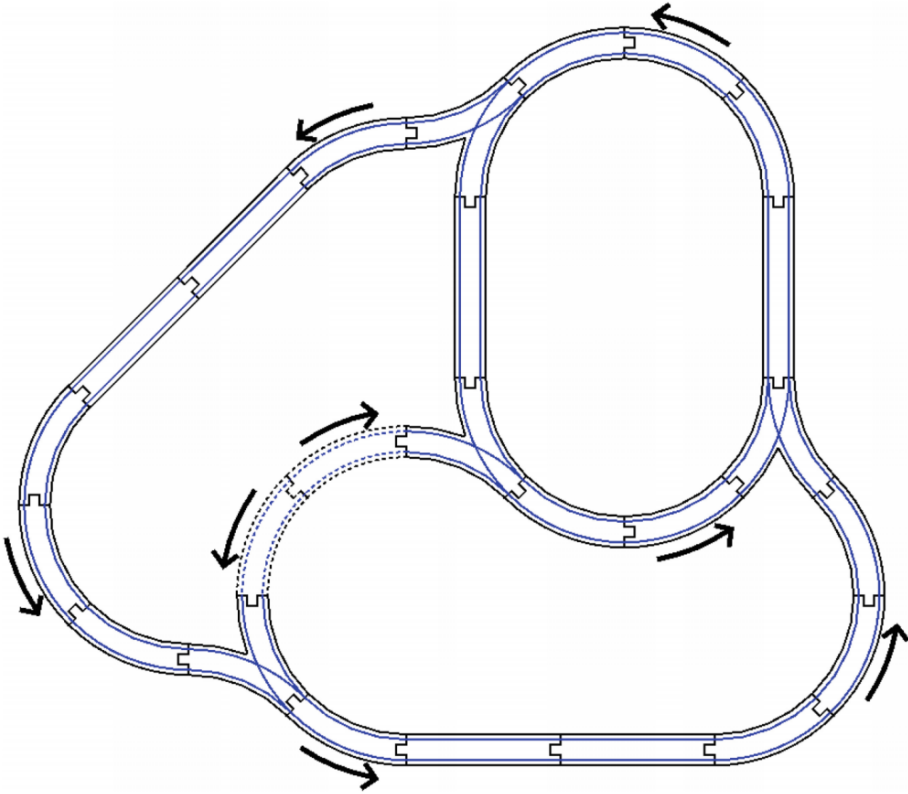


Fig. 14. This layout is safe for travel in the direction of the arrows, because the dashed section of track is unreachable. However, the divergent arrows in the dashed section mean that logical polarities cannot be used to prove safety.

A more exotic layout is shown in Fig. 13. This layout is safe for one direction of travel (anticlockwise around the perimeter and the upper right loop) but unsafe in the other direction. More precisely, if a train starts moving clockwise around the perimeter, there is a path that takes it away from the perimeter and then back to the perimeter but moving anticlockwise, so that it could collide with another clockwise train.

