

Issue 2025-1  
January/February 2025

**F A C S**

**A**

**C**

**T**

**S**

FME  
A ACM  
C T  
L F C  
METHODS C  
BCS R SCSC  
M  
Z A  
UML  
IFMSIG  
E E E



Formal Aspects of  
Computing Science  
Specialist Group

The Newsletter of the  
Formal Aspects of Computing Science  
(FACS) Specialist Group

ISSN 0950-1231

## About FACS FACTS

FACS FACTS (ISSN: 0950-1231) is the newsletter of the BCS Specialist Group on Formal Aspects of Computing Science (FACS). FACS FACTS is distributed in electronic form to all FACS members.

Submissions to FACS FACTS are always welcome. Please visit the newsletter area of the BCS FACS website for further details at:

<https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/>

Back issues of FACS FACTS are available for download from:

<https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/back-issues-of-facs-facts/>

## The FACS FACTS Team

### Newsletter co-editors:

Tim Denvir      [timdenvir@bcs.org](mailto:timdenvir@bcs.org)  
Brian Monahan      [brianqmonahan@gmail.com](mailto:brianqmonahan@gmail.com)

### Editorial team:

Jonathan Bowen, Tim Denvir, Keith Lines, Brian Monahan

### Contributors to this issue:

Jonathan Bowen, Tim Denvir, Steve Dunne, Einar Broch Johnsen, Keith Lines,  
Brian Monahan, Chunyan Mu, David Pym, Bill Stoddart

### BCS-FACS websites:

BCS      <https://facs.bcs.org>  
LinkedIn      <https://www.linkedin.com/groups/2427579/>  
Wikipedia      <https://en.wikipedia.org/wiki/BCS-FACS>

If you have any questions about BCS-FACS, please send these to Jonathan Bowen at:  
[jonathan.bowen@lsbu.ac.uk](mailto:jonathan.bowen@lsbu.ac.uk).

## Editorial

Dear reader,

In this issue we have the annual report to the BCS on the activities of FACS over 2024 from our chairman, Jonathan Bowen. We can feel satisfied that six FACS seminars were held in 2024, including the annual Peter Landin Semantics seminar from David Pym. The annual FACS/LMS lecture from Annabel McIver and a report has been promised for the July newsletter. Then we have two feature articles, *Much ado about nothing* by Bill Stoddart, Steve Dunne, and Chunyan Mu and *Linear Steganography, Ring Homomorphisms and Homomorphic Encipherment* by Theodore Murray Abelson. There's a pseudonymous contribution if ever there was one!

A report by Brian Monahan and Bill Stoddart on the 2024 Landin Semantics seminar from David Pym follows. Keith Lines reports on the FME/BCS-FACS Joint webinar: *The Self-Aware Digital Twin*, presented by Einar Broch Johnsen. Then Jonathan Bowen reviews recent Festschrift publications in the Springer LNCS series, a two-volume work celebrating the contributions of Cliff Jones and another volume marking the retirement of Jim Woodcock, whose work has benefited from the inspiration of Cliff Jones. Next Tim Denvir reviews both the book, *The Art of Uncertainty* by David Spiegelhalter and his strongly related talk on "Chance, Luck and Ignorance (How to put our uncertainty into numbers)" which he gave to the Edinburgh Mathematical Society. Risk and uncertainty have some relevance to the expectations of users when faced with a user interface, which can lead to misunderstandings and unexpected behaviours.

We also have some news about a couple of computer scientists who will be well-known to readers. First of all, we can report that Glynn Winskel was appointed Professor of Theoretical Computer Science at Queen Mary University of London from January 2024. However, on a sadder note, we learned from his [Wikipedia page](#) that Mike Shields died in September 2023. Mike specialised in the semantics of concurrency, and his book, *Semantics of Parallelism: Non-interleaving representation of behaviour*, Springer, ISBN 978-3540760597, (1997), was seminal. When we have more secure confirmation, we shall produce a more comprehensive tribute to Mike Shields in the mid-year issue.

Most FACS readers will have noticed that 2025 is a square, which makes it rather special. The last square year was 1936 and the next will not be until 2116: some of our youngest grandchildren might get to experience that one!

We do hugely appreciate and look forward to your contributions, including comments, from you, our readers. We hope you enjoy FACS FACTS issue 2025-1.

*Tim Denvir*  
*Brian Monahan*

## A New Look

You may have noticed that this edition of FACS FACTS looks a little bit different from usual. We have now whole-heartedly embraced using  $\LaTeX$ , the academic and technical publication standard, for *producing* our newsletter. This has allowed a number of improvements in process, productivity, and hopefully, uniformity.

## Contributions

For your reassurance, we continue to accept contributions in a wide range of formats, including:  $\LaTeX$ , Word, ODT, HTML, Markdown, plain text, and even PDF. If your contribution involves technical code, formulae, diagrams etc., it is often helpful to provide not only the source but also a PDF of your contribution showing how it should appear.

Finally, although we very much welcome relevant contributions to FACS FACTS from the FACS community, the editors and editorial team reserve the right to select which contributions are published in the newsletter. FACS FACTS is a newsletter, not a journal. As such, we have neither the resource, the capacity, nor access to the expertise to conduct detailed peer reviews of submitted contributions. Therefore, readers should regard any technical articles published here as, at best, research announcements and ongoing work-in-progress drafts.

*The Editorial Team*

## Table of Contents

<b>Editorial</b> .....	3
<b>BCS-FACS Specialist Group 2024 Chair’s Report</b> .....	6
<b>Linear Steganography, Ring Homomorphisms, and Homomorphic Encipherment</b> .....	27
<i>by Theodore Murray Abelson</i>	
<b>BCS-FACS Peter Landin Semantics Seminar 2024</b>	
<b>Inferentialism, Proof-theoretic Semantics, and Computation</b> .....	50
<i>Speaker: David Pym</i>	
<i>Reported by: Brian Monahan and Bill Stoddart</i>	
<b>FME/BCS-FACS Joint Webinar</b>	
<b>The Self-Aware Digital Twin</b> .....	55
<i>Speaker: Einar Broch Johnsen</i>	
<i>Reported by: Keith Lines</i>	
<b>Festschrift reviews: Cliff Jones and Jim Woodcock</b> .....	60
<i>Reviewed by: Jonathan P. Bowen</i>	
<b>Book Review: The Art of Uncertainty</b> .....	63
<i>by David Spiegelhalter</i>	
<i>Reviewed by: Tim Denvir</i>	
<b>Forthcoming Events</b> .....	65
<b>FACS Committee</b> .....	66

## BCS-FACS Specialist Group 2024 Chair's Report

<b>Member Group Name:</b>	FACS Specialist Group
<b>Year:</b>	2024
<b>Report By:</b>	Jonathan Bowen

<b>Group Chair:</b>	Jonathan Bowen	
<b>Group Treasurer:</b>	John Cooke	
<b>Group Secretary:</b>	Roger Carsley	
<b>Group Inclusion Officer:</b>	Margaret West	
<b>Other Committee Members:</b>	Ana Cavalcanti	FME Liaison
	Tim Denvir	<i>FACS FACTS</i> newsletter co-editor
	Brijesh Dongol	Workshop Liaison
	Keith Lines	Government and Standards Liaison
	Alvaro Miyazawa	Seminar Organiser
	Brian Monahan	<i>FACS FACTS</i> newsletter co-editor
	Andrei Popescu	LMS Liaison

## Successes

Success	Additional Comments
1. Continued evening seminars online, with recordings on YouTube	The BCS Zoom facilities and recording transfer to YouTube have widened access to FACS seminars and support for these by the BCS has improved. Thank you to Alvaro Miyazawa, Andrei Popescu, Keith Lines, and the Chair for help with organising FACS events in 2024.
2. Publication of <i>FACS FACTS</i> newsletters by BCS-FACS and <i>Formal Aspects of Computing</i> journal by ACM	We aim for two major newsletters each year, published online in PDF format. Thank you to Tim Denvir and Brian Monahan for continuing with their excellent editing of the two 2024 newsletters. Contributions by FACS members are always welcome. The associated FAC journal is published by ACM with open access to papers. See <a href="https://dl.acm.org/journal/fac">https://dl.acm.org/journal/fac</a>

Success	Additional Comments
3. Move to online and hybrid events	In 2024, six evening seminars were delivered as webinars or hybrid events, including our usual our highlight event, the Peter Landin Semantics Seminar at the BCS London office in December, delivered in hybrid format after the FACS AGM.

## Plans

Planned Activity	Additional Comments
1. Continued online and some hybrid events	The BCS facilities for online and hybrid talks enable these modes of delivery. Alvaro Miyazawa of the University of York is the FACS Seminar Organiser.
2. At least two <i>FACS FACTS</i> newsletters	We continue to aim for early and mid-year as times of publication.
3. Collaboration with related organizations such as Formal Methods Europe (FME), London Mathematical Society (LMS), and newly the Edinburgh Mathematical Society.	Our LMS Liaison Officer, Andrei Popescu is organizing an online FACS/LMS talk in January 2025 by Annabel McIver (based in Australia).

## Impediments

Impediment	Description
1. Succession planning for executive officers	The FACS executive officers will need to be replaced at some point in the not-too-distant future. Succession planning is needed for this. We do have some younger new members of the FACS committee at least.
2. Reduced BCS funding	Our funding from the BCS for the 2024/25 financial year continues at a much-reduced amount compared to pre-Covid times. Thus, our programme of seminars is now realistically reduced to a maximum of two physical events per year. That said, online seminars are much more cost-neutral, so we can continue with these more easily.

Impediment	Description
3. Reduced physical meetings	The lack of physical meetings impedes networking by FACS members. On the other hand, online seminars are popular due to the reduced cost for attendees and provide a wider national and international reach.

## Additional Facts and Figures

We aim for at least two *FACS FACTS* newsletters per year (with two in 2024, in January and July). We also aim for four to six online/hybrid evening seminars per year (with six in 2024, four online and two hybrid).

## Further Comments

For the record, FACS organised the following events during 2024:

1. *Formalising 21st-Century Mathematics*, by **Lawrence Paulson**, University of Cambridge, 15 January. Online with LMS, organised by Andrei Popescu.
2. *The SI Digital Framework: Underpinning FAIR measurement data*, by **Jean-Laurent Hippolyte**, NPL, 20 February. Online, organised by Keith Lines.
3. *Scott Models for Probabilistic Computation*, by **Abbas Edalat**, Imperial College London, 26 March. Online, organised by Alvaro Miyazawa.
4. *Verifying System-level Properties of Neural-network Robotic Controllers*, by **Jim Woodcock**, University of York, 28 May. Hybrid at the University of York, organised by Alvaro Miyazawa.
5. *The Self-aware Digital Twin*, by **Einar Broch Johnsen**, University of Oslo, Norway, 15 October. Online with FME, organised by Alvaro Miyazawa and Ana Cavalcanti.
6. *Inferentialism, Proof-theoretic Semantics, and Computation*, by **David Pym**, University College London, 12 December. Hybrid Landin Seminar with FACS AGM at the BCS London office, organised by Jonathan Bowen.

Thank you to all the FACS committee members for performing their various roles, as detailed above. New committee members and new ideas for activities/collaborations are very welcome, especially if interested in co-organising events.



# Much Ado About Nothing

Bill Stoddart Steve Dunne Chunyan Mu  
*Teesside University*

Corresponding author: [w.j.stoddart@gmail.com](mailto:w.j.stoddart@gmail.com)

## Abstract

We propose that difficulties in handling partial functions and undefined terms in the context of classical first order predicate logic can be eliminated by a small tweak to FOPL along with a corresponding adjustment to our mathematical theory which has to be able to incorporate a formalisation of “nothing” as a conceptual object within our domain of discourse.

## 1. Introduction

Eric Hehner uses a radical reformulation of set theory, in which the collection and packaging of elements are seen as separate activities. This provides for unpacked collections, referred to as “bunches”. Bunches allow us to reason about “nothing”, which is just an empty bunch (and very different from an empty set).

When our research group at Teesside University first heard of bunches, we were using the B-method to formalise a theory of reversible computation. We found that using bunch theory, with its simple and intuitive description of non-determinism, could greatly simplify our presentation. In [DFM<sup>+</sup>23] we employ it to describe backtracking GSL (bGSL), a variant of Abrials Generalised Substitution Language adapted for describing the semantics and refinement of backtracking programs. In [DMSZ24] we give the logic and axioms of our theory of bunches and provide a model. We also continue our discussion of bGSL, which we propose as an extremely expressive guarded command language, supporting new programming structures able to perform speculative computations free of side effects, and provide for simple unification of non-deterministic and probabilistic choice.

The B-method is formulated using classical logic, by which we mean first order predicate logic with equality (FOPL). FOPL is a logical foundation on which diverse theories can be constructed by stating further axioms which introduce, following a strict recipe, a “universe of discourse” (UofD). For the purposes of setting exercises in a course on logic, such axioms and their associated domain of discourse could provide a theory in which “All the nice girls love a sailor”. More serious examples include a theory of natural numbers, using Peano’s axioms, or set theory (and with it, some would argue, the whole of mathematics). In addition, FOPL has been widely studied, and we have, for example, the result from Turing’s paper of 1936 [Tur36] that a theory underpinned by FOPL may not be decidable, which contributed to the resolution of Hilbert’s Entscheidungsproblem.

FOPL is widely considered to be the standard logic for mathematical theories, as noted in these extracts from the article “The Emergence of first order logic” from the Stanford Encyclopaedia of Philosophy: [Ewa19]

For anybody schooled in modern logic, first-order logic can seem an entirely natural object of study, and its discovery inevitable. It is semantically complete; it is adequate to the axiomatization of all ordinary mathematics;..

The second quote terminates a historical review in the same article:

At this point in the 1930s, however, several other strands of thinking about logic now coalesced...The fact that higher-order logic could be construed as (in Quine's later phrase) "set theory in sheep's clothing" reinforced the other tendencies: "true" logic was first-order; higher-order logic was "really" set theory. By the end of the decade, a consensus had been reached that, for purposes of research in the foundations of mathematics, mathematical theories ought to be formulated in first-order terms. Classical first-order logic had become "standard".

However, we will see in this article that FOPL is not adequate as an underpinning for a theory of bunches, and this is precisely because of an inability to accommodate the formal concept of "nothing" that bunch theory allows us to express. We will formulate the changes required to adapt it to this purpose, obtaining the logic FOPLN which is equipped for reasoning about "nothing".

Our second important theme is that the ability to express the concept of nothing resolves problems associated with using classical logic along with partial functions.

The rest of this article is organised as follows: in section 2 we give a sufficient description of bunch theory; in section 3 we consider first order predicate logic (FOPL) and its ability to prove that there exist values corresponding to undefined terms, such as  $1/0$ ; in section 4 we show that FOPL plus bunch theory is inconsistent and introduce FOPLN, a first order predicate logic which can accommodate the concept of "nothing" and is suitable for use with bunch theory; in section 6 we describe "Mini bunch theory" which enables FOPLN to be used with set theory, in section 7 we ask whether FOPLN simplifies the use of partial functions to the extent of providing a "free lunch" [Jon96], and in sections 8 and 9 we consider related work and draw our conclusions. In the appendix we give a list of logic toolbox results provable in both FOPL and FOPLN (with proofs available on line), and give the inference rules for FOPLN.

## 2. About bunch theory

Following Hehner [Heh93, Heh23], we give a mathematical meaning to the contents of a set, which we call a bunch: e.g. the contents of the set  $\{1, 2\}$  is the bunch  $1, 2$ .