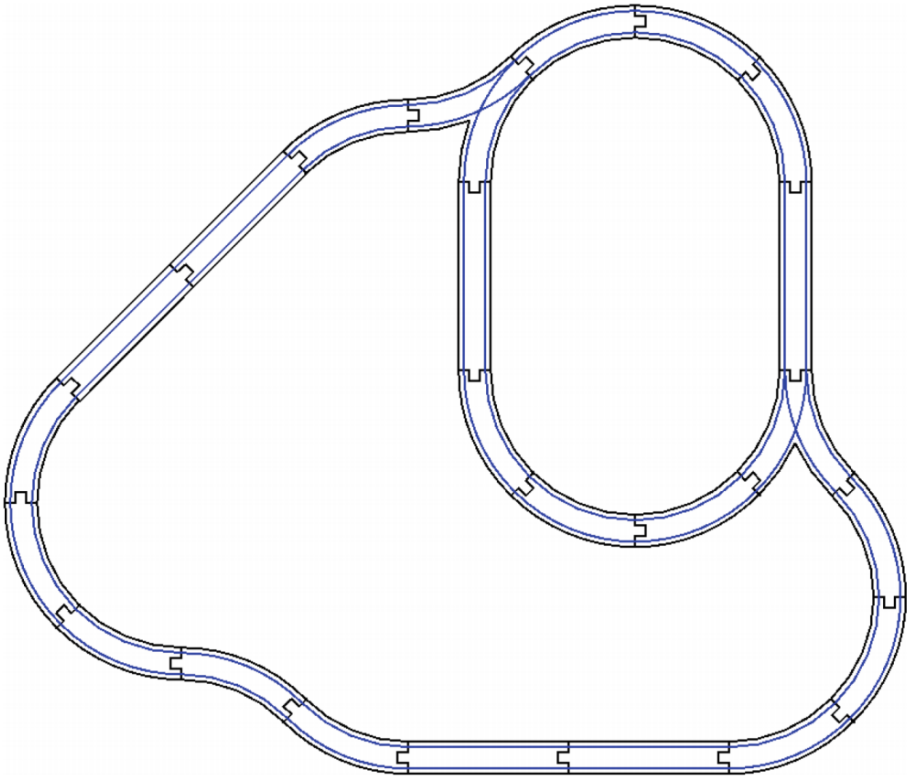


Fig. 15. The layout of Fig. 13 with the problematic section of track removed. This layout is safe in both directions. For clockwise travel around the perimeter, following the physical polarities, logical polarities are assigned to the inner loop.

Safety of the anticlockwise direction cannot be proved by physical polarities, because of the Y pieces. Figure 14 shows that it cannot be proved even by using logical polarities. This is because the section with dashed lines, where the arrows diverge, would require a connection between two logically negative endpoints. To prove safety we can observe that for the safe direction of travel, the section with dashed lines is unreachable. Therefore we can remove it (Fig. 15) to give an equivalent layout in which safety can be proved by logical polarities. In fact the layout of Fig. 15 is safe in both directions.

4 Conclusion

I have illustrated the ideas of static type systems in two non-programming domains: serial cables, and model railways. The examples demonstrate the following concepts.

- Typing rules impose local constraints on how components can be connected.

- Following the local typing rules guarantees a global semantic property.
- Typability is an approximation of semantic safety, and there are semantically safe systems whose safety can only be proved by reasoning outside the type system.
- If a type system doesn't type all of the configurations that we know to be safe, then a refined type system can be introduced in order to type more configurations (this is the step from physical to logical polarities in the railway example).

As far as I know, the use of a non-programming scenario to illustrate these concepts is new, or at least unusual, although I have not systematically searched for other examples.

There are several possible directions for future work. One is to increase the level of formality in the analysis of railway layouts, so that the absence of head-on collisions can be stated precisely as a theorem, and proved. Another is to elaborate on the step from physical to logical polarities, again in the railway scenario. Finally, it would be interesting to develop teaching and activity materials based on either or both examples, at a similar level to CS Unplugged.

Acknowledgements. I am grateful to Ornela Dardha, Conor McBride and Phil Wadler for comments on this paper and the seminar on which it is based; to João Seco for telling me about the *Alligator Eggs* presentation of untyped λ -calculus; and to an anonymous reviewer for noticing a small error.

References

1. Brio. www.brio.uk
2. CS Unplugged. csunplugged.org
3. Hornby. www.hornby.com
4. Theoretical Computer Science. www.journals.elsevier.com/theoretical-computer-science
5. Thomas & Friends. www.thomasandfriends.com
6. Abramsky, S., Gay, S.J., Nagarajan, R.: Interaction categories and the foundations of typed concurrent programming. In: Broy, M. (ed.) Proceedings of the NATO Advanced Study Institute on Deductive Program Design, pp. 35–113 (1996)
7. Awdrey, W.: Thomas the tank engine (1946)
8. Girard, J.-Y.: Linear logic. Theoret. Comput. Sci. **50**, 1–102 (1987)
9. Joyal, A., Street, R., Verity, D.: Traced monoidal categories. Math. Proc. Cambridge Philos. Soc. **119**(3), 447–468 (1996)
10. Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. J. Pure Appl. Algebra **19**, 193–213 (1980)
11. Ştefănescu, G.: Network Algebra. Springer, Heidelberg (2000). <https://doi.org/10.1007/978-1-4471-0479-7>
12. Victor, B.: Alligator eggs. worrydream.com/AlligatorEggs



Cathoristic Logic

A Logic for Capturing Inferences Between Atomic Sentences

Richard Evans^{1(✉)} and Martin Berger²

¹ Imperial College, London, UK
richardevans@google.com

² University of Sussex, Brighton, UK
M.F.Berger@sussex.ac.uk

Abstract. Cathoristic logic is a multi-modal logic where negation is replaced by a novel operator allowing the expression of incompatible sentences. We present the syntax and semantics of the logic including complete proof rules, and establish a number of results such as compactness, a semantic characterisation of elementary equivalence, the existence of a quadratic-time decision procedure, and Brandom's incompatibility semantics property. We demonstrate the usefulness of the logic as a language for knowledge representation.

Keywords: Modal logic · Hennessy-Milner logic · Transition systems · Negation · Exclusion · Elementary equivalence · Incompatibility semantics · Knowledge representation · Philosophy of language

1 Introduction

Natural language is full of incompatible alternatives. If Pierre is the current king of France, then nobody else can simultaneously fill that role. A traffic light can be green, amber or red - but it cannot be more than one colour at a time. Mutual exclusion is a natural and ubiquitous concept.