We write  $\sim A$  ("unpack  $A$ ") for the contents of set  $A$ , thus  $\sim\{1, 2\} = 1, 2$ . The comma in the expression  $1, 2$  is now a mathematical operator, called bunch union. It is associative and commutative, and its precedence is just below that of the expression connectives. It is associated with set union via the rule

$$\{A\} \cup \{B\} = \{A, B\}$$

We have the dual notion of bunch intersection  $A'B$  which is associated with set intersection via the rule

$$\{A\} \cap \{B\} = \{A'B\}$$

A bunch  $A$  is a sub-bunch of  $B$  if each element of  $A$  is an element of  $B$ . We write this as  $A : B$  (" $A$  is part of  $B$ "). Sub-bunches are related to subsets by the rule:

$$A : B \Leftrightarrow \{A\} \subseteq \{B\}$$

The empty bunch *null* is obtained by unpacking the empty set:

$$null \hat{=} \sim \{ \}$$

Constants can represent any bunch, for example we can characterise the bunch of natural numbers *nat* by the property  $nat \neq null \wedge nat = nat + 1$ . Variables always take elemental values.

Function application is lifted when applied to a bunch, thus if  $A = x, y$  then  $f(A) = f(x), f(y)$ . We take the infix syntax of binary operations as a sugared form of standard function application, so these are similarly lifted to apply pairwise, e.g. adding the bunches 0, 1 and 2, 4 yields  $0 + 2, 0 + 4, 1 + 2, 1 + 4$ .

A partial function applied to an element gives a *null* result iff the element is not in the function's domain.

We generalise function application  $f(E)$  such that  $f$  may be any relation and  $E$  a bunch. To formally define  $f(E)$  we use B's relational image:

$$f(E) \hat{=} \sim f(\{E\})$$

Since relational image is always well defined, this gives us the pleasant property that function application is always well defined. In contrast to function definition in Z, B and friends, where  $f(x)$  is undefined when  $x \notin \text{dom}(f)$ , we have  $x \notin \text{dom}\{f\} \Rightarrow f(x) = null$ , e.g.  $1/0 = null$ , and *null* is perfectly well defined.

Since the logical foundation of B is FOPL, a predicate in B is totally different from an expression. A predicate is a formula that may be subject to proof, whereas expressions denote values. In particular in B predicates *do not denote boolean truth values*. This remains the case in our theory of bunches, which we obtain by extending the set theory of B and replacing FOPL with FOPLN. Accordingly, predicates in our bunch theory are not lifted, but must still be generalised for use with bunches.

The basic predicates in set theory are equality and set membership.

We inherit the equality predicate from our logic, which gives us the axiom  $E = E$  and the referential transparency rule of Leibniz. We use  $E \neq F$  as syntactic sugar for  $\neg E = F$ .<sup>1</sup>

For membership, since we are extending the set theory of B to include bunches, we initially inherit the membership predicate from set theory, as it applies to elements, and we redefine this to apply more generally to bunches. In such redefinitions the  $\in$  to the left of the definition symbol is the new predicate being defined, and the  $\in$  to the right is the original predicate we inherit from set theory. We will say  $E \in S$  will be true exactly when each element of the bunch  $E$  is a member of each element of the bunch  $S$ . Formally, given  $E$  of type  $T$  and  $S$  of type  $\mathbb{P}(T)$ , we have

$$E \in S \hat{=} \forall e, s \bullet e : E \wedge s : S \Rightarrow e \in s$$

However, this definition allows for vacuous truths, i.e truths with no exemplers, when either  $E$  or  $S$  is *null*. Vacuous truths are familiar to us through predicates such  $\forall x \bullet x \in \{ \} \Rightarrow P$ , and through the vacuous truth established by the infeasible operation *magic* in weakest pre-condition program semantics.

However, it has been argued that in natural language a sentence such as “the current king of France is bald” includes an existential claim that there is a current king of France [Rus05, WR10]. Thus if we wish to express this in our theory as  $king\_of(France) \in bald$ , the membership predicate should include such a claim. We leave it as an exercise for the interested reader to modify the definition for membership given above to include existential entailment.

<sup>1</sup>In the set theory of classical B, the symbol  $\neq$  is used for relational inequality. In bunch theory the relational inequality of  $E$  and  $F$  represents a complete absence of overlap between bunches  $E$  and  $F$ , which we can express using bunch intersection as  $E \cap F = null$ ; a relational inequality symbol would be redundant, and possibly misleading: for example, if  $\neq$  is relational inequality, then  $1, 2 \neq 1, 3$  is false.

All “relational” predicate symbols are derived from set membership. For example given the relation defined as: <sup>2</sup>

$$_ < _ \hat{=} \{x, y \mid x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge x < y\}$$

we interpret  $E < F$  as  $E \mapsto F \in \_ < \_$ , where maplet formation is lifted similarly to function application. Thus  $E < F$  tells us that each element of  $E$  is less than every element of  $F$ . Such predicates are termed relational.

We have said that variables in our theory represent elements, and indeed it is a consequence of our *logic* that this will hold for bound variables. Indeed, were they to range over all possible bunches there would be unfortunate consequences, e.g.  $\forall x \bullet x \in \mathbb{N} \Rightarrow x < x + 1$  would no longer be true.

The main exposition of Bunch Theory is given in Hehner’s book “A Practical Theory of Programming” [Heh93]. Contributions to the theory and application of bunches by Hehner’s co-workers include [NH92] and [PH99]. Morris and Bunkenburg provided a model for a simple theory of bunches which does not include sets, but does include functions [MB01]. “The functional treatment of parsing” [Lee93] is a text-book that uses bunches as the basis of its approach.

Our approach differs from all other formulations of bunch theory in that we build it on a foundation of first order predicate logic, though as we will see in the following section, some changes to classical FOPL are required.

### 3. First order predicate logic and the mysterious number equal to 1/0

In FOPL, with integers as a domain of discourse, it is possible to prove that there exists a number equal to 1/0. The proof takes two lines:

1.  $1/0 = 1/0$       EQUALS
2.  $\exists x \bullet x = 1/0$      $\exists$ -intro

Step 1 of the proof is justified by the property of equality that says any expression is equal to itself, and step 2 is an existential introduction step we for which we are using the rule:

$$\frac{HYP \vdash P[E/x]}{HYP \vdash \exists x \bullet P} \quad \exists\text{-intro}$$

In this rule,  $P$  is a predicate in which  $x$  may occur free, and  $P[E/x]$  is  $P$  with some expression  $E$  replacing all occurrences of  $x$ . For our proof step  $P$  can be  $x = 1/0$ ,  $E$  is  $1/0$ , and thus  $P[E/x]$  is  $1/0 = 1/0$ , which has been shown to be true in line 1. Thus the premiss of the  $\exists$ -into rule is true, and we can conclude that there exists a number equal to 1/0.

We can also use the  $\forall$ -elim rule to prove the dual of the above result, namely that it is not true that every number is not equal to 1/0.

The article “First Order Model Theory” [HS24] in the Stanford Encyclopaedia of Philosophy, comments as follows: “any competent mathematician puts the condition ‘x is not zero’ before dividing by x, and so it never matters what the value of 1/0 is, and we can harmlessly take it to be 42”. This expresses a view with which one might disagree, and with which we are forced to disagree when we formulate

---

<sup>2</sup>Because we extend the set theory of B to include bunches, and we respect established usage, we are left with some uses of comma that do not represent bunch union, and here we have such an example in  $\{x, y \mid \dots$

our theory of bunches, since in our theory we have  $1/0 = \text{null}$ .

In B, classical FOPL is used, but to provide for type checking the syntactic form for existential quantification is  $\exists x \bullet x \in S \wedge P$ <sup>3</sup>, so the predicate  $\exists x \bullet x = 1/0$  is not available to be proved. When it is re-expressed in the required form we have  $\exists x \bullet x \in \mathbb{Z} \wedge x = 1/0$ . This type checks, but cannot be proved, because we cannot establish a suitable premiss for our  $\exists$ -into rule, e.g.  $1/0 \in \mathbb{Z} \wedge 1/0 = 1/0$ , because the proof rules of B do not allow us to prove anything about the application of a partial function outside its domain, other than that it equals itself.

## 4. FOPLN, a logic which accommodates the concept of “nothing”

### First order predicate logic (FOPL) plus bunch theory is inconsistent

When FOPL is used with Bunch Theory, we have inconsistency:  $\text{null} = \text{null}$  from the EQUALS rule of FOPL, and thus, equivalently,  $(x = \text{null})[\text{null}/x]$ . Using this as a premiss we can use the  $\exists$ -into rule of FOPL to obtain  $\exists x \bullet x = \text{null}$ , and recalling that variables, including variables bound by quantifiers, range over elements, we have proved the impossible result that there exists an *element*  $x$  equal to *null*.

### Accommodating the *null* value

Bunch theory eliminates gaps in theories caused by undefined terms, but in the previous section we have seen that the presence of *null*, a conceptual entity representing nothing, renders FOPL inconsistent. We now describe a new logic FOPLN which is a first order predicate logic which can accommodate the concept of “nothing” as represented formally by *null*, and which is suitable for use with bunch theory (as well as with other theories which can generate undefined terms).

The inference rules of FOPLN can be matched 1 to 1 with those of FOPL and are identical apart from the quantifier rules.

The existence of a model for our version of bunch theory, which we have shown in [DMSZ24], and which enables any formal argument in our bunch theory to be translated to a formal argument in the language of the model, which is set theory, ensures that our theory, including its logic, are “sound”. I.e. that the conclusions reached when reasoning with our theory are “true”, in the sense that the model allows us to translate any argument from bunch theory into an equivalent argument in set theory, and from long experience we believe that formal arguments in set theory lead to true conclusions. An automatic corollary of soundness is that our logic and theory are consistent, that is there is no  $P$  for which we can prove both  $P$  and  $\neg P$ , for if there were, we would be able to use our model to translate our argument to the language of the model and prove inconsistency there, which we know (or at least firmly believe) to be impossible.

The “logic toolbox” laws of FOPL listed in the appendix can generally also be proved in FOPLN, though

<sup>3</sup>Our logical formulae are parsed according to the precedence of their connectives with the dot  $\bullet$  having an extremely low precedence. Other writers use a high precedence for this symbol. Thus we write  $\exists x \bullet x \in S \wedge P$ , rather than  $\exists x.(x \in S \wedge P)$ , which is used in notations where  $.$  is a high precedence symbol.

the proofs are not the same). However, the one-point laws of FOPL:

$$\begin{aligned} (\forall x \bullet x = E \Rightarrow P) &\Leftrightarrow P[E/x] \\ (\exists x \bullet x = E \wedge P) &\Leftrightarrow P[E/x] \end{aligned}$$

will need the premiss that expression  $E$  represents an element. We write this as  $\delta(E)$ .

## Quantifier laws of FOPL and FOPLN.

The  $\exists$ -intro rule of FOPL says that if a proposition  $P$  is true when  $x$  is replaced by some expression  $E$ , then we can say that there exists some  $x$  for which  $P$  is true. The  $\exists$ -intro rule for FOPL is:

$$\frac{HYP \vdash P[E/x]}{HYP \vdash \exists x \bullet P} \quad \exists\text{-intro}$$

However, once we are dealing with bunches an expression may represent *null*, and  $P[E/x]$  may then be vacuously true, and provide no evidence that some  $x$  exists such that  $P$  is true. The  $\exists$ -intro rule of FOPLN has an additional premiss that the expression we are citing is an element.

$$\frac{HYP \vdash \delta(E), HYP \vdash P[E/x]}{HYP \vdash \exists x \bullet P} \quad \exists\text{-intro}$$

The  $\forall$ -elim rule of FOPL says that if a proposition  $P$  is true for all values of  $x$ , then  $P[E/x]$  is true for an arbitrary expression  $E$ . The  $\forall$ -elim rule for FOPL is:

$$\frac{HYP \vdash \forall x \bullet P}{HYP \vdash P[E/x]} \quad \forall\text{-elim}$$

For FOPLN we need the additional premiss that  $E$  denotes an element. The  $\forall$ -elim rule for FOPLN is:

$$\frac{HYP \vdash \forall x \bullet P, HYP \vdash \delta(E)}{HYP \vdash P[E/x]} \quad \forall\text{-elim}$$

We will also need to change the  $\forall$ -intro and  $\exists$ -elim rules. Both of these introduce a “fresh” variable  $\alpha$ , which refers to an arbitrary element, and, since it is fresh, is not constrained by the hypotheses under which we are reasoning.

The  $\forall$ -intro rule of FOPL is:

$$\frac{HYP \vdash P[\alpha/x]}{HYP \vdash \forall x \bullet P} \quad \alpha \text{ fresh } \forall\text{-intro}$$

and the idea of the rule is that if we can prove  $P$  is true when  $\alpha$  (an arbitrary element about which we know nothing) is substituted for  $x$ , then effectively we have proved  $P$  for all  $x$

In FOPLN, to establish the conclusion of the  $\forall$ -intro rule we can make use of the additional hypothesis  $\delta(\alpha)$ .

The  $\forall$ -intro rule of FOPLN is:

$$\frac{HYP, \delta(\alpha) \vdash P[\alpha/x]}{HYP \vdash \forall x \bullet P} \quad \alpha \text{ fresh } \forall\text{-intro},$$

The  $\exists$ -elim rule of FOPL is gives a generalised form of argument by cases. Suppose  $x$  is not free in  $Q$  (which we will write as  $x \setminus Q$ ), and that according to the hypotheses under which we are reasoning we can prove  $P[\alpha/x] \Rightarrow Q$  where  $\alpha$  is fresh, and suppose also that under the same hypotheses we can prove  $\exists x \bullet P$ , then we can conclude  $Q$ .

The  $\exists$ -elim rule for FOPL is:

$$\frac{HYP \vdash P[\alpha/x] \Rightarrow Q, HYP \vdash \exists x \bullet P}{HYP \vdash Q} \alpha \text{ fresh, } x \setminus Q \quad \exists\text{-elim}$$

The  $\exists$ -elim rule for FOPLN provides an additional hypothesis  $\delta(\alpha)$  which we can use in establishing its conclusion.

The  $\exists$ -elim rule for FOPLN is:

$$\frac{HYP, \delta(\alpha) \vdash P[\alpha/x] \Rightarrow Q, HYP, \delta(\alpha) \vdash \exists x \bullet P}{HYP \vdash Q} \alpha \text{ fresh, } x \setminus Q \quad \exists\text{-elim}$$

Given that proofs of predicate calculus equivalence laws involve bound variables, and that bound variables in FOPLN range over elements. it is not surprising that the equivalence laws of FOPL are also provable in FOPLN.

## Comparison of proofs in FOPL and FOPLN

Below we give a proof of the De Morgan law  $\neg(\forall x \bullet P) \Rightarrow (\exists x \bullet \neg P)$  in both logics. In our proofs we indent after each assumption. By looking down the proof to the matching outdent we can see the point at which the assumption is discharged, and why the assumption was made.

First the proof in FOPL:

- |   |                     |
|---|---------------------|
| 1. $\neg(\forall x \bullet P)$  | assumption          |
| 2. $\neg(\exists x \bullet \neg P)$                                   | assumption          |
| 3. $\neg P[\alpha/x]$   | assumption          |
| 4. $\exists x \bullet \neg P$   | 3, $\exists$ -intro |
| 5. $P[\alpha/x]$  | 2,4, Contradiction  |
| 6. $\forall x \bullet P$  | 5, $\forall$ -intro |
| 7. $\exists x \bullet \neg P$   | 1,6, Contradiction  |
| 8. $\neg(\forall x \bullet P) \Rightarrow (\exists x \bullet \neg P)$ | 7, Deduction        |

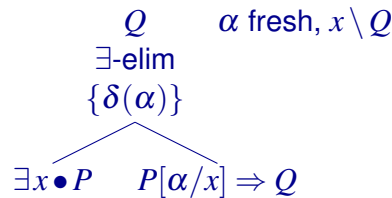
And now in FOPLN:

- |   |                       |
|---|-----------------------|
| 1. $\neg(\forall x \bullet P)$  | assumption            |
| 2. $\neg(\exists x \bullet \neg P)$                                   | assumption            |
| 3. $\delta(\alpha)$   | assumption            |
| 4. $\neg P[\alpha/x]$   | assumption            |
| 5. $\exists x \bullet \neg P$   | 3,4, $\exists$ -intro |
| 6. $P[\alpha/x]$  | 2,5, Contradiction    |
| 7. $\forall x \bullet P$  | 3,6 $\forall$ -intro  |
| 8. $\exists x \bullet \neg P$   | 1,7, Contradiction    |
| 9. $\neg(\forall x \bullet P) \Rightarrow (\exists x \bullet \neg P)$ | 8, Deduction          |

The main difference in the proofs is a little more housekeeping in the FOPLN proof to record that fresh variables denote elements. In proving the laws of our logic toolbox in FOPLN we found that this was generally the case, but that for a handful of examples the proof in FOPLN was more fastidious. An example that is quite complex to prove in FOPLN is  $(\forall x \bullet P) \Rightarrow (\exists x \bullet P)$ . These more fastidious proofs occur when we must rely on there being a non-empty domain of discourse. In the next section we prove some derived inference rules that will simplify such proofs.