First-order logic can represent mutually exclusive alternatives, of course. To say that Pierre is the only king of France, we can write, following Russell:

$$\text{king}(\text{france}, \text{pierre}) \wedge \forall x. (\text{king}(\text{france}, x) \rightarrow x = \text{pierre}).$$

To say that a particular traffic light, tl , is red - and red is its only colour - we could write:

$$\text{colour}(tl, \text{red}) \wedge \forall x. \text{colour}(tl, x) \rightarrow x = \text{red}.$$

In this approach, incompatibility is a *derived* concept, reduced to a combination of universal quantification and identity. First-order logic, in other words, uses relatively complex machinery to express a simple concept:

- Quantification’s complexity comes from the rules governing the distinction between free and bound variables¹.
- Identity’s complexity comes from the infinite collection of axioms required to formalise the indiscernibility of identicals.

The costs of quantification and identity, such as a larger proof search space, have to be borne every time one expresses a sentence that excludes others - even though incompatibility does not, *prima facie*, appear to have anything to do with the free/bound variable distinction, or require the full power of the identity relation.

This paper introduces an alternative approach, where exclusion is expressed directly, as a first-class concept. Cathoristic logic² is the simplest logic we could find in which incompatible statements can be expressed. It is a multi-modal logic, a variant of Hennessy-Milner logic, that replaces negation with a new logical primitive

$$!A$$

pronounced *tantum*³ A . Here A is a finite set of alternatives, and $!A$ says that the alternatives in A exhaust all possibilities. For example:

$$!\{green, amber, red\}$$

states that nothing but *green*, *amber* or *red* is possible. Our logic uses modalities to state facts, for example $\langle amber \rangle$ expresses that *amber* is currently the case. The power of the logic comes from the conjunction of modalities and *tantum*. For example

$$\langle amber \rangle \wedge !\{green, amber, red\}$$

expresses that *amber* is currently the case and *red* as well as *green* are the only two possible alternatives to *amber*. Any statement that exceeds what *tantum A* allows, like

$$\langle blue \rangle \wedge !\{green, amber, red\},$$

is necessarily false. When the only options are green, amber, or red, then blue is not permissible. Now to say that Pierre is the only king of France, we write:

$$\langle king \rangle \langle france \rangle (\langle pierre \rangle \wedge !\{pierre\}).$$

Crucially, cathoristic logic’s representation involves no universal quantifier and no identity relation. It is a purely propositional formulation. To say that the traffic light is currently red, and red is its only colour, we write:

$$\langle tl \rangle \langle colour \rangle (\langle red \rangle \wedge !\{red\}).$$

¹ Efficient handling of free/bound variables is an active field of research, e.g. nominal approaches to logic [23]. The problem was put in focus in recent years with the rising interest in the computational cost of syntax manipulation in languages with binders.

² “Cathoristic” comes from the Greek *καθορίζειν*: to impose narrow boundaries. We are grateful to Tim Whitmarsh for suggesting this word.

³ “Tantum” is Latin for “only”.

This is simpler, both in terms of representation length and computational complexity, than the formulation in first-order logic given on the previous page. Properties changing over time can be expressed by adding extra modalities that can be understood as time-stamps. To say that the traffic light was red at time t_1 and amber at time t_2 , we can write:

$$\langle tl \rangle \langle colour \rangle (\langle t_1 \rangle (\langle red \rangle \wedge !\{red\}) \wedge \langle t_2 \rangle (\langle amber \rangle \wedge !\{amber\}))$$

Change over time can be expressed in first-order logic with bounded quantification - but modalities are succinct and avoid introducing bound variables.

Having claimed that incompatibility is a natural logical concept, not easily expressed in first-order logic⁴, we will now argue the following:

- Incompatibility is conceptually prior to negation.
- Negation arises as the weakest form of incompatibility.

1.1 Material Incompatibility and Negation

Every English speaker knows that

“Jack is male” is incompatible with “Jack is female”

But *why* are these sentences incompatible? The orthodox position is that these sentences are incompatible because of the following general law:

If someone is male, then it is not the case that they are female

Recast in first-order logic:

$$\forall x.(male(x) \rightarrow \neg female(x)).$$

In other words, according to the orthodox position, the incompatibility between the two particular sentences depends on a general law involving universal quantification, implication and negation.

Brandom [7] follows Sellars in proposing an alternative explanation: “Jack is male” is incompatible with “Jack is female” because “is male” and “is female” are *materially incompatible* predicates. They claim we can understand incompatible predicates even if we do not understand universal quantification or negation. Material incompatibility is conceptually prior to logical negation.