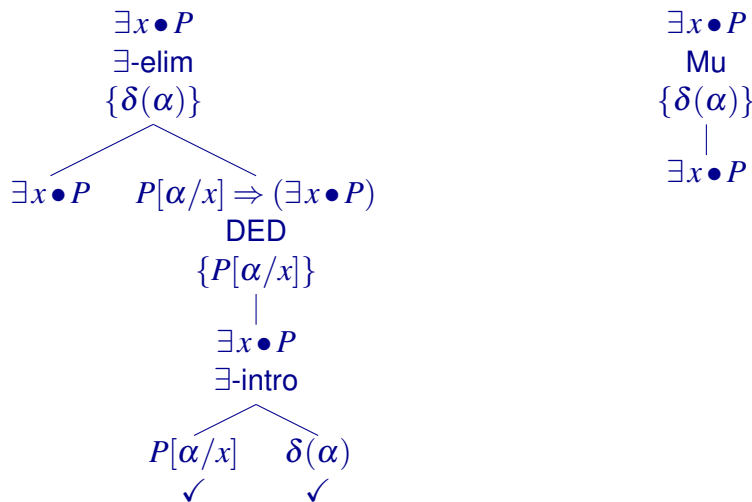
## Derived inference laws for FOPLN and backward proof trees

We describe some derived laws and their proofs in terms of “backward proof trees”<sup>4</sup> Backward proof trees are a way of presenting proofs in which the conclusion of a rule is seen as a goal to be established, and application of the rule generates the premisses of the rule as sub-goals. A tree node consists of a goal, the name of the inference rule applied to prove that goal, and possibly an additional hypothesis that can be used in establishing the sub goals. The sub-goals become child nodes. We start by expressing the  $\exists$ -elim rule of FOPLN as a tree, aka “in inverted form”.



The law tells us we can prove some goal  $Q$  by proving the sub goals  $\exists x \bullet P$  and  $P[\alpha/x] \Rightarrow Q$ . Furthermore, this proof may be carried out under the assumption  $\delta(\alpha)$ . We write proof goals above the name of the rule and any new assumption below, enclosed in curly brackets. These curly brackets are purely for emphasis and have no formal meaning.

We can extend this principle to an entire proof. We now give a derived law (on the right) and its proof (on the left).



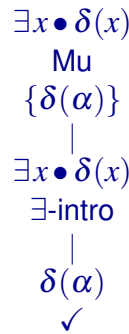
In the left-hand tree, proof goals in the non-terminal nodes are discharged by proving their sub-goals. The terminal nodes marked with a tick correspond to the hypotheses (assumptions) we are reasoning under, and thus require no further proof. The terminal node  $\exists x \bullet P$  has not been discharged, but we now have an additional hypothesis to help us.

In the right hand tree we have pruned away the nodes that we know will be discharged, obtaining a rule that tells us that in the case where a proof rule has the form  $\exists x \bullet P$  we can call on an added hypothesis  $\delta(\alpha)$ .

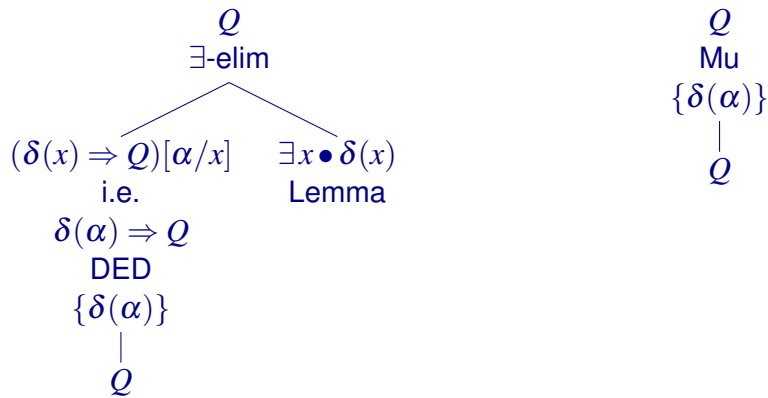
Using the derived rule Mu we can immediately prove  $\exists x \bullet \delta(x)$ , which is an explicit statement that our domain of discourse is non-empty.

<sup>4</sup>Not to be confused with “tree proofs” as used to indicate semantic tableaux.





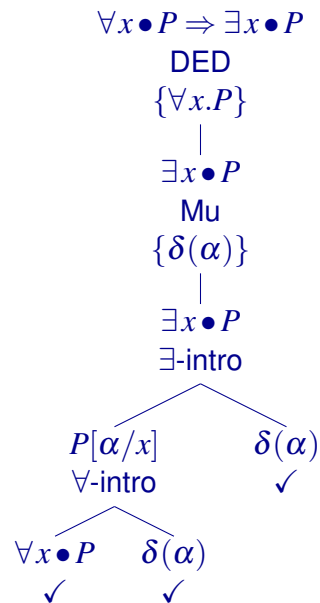
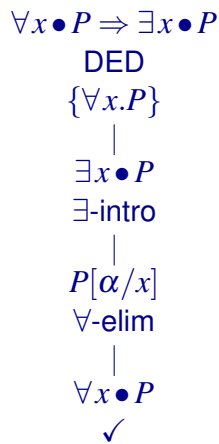
Using  $\exists x \bullet \delta(x)$  as a lemma we can prove a generalisation of the Mu rule, namely that when proving any proof goal  $Q$  we are entitled to do so under the assumption of  $\delta(\alpha)$



We illustrate the use of our derived rule Mu by giving backward tree proofs of  $(\forall x.P) \Rightarrow (\exists x \bullet P)$  in FOPL (on the left) and FOPLN (on the right).

FOPL

FOPLN



As a final proof in this section, we show that bound variables range over elements, a result we can express as  $\forall x \bullet \delta(x)$ . In the trees below we show the  $\forall$ -intro rule of FOPLN in inverted form (on the left) and the proof of  $\forall x \bullet \delta(x)$ , which applies this rule, (on the right).

$$\begin{array}{ccc}
 \forall x \bullet P & & \forall x \bullet \delta(x) \\
 \forall\text{-intro} & & \forall\text{-intro} \\
 \{\delta(\alpha)\} & & \{\delta(\alpha)\} \\
 | & & | \\
 P[\alpha/x] & & \delta(x)[\alpha/x] \\
 & & \text{i.e.} \\
 & & \delta(\alpha) \\
 & & \checkmark
 \end{array}$$

The dual result for existential quantification is  $\neg(\exists x \bullet \neg \delta(x))$ , which we immediately obtain by De Morgan.

## 5. “Mini bunch theory”: adapting set theory for use with FOPLN

We can adapt our theory of bunches so that the only non-elemental bunch present in our UofD is the null bunch. We call this “mini bunch theory”.

The first step in so doing is to remove from our language the unpack operator  $\sim$ , along with any definitions which make use of it; these include function application, bunch union, and *null* itself, which we previously obtained by unpacking the empty set.

We replace the explicit definition of *null* with axioms that characterise its properties.

$$\begin{array}{ll}
 \{null\} = \{\} & \text{null axiom} \\
 null \mapsto E = E \mapsto null = null & \text{annihilator 1} \\
 null \times S = S \times null = null & \text{annihilator 2} \\
 E \in null & \text{vacuous membership 1} \\
 null \in S & \text{vacuous membership 2}
 \end{array}$$

We need a further axiom that says an expression *E* represents an element exactly when it is non-null.

$$\delta(E) \Leftrightarrow \neg(E = null)$$

And we need a new definition of function application. For  $f \in S \leftrightarrow T$  we have:

$$f(E) \hat{=} choice(f(\{E\}))$$

where *choice* from the empty set is defined to be *null*.

Although defined for any relation, application is designed to be used with partial functions. Any more general usage requires extreme care and may sometimes lead to results that, although formally correct, may be misleading.

Our UofD is now identical to that of Abrial’s set theory with the addition of *null*, and we can use FOPLN as our underlying logic and obtain the same elimination of undefined terms as with bunch theory.

## 6. Partial functions: a free lunch?

In this section we review the problems caused by using partial functions together with classical FOPL, and we note that, as long as we have a UofD that includes *null*, these may be resolved by using FOPLN as our logic instead of FOPL. Perhaps this is the mythical free lunch that Cliff Jones has

claimed does not exist? [Jon96].

We first look at these long-standing problems in a historical context. In our formal notation, as in natural language, many objects are referred to by means of a description rather than being named directly. The problems which arise when such descriptions do not refer to anything (for example  $1/0$  or "the present king of France"), or are ambiguous, have been the source of extensive debate in mathematics and philosophy at least since the emergence of Russell's Theory of Descriptions (RTD) [Rus05, WR10].

In contrast to FOPL, RTD places partial functions at the heart of logical theory. The natural language statement "The present king of France is bald" is considered to make an existential claim that there *is* a present king of France, i.e. "The present king of France is bald" is equivalent to "There is a present king of France, and the present king of France is bald". Since there is no present king of France this statement is false, and we need give no significance to the description "the present king of France".

To follow this interpretation in Bunch Theory we must make the existential claim explicit. Given a partial function  $king\_of \in COUNTRY \rightarrow PERSON$  and a set  $BALD \subseteq PERSON$  we can represent our sentence as

$$\exists x \bullet x = king\_of(France) \wedge x \in BALD$$

which will be false since  $France \notin dom(king\_of)$  and therefore  $king\_of(France) = null$ , whereas  $x$  is a bound variable and must represent an element.

Now let us consider some limitations in current formal notations. The definite description  $\iota x \bullet P$  commonly represents the supposedly unique value of  $x$  described by predicate  $P$ . A common form of definite description is function application, noting that the term  $f(x)$  can be expressed as  $\iota y \bullet x \mapsto y \in f$ . In a formalism in which partial and total functions are a particular form of binary relation (an approach which RTD shares with a number of current formalisms including Z, B, and VDM) the term  $f(x)$  is taken to be an invalid description, aka undefined term, when there is no unique  $y$  such that  $x \mapsto y \in f$ .

In Abrial's approach, instead of undefined terms being non-denoting (as in RTD), we hold that they do denote an object of the correct type, but we do not know what it is. In some ways this resembles a formalism in which all functions are total, but we only have knowledge of their application over a limited domain. These approaches provides the advantage of a formal system compatible with FOPL. In particular we have the simple equality law  $E = E$ . For example we can prove  $1/0 = 1/0$ . This is not possible in RTD, because it entails the existential claim that there is a number that is  $1/0$ . However, as we will see, Abrial's approach means that stating  $f(x) = y$  is not enough for us to conclude  $x \in dom(f)$ . Before that however, we need to take a short detour to explain why *null* is not associated with non-termination.

## 7. Non-termination and *null*

Since the application of a function outside its domain yields *null* in our formalism and executing a function on an element outside its domain typically has unpredictable consequences, there may be a natural inclination on the part of the reader to associate *null* with non-termination. However, in our programming semantics, attempting to compute an expression with a *null* value triggers backtracking. As in B, functions are invoked within operations, and operations have pre-conditions. Non-termination results when an operation is executed outside of its pre-condition, but we discharge proof obligations

to ensure such executions never occur. If we wish to use an implementation of division for which division by zero is non-terminating, we deal with this by imposing pre-conditions on our operations to ensure that division by zero is never performed. This approach allows us to consider functions as “descriptions”. rather than recipes for computation.

## Partial functions as descriptions

In his 1905 paper “On Denoting” [Rus05] and in “Principia Mathematica” [WR10] Russell discusses partial functions using the example of kings and countries, and we continue in this tradition here.

Suppose we declare a partial function  $king\_of \in COUNTRY \rightarrow NAME$  and give the following properties:

$$king\_of(Belgium) = Philippe \wedge king\_of(Norway) = Harald \wedge \dots$$

it would seem that we have said that *Philippe* is king of *Belgium*, and so on, but this would be a premature conclusion. In Z and B for example, with a logical foundation of FOPL we cannot draw any conclusions about a partial function from knowledge of its applications unless we also know its domain. We would like  $king\_of(Belgium) = Philippe$  to imply that Belgium has a king, i.e. that  $Belgium \in \text{dom}(king\_of)$ , but this requires a law that we cannot have.

The law we would like, and which we will call “The Prize Law” is:

$$f \in X \rightarrow Y \wedge f(x) = y \Rightarrow x \mapsto y \in f$$

but incorporating such a law when using partial functions in combination with FOPL *must introduce an inconsistency* into the system.

To see this let  $f \in A \rightarrow B$  and  $f = \{ \}$ , and  $a \in A$ , so  $f(a)$  will be an undefined term.

Because the equality axiom of FOPL asserts that any term is equal to itself, we can prove  $f(a) = f(a)$ .

Now applying our desired law we match  $f(x) = y$  in the prize law to  $f(a) = f(a)$  and obtain  $a \mapsto f(a) \in f$ .

Finally we apply the  $\exists$ -intro law of FOPL to conclude that  $\exists x \bullet x \in f$ . But  $f$  was the empty set, and we have thus proved the existence of a element that belongs to the empty set. We cannot formulate the law that would allow us to conclude that Belgium has a king from our knowledge that  $king\_of(Belgium) = Philippe!$

Also, supposing that  $France \notin \text{dom}(king\_of)$  we would like to be able to prove  $\neg \exists x \bullet x = king\_of(France)$ , and this again is something we cannot do when using partial functions with classical logic, which in fact allows us to prove  $\exists x \bullet x = king\_of(France)$  in the same way we proved  $\exists x \bullet x = 1/0$  earlier.

Such considerations have led Cliff Jones and his associates to construct LPF, a three-valued logic for partial functions[BCJ84], in which predicates can be *true*, *false*, or *undefined*. With this approach we *can conclude* from  $f(x) = y$  that  $y$  is the unique value associated with  $x$  in  $f$ . However, there is a price to pay: we lose proof by contradiction and the law of the excluded middle<sup>5</sup> as well as the simple law for equality.

<sup>5</sup>As an example of a beautiful proof which uses the law of the excluded middle, we can show as follows that there are irrational numbers  $p, q$  such that  $p^q$  is rational. If  $\sqrt{2}^{\sqrt{2}}$  is rational our proof is achieved with  $p = \sqrt{2}$  and  $q = \sqrt{2}$ . Otherwise  $\sqrt{2}^{\sqrt{2}}$  is irrational and using the rule  $(a^b)^c = a^{b*c}$  we see that  $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}*\sqrt{2}} = \sqrt{2}^2 = 2$  and our proof is achieved with  $p = \sqrt{2}^{\sqrt{2}}$  and  $q = \sqrt{2}$ .

In the following section we will see how the presence of *null* in our theory allows us access to the prize law, and also allows us to prove there is no king of France.

## Partial functions and descriptions in the presence of *null*.

In the function application rules we have given for our theories an application of partial function  $f$  to an element outside its domain yields *null*. Thus given that  $a, b$  are elements:

$$f(a) = b \Rightarrow a \in \text{dom}(f)$$

From the above we can see that we can gain complete information about a partial function  $f$  from the effects obtained by applying  $f$  to all elements of the correct type, and without any prior knowledge of the function's domain. So unlike the situation in which we use FOPL and partial functions defined in set theory, specifying that

$$\text{king\_of}(\text{Belgium}) = \text{Philippe} \wedge \text{king\_of}(\text{Norway}) = \text{Harald} \wedge \dots$$

does now tell us that Belgium and Norway have kings, as well as telling us who they are.