Imagine, to make this vivid, a primitive people speaking a primordial language of atomic sentences. These people can express sentences that *are* incompatible. But they cannot express *that* they are incompatible. They recognise when atomic sentences are incompatible, and see that one sentence entails another - but their behaviour outreaches their ability to articulate it.

Over time, these people *may* advance to a more sophisticated language where incompatibilities are made explicit, using a negation operator - but this is a later (and optional) development:

⁴ We will precisify this claim in later sections; (1) first-order logic’s representation of incompatibility is longer in terms of formula length than cathoristic logic’s (see Sect. 4.2); and (2) logic programs in cathoristic logic can be optimised to run significantly faster than their equivalent in first-order logic (see Sect. 5.3).

[If negation is added to the language], it lets one say that two claims are materially incompatible: “If a monochromatic patch is red, then it is not blue.” That is, negation lets one make explicit in the form of claims - something that can be said and (so) thought - a relation that otherwise remained implicit in what one practically did, namely treat two claims as materially incompatible⁵.

But before making this optional explicating step, our primitive people understand incompatibility without understanding negation. If this picture of our primordial language is coherent, then material incompatibility is conceptually independent of logical negation.

Now imagine a modification of our primitive linguistic practice in which no sentences are ever treated as incompatible. If one person says “Jack is male” and another says “Jack is female”, nobody counts these claims as *conflicting*. The native speakers never disagree, back down, retract their claims, or justify them. They just say things. Without an understanding of incompatibility, and the variety of behaviour that it engenders, we submit (following Brandom) that there is insufficient richness in the linguistic practice for their sounds to count as assertions. Without material incompatibility, their sounds are just *barks*.

Suppose the reporter’s differential responsive dispositions to call things red are matched by those of a parrot trained to utter the same noises under the same stimulation. What practical capacities of the human distinguish the reporter from the parrot? What, besides the exercise of regular differential responsive dispositions, must one be able to *do*, in order to count as having or grasping *concepts*? ... To grasp or understand a concept is, according to Sellars, to have practical mastery over the inferences it is involved in... The parrot does not treat “That’s red” as incompatible with “That’s green”⁶.

If this claim is also accepted, then material incompatibility is not just conceptually *independent* of logical negation, but conceptually *prior*.

1.2 Negation as the Minimal Incompatible

In [7] and [8], Brandom describes logical negation as a limiting form of material incompatibility:

Incompatible sentences are Aristotelian *contraries*. A sentence and its negation are *contradictories*. What is the relation between these? Well, the contradictory is a contrary: any sentence is incompatible with its negation. What distinguishes the contradictory of a sentence from all the rest of its contraries? The contradictory is the *minimal* contrary: the one that is entailed by all the rest. Thus every contrary of “Plane figure *f* is a circle” - for instance “*f* is a triangle”, “*f* is an octagon”, and so on - entails “*f* is *not* a circle”.

⁵ [8] pp. 47–48.

⁶ [7] pp. 88–89, our emphasis.

Here, “Jack loves Jill” is not a syntactic constituent⁸.

There are many types of inferential relations between atomic sentences of a natural language. For example:

- “Jack is male” is incompatible with “Jack is female”
- “Jack loves Jill” implies “Jack loves”
- “Jack walks slowly” implies “Jack walks”
- “Jack loves Jill and Joan” implies “Jack loves Jill”
- “Jack is wet and cold” implies “Jack is cold”

The first of these examples involves an incompatibility relation, while the others involve entailment relations. A key question this paper seeks to answer is: what is the simplest logic that can capture these inferential relations between atomic sentences?

1.4 Wittgenstein’s Vision of a Logic of Elementary Propositions

In the *Tractatus* [34], Wittgenstein claims that the world is a set of atomic sentences in an idealised logical language. Each atomic sentence was supposed to be *logically independent* of every other, so that they could be combined together in every possible permutation, without worrying about their mutual compatibility. But already there were doubts and problem cases. He was aware that certain statements seemed atomic, but did not seem logically independent:

For two colours, e.g., to be at one place in the visual field is impossible, and indeed logically impossible, for it is excluded by the logical structure of colour. (6.3751)

At the time of writing the *Tractatus*, he hoped that further analysis would reveal that these statements were not really atomic.

Later, in the *Philosophical Remarks* [33], he renounced the thesis of the logical independence of atomic propositions. In §76, talking about incompatible colour predicates, he writes:

That makes it look as if *a construction might be possible within the elementary proposition*. That is to say, as if there were a construction in logic which didn’t work by means of truth functions. What’s more, it also seems that these constructions have an effect on one proposition’s following logically from another. For, if different degrees exclude one another it follows from the presence of one that the other is not present. In that case, *two elementary propositions can contradict one another*.