And the result  $\text{king\_of}(\text{France}) = \text{null}$  tells us that France does not have a king.

## 8. Related work

The ability of FOPLN to resolve an important question in the theory of descriptions raises the question of whether any logic already exists in the literature that would save the same purpose. We are grateful to an anonymous reviewer for pointing us in the direction of the so-called “free logics” [No121]. This is Karel Lambert's term (1960) for a family of logics which are free of existence assumptions. They come in various flavours. They may be positive (in which case  $1/0 = 1/0$  is true), negative (in which case  $1/0=1/0$  is false) or neutral (in which case there are truth gaps). They may be classical or intuitionistic [Sco79], their empty terms may be equal or may be differentiated; they may allow an empty domain of discourse; they may be first order or higher order, bi-valent or multi-valent. A positive bi-valent free logic with a non empty domain of discourse whose undefined terms are all equal would appear to allow us to formulate, under appropriate definedness conditions for  $f, a, b$ , the desirable function application rule,  $f(a) = b \Rightarrow a \mapsto b \in f$ . However, the definition of free logics given in [No121] involves not only their possible inference rules but also the forms of domain of discourse required for their interpretation, and these do not suggest that free logic would be a suitable underpinning for bunch theory. Our theory of bunches eliminates the referential failure tolerated by free logics, replacing such failure by *null*, which has a denotation and is perfectly well defined.

A second area of related work concerns the defensive measures that can be taken to ensure well-definedness when using partial functions in conjunction with classical logic. Automated support for such checks was a major element of the RODIN project [ABHV06], which developed comprehensive tool support for Event B.

## 9. Conclusions and Future Work

Bunch theory is a reformulation of set theory which separates collection and packaging into separ-

ate activities, and thus allows us to reason about “unpacked” collections. This is useful where classical set theory introduces unwanted structure. In our research we have used it to unify non-determinism, preferential choice, probabilistic choice, and backtracking, within a simple theory of program semantics.

Perhaps the most radical change introduced by bunch theory is the ability to conceptualise “nothing”, and name it as the constant *null*.

Bunch theory, and other theories which admit *null*, are not consistent with classical logic (FOPL). We have described our revision of FOPL to FOPLN, which resolves this issue, and provides a suitable logic to act as a foundation for bunch theory. We introduce “Mini bunch theory”, whose domain of discourse consists of elements and *null*, to make FOPLN suitable for use with other formalisms, such as Z and B, that make use of partial functions. FOPLN is very similar to FOPL, in that they share propositional logic, and the logical identities provable in FOPL are generally provable in FOPLN, with only the one-point rules requiring an extra definedness condition.

This paper does not include a formal argument for the soundness of FOPLN with respect to bunch theory. For that we refer the reader to the denotation model we have formulated in [DMSZ24]. A proof of soundness for Mini bunch theory is left as future work.

Bound variables in bunch theory range over elements. One nice feature of FOPLN is that rather than being a design decision in our theory, this is a result which we can prove in our logic.

We claim that FOPLN eliminates the well known problems that are present when classical logic is used along with partial functions. In theories that support *null*, full information about a partial function can be obtained from the results of function applications *without* having prior knowledge of the function’s domain.

## 10. Acknowledgements

We warmly thank Eric Hehner for extended conversations on bunch theory, John Nolt for his responses to our queries on free logic, and an anonymous referee who provided invaluable comments.

## References

- [ABHV06] J-R Abrial, M Butler, S Hallerstede, and L Voisin. An open extensible tool environment for Event-B. In *ICFEM 2006*, number 4260 in Lecture Notes in Computer Science, 2006.
- [BCJ84] H Barringer, J H Chang, and C B Jones. A logic covering undefinedness in program proofs. *Acta Informatica* 21, 1984.
- [CAD96] CADE96. *CADE-13 Workshop on Mechanisation of Partial Functions, held at Rutgers University, New Brunswick NJ USA.*, July 1996. Papers available from <http://WWW.cs.bham.ac.uk/~mmk/cade96-partiality>.
- [DFM<sup>+</sup>23] S E Dunne, J F Ferreira, A Mendes, C Ritchie, W J Stoddart, and F Zeyda. bGSL: An imperative language for specification and refinement of backtracking programs. *Journal of Logical and Algebraic Methods in Programming*, 130, 2023.
- [DMSZ24] S E Dunne, C Mu, W J Stoddart, and F Zeyda. Bunch theory: axioms, logic, applications and model. *Journal of Logical and Algebraic Methods in Programming*, 140, 2024.

- [Ewa19] William Ewald. The Emergence of First-Order Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2019 edition, 2019.
- [Heh93] E C R Hehner. *A Practical Theory of Programming*. Springer Verlag, 1993. Latest version available on-line.
- [Heh23] E C R Hehner. *A Practical Theory of Programming, 2023 edition*. University of Toronto, 2023. Available at <https://www.cs.toronto.edu/hehner/aPToP/aPToP.pdf>.
- [HS24] Wilfrid Hodges and Thomas Scanlon. First-order Model Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Spring 2024 edition, 2024.
- [Jon96] C B Jones. TNSTAAFL (with partial functions). 1996. In [CAD96].
- [Lee93] René Leermakers. *The functional treatment of parsing*. Springer Verlag, 1993. 2nd edition 2012.
- [MB01] J Morris and A Bunkenburg. A theory of bunches. *Acta Informatica*, 37(8), 2001.
- [NH92] T S Norvell and E C R Hehner. Logical specifications for functional programs. In R S Bird, C C Morgan, and J C P Woodcock, editors, *Mathematics of Program Construction*. Springer Verlag, 1992.
- [Nol21] John Nolt. Free Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.
- [PH99] R F Paige and E C R Hehner. Bunches for Object Oriented, Current and Real Time Specification. In J M Wing, Woodcock J, and Davies J, editors, *FM99 vol 1*, number 1708 in Lecture Notes in Computer Science. Springer Verlag, 1999.
- [Rus05] Bertrand Russell. On denoting. *MIND*, 14:479–493, 1905.
- [SCD24] W J Stoddart, Mu C, and S Dunne. Logic proofs in FOPL and FOPLN. arXiv:5508652 [cs.LO], 2024.
- [Sco79] Dana Scott. *Identity and existence in intuitionistic logic*, pages 660–696. Springer Berlin Heidelberg, Berlin, Heidelberg, 1979.
- [Tur36] Alan M Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.
- [WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1910.

## Appendix

### 11. Some laws provable in both FOPL and FOPLN

For proofs of these results see [SCD24].

De Morgan

1.  $(\neg \forall x \bullet P) \Leftrightarrow (\exists x \bullet \neg P)$
2.  $(\forall x \bullet \neg P) \Leftrightarrow \neg (\exists x \bullet P)$

Change of order of bound variables, assuming  $P$  admits  $y$  for  $x$ .<sup>6</sup>

3.  $(\forall x \bullet \forall y \bullet P) \Leftrightarrow (\forall y \bullet \forall x \bullet P)$
4.  $(\exists x \bullet \exists y \bullet P) \Leftrightarrow (\exists y \bullet \exists x \bullet P)$

Splitting

5.  $(\forall x \bullet P \wedge Q) \Leftrightarrow (\forall x \bullet P) \wedge (\forall x \bullet Q)$
6.  $(\exists x \bullet P \vee Q) \Leftrightarrow (\exists x \bullet P) \text{ or } (\exists x \bullet Q)$
7.  $(\exists x \bullet P \wedge Q) \Leftrightarrow (\exists x \bullet P) \wedge (\exists x \bullet Q)$

Assuming  $x$  is not free in  $Q$

8.  $(\forall x \bullet P) \wedge Q \Leftrightarrow \forall x \bullet (P \wedge Q)$
9.  $(\forall x \bullet P) \vee Q \Leftrightarrow \forall x \bullet (P \vee Q)$
10.  $(\exists x \bullet P) \wedge Q \Leftrightarrow \exists x \bullet (P \wedge Q)$
11.  $(\exists x \bullet P) \vee Q \Leftrightarrow \exists x \bullet (P \vee Q)$
13.  $\forall x \bullet (Q \Rightarrow P) \Leftrightarrow Q \Rightarrow \forall x \bullet P$
14.  $\exists x \bullet (P \Rightarrow Q) \Leftrightarrow (\forall x \bullet P) \Rightarrow Q$
15.  $\forall x \bullet (P \Rightarrow Q) \Leftrightarrow (\exists x \bullet P) \Rightarrow Q$
16.  $\exists x \bullet (Q \Rightarrow P) \Leftrightarrow Q \Rightarrow \exists x \bullet P$