Here, he is clearly imagining a logical language in which there are incompatibilities between atomic propositions. In §82:

⁸ To see that “Jack loves Jill” is not a constituent of “Jack loves Jill and Joan”, observe that “and” conjoins constituents of the *same syntactic type*. But “Jack loves Jill” is a sentence, while “Joan” is a noun. Hence the correct parsing is “Jack (loves (Jill and Joan))”, rather than “(Jack loves Jill) and Joan”.

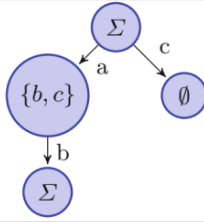


Fig. 1. Example model.

Here we assume, as we do with all subsequent figures, that the top state is the start state. The same model does not satisfy any of the following formulae.

$$\langle b \rangle \quad !\{a\} \quad !\{a, c\} \quad \langle a \rangle !\{b\} \quad \langle a \rangle \langle c \rangle \quad \langle a \rangle \langle b \rangle !\{c\}$$

Figure 2 shows various models of $\langle a \rangle \langle b \rangle$ and Fig. 3 shows one model that does, and one that does not, satisfy the formula $!\{a, b\}$. Both models validate $!\{a, b, c\}$.

Cathoristic logic does not have the operators $\neg, \vee,$ or \rightarrow . This has the following two significant consequences. First, every satisfiable formula has a unique (up to isomorphism) simplest model. In Fig. 2, the left model is the unique simplest model satisfying $\langle a \rangle \langle b \rangle$. We will clarify below that model simplicity is closely related to the process theoretic concept of similarity, and use the existence of unique simplest models in our quadratic-time decision procedure.

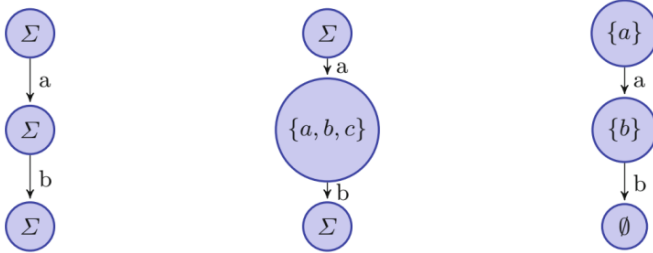


Fig. 2. Three models of $\langle a \rangle \langle b \rangle \top$

Secondly, cathoristic logic is different from other logics in that there is an asymmetry between tautologies and contradictories: logics with conventional negation have an infinite number of non-trivial tautologies, as well as an infinite number of contradictories. In contrast, because cathoristic logic has no negation or disjunction operator, it is expressively limited in the tautologies it can express: \top and conjunctions of \top are its sole tautologies. On the other hand, the tantum operator enables an infinite number of contradictories to be expressed. For example:

$$\langle a \rangle \wedge !\emptyset \quad \langle a \rangle \wedge !\{b\} \quad \langle a \rangle \wedge !\{b, c\} \quad \langle b \rangle \wedge !\emptyset$$



Fig. 16. Worked example of neg . Note that the transition system on the left is non-deterministic.

Now:

$$\begin{aligned}
 & \text{neg}(y, \langle a \rangle (\langle b \rangle \top \wedge \langle c \rangle \top)) \\
 &= \bigwedge_{y \xrightarrow{a} z} \langle a \rangle \text{neg}(z, \langle b \rangle \top \wedge \langle c \rangle \top) \\
 &= \langle a \rangle \text{neg}(z_1, \langle b \rangle \top \wedge \langle c \rangle \top) \wedge \langle a \rangle \text{neg}(z_2, \langle b \rangle \top \wedge \langle c \rangle \top) \\
 &= \langle a \rangle (\langle b \rangle \top \wedge \text{neg}(z_1, \langle c \rangle \top)) \wedge \langle a \rangle \text{neg}(z_2, \langle b \rangle \top \wedge \langle c \rangle \top) \\
 &= \langle a \rangle (\langle b \rangle \top \wedge \text{neg}(z_1, \langle c \rangle \top)) \wedge \langle a \rangle (\text{neg}(z_2, \langle b \rangle \top) \wedge \langle c \rangle \top) \\
 &= \langle a \rangle (\langle b \rangle \top \wedge !\{b\}) \wedge \langle a \rangle (\text{neg}(z_2, \langle b \rangle \top) \wedge \langle c \rangle \top) \\
 &= \langle a \rangle (\langle b \rangle \top \wedge !\{b\}) \wedge \langle a \rangle (!\{c\} \wedge \langle c \rangle \top)
 \end{aligned}$$

The resulting formula is true in y but not in y'_j .

B Omitted Proofs

B.1 Proof of Lemma 5

If $\mathfrak{M} \models \phi$ then $\mathfrak{M} \preceq \text{simpl}(\phi)$.

Proof. We shall show $\text{Th}(\text{simpl}(\phi)) \subseteq \text{Th}(\mathfrak{M})$. The desired result will then follow by applying Theorem 1. We shall show that

$$\text{If } \mathfrak{M} \models \phi \text{ then } \text{Th}(\text{simpl}(\phi)) \subseteq \text{Th}(\mathfrak{M})$$

by induction on ϕ . In all the cases below, let $\text{simpl}(\phi) = (\mathcal{L}, w)$ and let $\mathfrak{M} = (\mathcal{L}', w')$. The case where $\phi = \top$ is trivial. Next, assume $\phi = \langle a \rangle \psi$. We know $\mathfrak{M} \models \langle a \rangle \psi$ and need to show that $\text{Th}(\text{simpl}(\langle a \rangle \psi)) \subseteq \text{Th}(\mathfrak{M})$. Since $(\mathcal{L}', w') \models \langle a \rangle \psi$, there is an x' such that $w' \xrightarrow{a} x'$ and $(\mathcal{L}', x') \models \psi$. Now from the definition of $\text{simpl}()$, $\text{simpl}(\langle a \rangle \psi)$ is a model combining $\text{simpl}(\psi)$ with a new state w not appearing in $\text{simpl}(\psi)$ with an arrow $w \xrightarrow{a} x$ (where x is the start state in $\text{simpl}(\psi)$), and $\lambda(w) = \Sigma$. Consider any sentence ξ such that $\text{simpl}(\langle a \rangle \psi) \models \xi$. Given the construction of $\text{simpl}(\langle a \rangle \psi)$, ξ must be a conjunction of \top and

formulae of the form $\langle a \rangle \tau$. In the first case, (\mathcal{L}', x') satisfies \top ; in the second case, $(\mathcal{L}', x') \models \tau$ by the induction hypothesis and hence $(\mathcal{L}', w') \models \langle a \rangle \tau$.

Next, consider the case where $\phi = !A$, for some finite set $A \subset \Sigma$. From the definition of $\text{simpl}()$, $\text{simpl}(!A)$ is a model with one state s , no transitions, with $\lambda(s) = A$. Now the only formulae that are true in $\text{simpl}(!A)$ are conjunctions of \top and $!B$, for supersets $B \supseteq A$. If $\mathfrak{M} \models !A$ then by the semantic clause for $!$, $\lambda'(w') \subseteq A$, hence \mathfrak{M} models all the formulae that are true in $\text{simpl}(!A)$.

Finally, consider the case where $\phi = \psi_1 \wedge \psi_2$. Assume $\mathfrak{M} \models \psi_1$ and $\mathfrak{M} \models \psi_2$. We assume, by the induction hypothesis that $\text{Th}(\text{simpl}(\psi_1)) \subseteq \text{Th}(\mathfrak{M})$ and $\text{Th}(\text{simpl}(\psi_2)) \subseteq \text{Th}(\mathfrak{M})$. We need to show that $\text{Th}(\text{simpl}(\psi_1 \wedge \psi_2)) \subseteq \text{Th}(\mathfrak{M})$. By the definition of $\text{simpl}()$, $\text{simpl}(\psi_1 \wedge \psi_2) = \text{simpl}(\psi_1) \sqcap \text{simpl}(\psi_2)$. If $\text{simpl}(\psi_1)$ and $\text{simpl}(\psi_2)$ are inconsistent (see the definition of inconsistent in Sect. 6.4) then $\mathfrak{M} = \perp$. In this case, $\text{Th}(\text{simpl}(\psi_1) \wedge \text{simpl}(\psi_2)) \subseteq \text{Th}(\perp)$. If, on the other hand, $\text{simpl}(\psi_1)$ and $\text{simpl}(\psi_2)$ are not inconsistent, we shall show that $\text{Th}(\text{simpl}(\psi_1 \wedge \psi_2)) \subseteq \text{Th}(\mathfrak{M})$ by reductio. Assume a formula ξ such that $\text{simpl}(\psi_1 \wedge \psi_2) \models \xi$ but $\mathfrak{M} \not\models \xi$. Now $\xi \neq \top$ because all models satisfy \top . ξ cannot be of the form $\langle a \rangle \tau$ because, by the construction of merge (see Sect. 6.4), all transitions in $\text{simpl}(\psi_1 \wedge \psi_2)$ are transitions from $\text{simpl}(\psi_1)$ or $\text{simpl}(\psi_2)$ and we know from the inductive hypothesis that $\text{Th}(\text{simpl}(\psi_1)) \subseteq \text{Th}(\mathfrak{M})$ and $\text{Th}(\text{simpl}(\psi_2)) \subseteq \text{Th}(\mathfrak{M})$. ξ cannot be $!A$ for some $A \subset \Sigma$, because, from the construction of merge , all state-labellings in $\text{simpl}(\psi_1 \wedge \psi_2)$ are no more specific than the corresponding state-labellings in $\text{simpl}(\psi_1)$ and $\text{simpl}(\psi_2)$, and we know from the inductive hypothesis that $\text{Th}(\text{simpl}(\psi_1)) \subseteq \text{Th}(\mathfrak{M})$ and $\text{Th}(\text{simpl}(\psi_2)) \subseteq \text{Th}(\mathfrak{M})$. Finally, ξ cannot be $\xi_1 \wedge \xi_2$ because the same argument applies to ξ_1 and ξ_2 individually. We have exhausted the possible forms of ξ , so conclude that there is no formula ξ such that $\text{simpl}(\psi_1 \wedge \psi_2) \models \xi$ but $\mathfrak{M} \not\models \xi$. Hence $\text{Th}(\text{simpl}(\psi_1 \wedge \psi_2)) \subseteq \text{Th}(\mathfrak{M})$. \square

B.2 Proof of Lemma 6

If $\phi \models \psi$ then $\text{simpl}(\phi) \preceq \text{simpl}(\psi)$

Proof. By Theorem 1, $\text{simpl}(\phi) \preceq \text{simpl}(\psi)$ iff $\text{Th}(\text{simpl}(\psi)) \subseteq \text{Th}(\text{simpl}(\phi))$. Assume $\phi \models \psi$, and assume $\xi \in \text{Th}(\text{simpl}(\psi))$. We must show $\xi \in \text{Th}(\text{simpl}(\phi))$. Now $\text{simpl}()$ is constructed so that:

$$\text{simpl}(\psi) = \bigsqcup \{ \mathfrak{M} \mid \mathfrak{M} \models \psi \}$$

So $\xi \in \text{Th}(\text{simpl}(\psi))$ iff for all models \mathfrak{M} , $\mathfrak{M} \models \psi$ implies $\mathfrak{M} \models \xi$. We must show that $\mathfrak{M} \models \phi$ implies $\mathfrak{M} \models \xi$ for all models \mathfrak{M} . Assume $\mathfrak{M} \models \phi$. Then since $\phi \models \psi$, $\mathfrak{M} \models \psi$. But since $\xi \in \text{Th}(\text{simpl}(\psi))$, $\mathfrak{M} \models \xi$ also. \square

B.3 Proof of Lemma 7

If $\mathcal{I}(\psi) \subseteq \mathcal{I}(\phi)$ then $\mathcal{J}(\text{simpl}(\psi)) \subseteq \mathcal{J}(\text{simpl}(\phi))$

Proof. Assume $\mathcal{I}(\psi) \subseteq \mathcal{I}(\phi)$ and $\mathfrak{M} \sqcap \text{simpl}(\psi) = \perp$. We need to show $\mathfrak{M} \sqcap \text{simpl}(\phi) = \perp$. If $\mathcal{I}(\psi) \subseteq \mathcal{I}(\phi)$ then for all formulae ξ , if $\text{simpl}(\xi) \sqcap \text{simpl}(\psi) = \perp$ then $\text{simpl}(\xi) \sqcap \text{simpl}(\phi) = \perp$. Let ξ be $\text{char}(\mathfrak{M})$. Given that $\mathfrak{M} \sqcap \text{simpl}(\psi) = \perp$ and $\text{simpl}(\text{char}(\mathfrak{M})) \preceq \mathfrak{M}$, $\text{simpl}(\text{char}(\mathfrak{M})) \sqcap \text{simpl}(\psi) = \perp$. Then as $\mathcal{I}(\psi) \subseteq \mathcal{I}(\phi)$, $\text{simpl}(\text{char}(\mathfrak{M})) \sqcap \text{simpl}(\phi) = \perp$. Now as $\mathfrak{M} \preceq \text{simpl}(\text{char}(\mathfrak{M}))$, $\mathfrak{M} \sqcap \text{simpl}(\phi) = \perp$. \square

References

1. Haskell implementation of cathoristic logic. Submitted with the paper (2014)
2. Abramsky, S.: Computational interpretations of linear logic. *TCS* **111**, 3–57 (1993)
3. Allan, K. (ed.): *Concise Encyclopedia of Semantics*. Elsevier, Boston (2009)
4. Aronoff, M., Rees-Miller, J. (eds.): *The Handbook of Linguistics*. Wiley-Blackwell, Hoboken (2003)
5. Blackburn, P., de Rijke, M., Venema, Y.: *Modal Logic*. Cambridge University Press, Cambridge (2001)
6. Brachman, R., Levesque, H.: *Knowledge Representation and Reasoning*. Morgan Kaufmann, Burlington (2004)
7. Brandom, R.: *Making It Explicit*. Harvard University Press, Cambridge (1998)
8. Brandom, R.: *Between Saying and Doing*. Oxford University Press, Oxford (2008)
9. Davey, B.A., Priestley, H.A.: *Introduction to Lattices and Order*. Cambridge University Press, Cambridge (1990)
10. Davidson, D.: *Essays on Actions and Events*. Oxford University Press, Oxford (1980)
11. Enderton, H.B.: *A Mathematical Introduction to Logic*. Academic Press, Cambridge (2001)
12. Evans, R., Short, E.: Versu. <http://www.versu.com>. <https://itunes.apple.com/us/app/blood-laurels/id882505676?mt=8>
13. Evans, R., Short, E.: Versu - a simulationist storytelling system. *IEEE Trans. Comput. Intell. AI Games* **6**(2), 113–130 (2014)
14. Fikes, R., Nilsson, N.: Strips: a new approach to the application of theorem proving to problem solving. *Artif. Intell.* **2**, 189–208 (1971)
15. Girard, J.-Y.: *Linear logic*. *TCS* **50**, 1–101 (1987)
16. Hennessy, M.: *Algebraic Theory of Processes*. MIT Press Series in the Foundations of Computing. MIT Press, Cambridge (1988)
17. Hennessy, M., Milner, R.: Algebraic laws for non-determinism and concurrency. *JACM* **32**(1), 137–161 (1985)
18. Honda, K.: A Theory of Types for the π -Calculus, March 2001. <http://www.dcs.qmul.ac.uk/~kohei/logics>
19. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) *ESOP 1998*. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0053567>
20. Honda, K., Yoshida, N.: A uniform type structure for secure information flow. *SIGPLAN Not.* **37**, 81–92 (2002)
21. O’Keeffe, A., McCarthy, M. (eds.): *The Routledge Handbook of Corpus Linguistics*. Routledge, Abingdon (2010)
22. Peregrin, J.: Logic as based on incompatibility (2010). <http://philpapers.org/rec/PERLAB-2>

2. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings 4th Annual ACM Symposium on Principles of Programming Languages, pp. 238–252 (1977)
3. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. POPL 1979, pp. 269–282. ACM, New York (1979). <https://doi.org/10.1145/567752.567778>
4. Duchi, J.C., Jordan, M.I., Wainwright, M.J.: Local privacy and statistical minimax rates. In: 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1592–1592, October 2013. <https://doi.org/10.1109/Allerton.2013.6736718>
5. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_1
6. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_14
7. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. *Found. Trends Theoret. Comput. Sci.* **9**, 211–407 (2014). <https://doi.org/10.1561/04000000042>
8. Ebadi, H.: Dynamic Enforcement of Differential Privacy. Ph.D. thesis, Chalmers University of Technology, March 2018
9. Ebadi, H.: The PreTPost Framework (2018). <https://github.com/ebadi/preTpost>
10. Ebadi, H., Sands, D.: PreTPost: a transparent, user verifiable, local differential privacy framework (2018). <https://github.com/ebadi/preTpost>. Also appears in [8]
11. Ebadi, H., Sands, D., Schneider, G.: Differential privacy: now it's getting personal. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL 2015, pp. 69–81. ACM (2015). <https://doi.org/10.1145/2676726.2677005>
12. Erlingsson, Ú., Pihur, V., Korolova, A.: RAPPOR: randomized aggregatable privacy-preserving ordinal response. In: CCS. ACM (2014)
13. Gaboardi, M., Haeberlen, A., Hsu, J., Narayan, A., Pierce, B.C.: Linear dependent types for differential privacy. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. POPL 2013, pp. 357–370. ACM, New York (2013). <https://doi.org/10.1145/2429069.2429113>
14. Giacobazzi, R., Ranzato, F.: Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program.* **32**(1), 177–210 (1998). [https://doi.org/10.1016/S0167-6423\(97\)00034-8](https://doi.org/10.1016/S0167-6423(97)00034-8), <http://www.sciencedirect.com/science/article/pii/S0167642397000348>. 6th European Symposium on Programming
15. Haeberlen, A., Pierce, B.C., Narayan, A.: Differential privacy under fire. In: Proceedings of the 20th USENIX Conference on Security. SEC 2011, pp. 33–33. USENIX Association, Berkeley (2011). <http://dl.acm.org/citation.cfm?id=2028067.2028100>
16. Hunt, S.: Abstract interpretation of functional languages: from theory to practice. Ph.D. thesis, Imperial College London, UK (1991)
17. Hunt, S., Sands, D.: Binding time analysis: a new perspective. In: Proceedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM 1991), pp. 154–164. ACM Press (1991)