Change of bound variable name

16.  $(\forall x \bullet P) \Leftrightarrow (\forall y \bullet P[y/x])$
17.  $(\exists x \bullet P) \Leftrightarrow (\exists y \bullet P[y/x])$

Monotonicity

18.  $(\forall x \bullet P \Rightarrow Q) \Leftrightarrow ((\forall x \bullet P) \Rightarrow (\forall x \bullet Q))$
19.  $(\forall x \bullet P \Rightarrow Q) \Leftrightarrow ((\exists x \bullet P) \Rightarrow (\exists x \bullet Q))$

Equivalence

20.  $(\forall x \bullet P \Leftrightarrow Q) \Rightarrow ((\forall x \bullet P) \Leftrightarrow (\forall x \bullet Q))$
21.  $(\forall x \bullet P \Leftrightarrow Q) \Rightarrow ((\exists x \bullet P) \Leftrightarrow (\exists x \bullet Q))$
22.  $(\exists x \bullet P \Rightarrow Q) \Leftrightarrow ((\forall x \bullet P) \Rightarrow (\exists x \bullet Q))$

one-point rules, for FOPLN we require the premiss  $\delta(E)$

23.  $(\forall x \bullet x = E \Rightarrow P) \Leftrightarrow P[E/x]$
24.  $(\exists x \bullet x = E \wedge P) \Leftrightarrow P[E/x]$

<sup>6</sup> $P$  admits  $y$  for  $x$  if every free occurrence of  $x$  in  $P$  becomes a free occurrence of  $y$  in  $P[y/x]$ .



## 12. Inference rules for FOPLN

### Propositional logic rules

$$\frac{HYP \vdash P, HYP \vdash Q}{HYP \vdash P \wedge Q} \text{ CONJ or } \wedge\text{-intro}$$

$$\frac{HYP \vdash P \wedge Q}{HYP \vdash P} \quad \frac{HYP \vdash P \wedge Q}{HYP \vdash Q} \text{ AND or } \wedge\text{-elim}$$

$$\frac{HYP \vdash P, HYP \vdash P \Rightarrow Q}{HYP \vdash Q} \text{ MODUS PONENS}$$

$$\frac{HYP \vdash P \Rightarrow Q, HYP \vdash Q \Rightarrow P}{HYP \vdash P \Leftrightarrow Q} \text{ EQUIV or } \Leftrightarrow\text{-intro}$$

$$\frac{HYP \vdash P \Leftrightarrow Q}{HYP \vdash P \Rightarrow Q} \quad \frac{HYP \vdash P \Leftrightarrow Q}{HYP \vdash Q \Rightarrow P} \text{ IMPLIES or } \Leftrightarrow\text{-elim}$$

$$\frac{HYP \vdash P}{HYP \vdash P \vee Q} \quad \frac{HYP \vdash Q}{HYP \vdash P \vee Q} \text{ OR or } \vee\text{-intro}$$

$$\frac{HYP \vdash P \Rightarrow R \quad HYP \vdash Q \Rightarrow R \quad HYP \vdash P \vee Q}{HYP \vdash R} \text{ CASES or } \vee\text{-elim}$$

$$\frac{HYP, P \vdash Q}{HYP \vdash P \Rightarrow Q} \text{ DED} \quad \frac{}{HYP, P \vdash P} \text{ INHYP}$$

$$\frac{HYP, P \vdash Q, HYP, P \vdash \neg Q}{HYP \vdash \neg P} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{ CONTR}$$

$$\frac{HYP, \neg P \vdash Q, HYP, \neg P \vdash \neg Q}{HYP \vdash P}$$

$$\frac{}{\vdash true} \text{ TRUE} \quad \frac{}{\vdash \neg false} \text{ FALSE}$$

*HYP* represents a bunch of assumptions under which we are reasoning. The “sequent”  $HYP \vdash P$  says that under the assumptions in *HYP*, *P* can be proved. An inference rule consists of one or more sequents above a line and one sequent below the line. It says that proof of the sequents above the line provides a proof of the sequent below the line. These rules are identical to those of FOPL.

**Predicate logic and equality proof rules.**

$$\frac{HYP, \delta(\alpha) \vdash P[\alpha/x]}{HYP \vdash \forall x \bullet P} \quad \alpha \text{ fresh } \forall\text{-intro,}$$

$$\frac{HYP \vdash \forall x \bullet P, HYP \vdash \delta(E)}{HYP \vdash P[E/x]} \quad \forall\text{-elim}$$

$$\frac{HYP \vdash \delta(E), HYP \vdash P[E/X]}{HYP \vdash \exists x \bullet P} \quad \exists\text{-intro}$$

$$\frac{HYP, \delta(\alpha) \vdash P[\alpha/x] \Rightarrow Q, HYP, \delta(\alpha) \vdash \exists x \bullet P}{HYP \vdash Q} \quad \alpha \text{ fresh, } x \setminus Q \quad \exists\text{-elim}$$

The equality rules are identical to those of FOPL.

$$\frac{}{\vdash E = E} \quad \text{EQUALS}$$

$$\frac{HYP \vdash E = F, HYP \vdash P}{HYP \vdash P[E/F]} \quad \text{RT or LEIBNIZ}$$

$$\frac{HYP \vdash P \Leftrightarrow Q, HYP \vdash R}{HYP \vdash R[P/Q]} \quad \text{REP}$$

**Notes**

RT = Referential transparency (of values).

REP = Replacement Theorem, a meta-theorem which asserts the referential transparency of predicates.

Fresh variables introduced by proof rules take arbitrary values.

$P[E/F]$  is predicate  $P$  with each occurrence of the expression  $F$  replaced by  $E$ .  $R[P/Q]$  is predicate  $R$  with each occurrence of the sub-predicate  $Q$  replaced by  $P$ . Substitutions should not cause variable capture.

# Linear Steganography, Ring Homomorphisms, and Homomorphic Encipherment

Theodore Murray Abelson

[theodore.murray.abelson@gmail.com](mailto:theodore.murray.abelson@gmail.com)

## 1. Introduction

I was reflecting upon something a friend (let's call him David) had said quite a few years back. He said that he didn't quite see what all the fuss was about with all this cryptography malarkey – he said why don't we just add a number to encrypt and then subtract that number again to decrypt. I told him rather smartly that this could be painfully naïve and simply would not do in actual practice.

For some odd reason, this brief exchange stuck with me all this time. The fellow wasn't usually given to making wildly silly suggestions. After all, he did say all of that with something of a sly grin and a knowing twinkle in the eye, it must be said. At the time, I thought David was just making an idle joke – and I left it at that.

However, I was prompted again to think about my friend's remark some time back, while I was reading about a fairly exotic topic calling itself "Homomorphic Encryption" [1]. What was known about that topic seemed offputtingly arcane, complicated and awkward to implement. However, nothing came to mind apart from David's sly remark, and there I left it for a few months. Nevertheless, it continued to nag at me – and it suddenly dawned on me that there might just be something interesting there after all.

After a bit of scribbling, it seemed that I could see how a mild generalisation of David's suggestion might provide something akin to Homomorphic Encryption as I understood it. A delightful aspect of what I was cooking up was it's refreshing simplicity and that it resembled the children's game known as "Think of a Number" This is where you are asked to 'think of a number', and then to perform some long-winded sequence of operations, only to end up with, depending on the trick, either some particular number told in advance, or alternatively, the number you first thought of!

Of course, all that's going on there is algebra, pure and simple, just as it is here as well. However, before going any further, it's helpful to put a bit more meat on the bones of David's idea.

## 2. Linear Steganography

Here we deal with a couple of the main ideas, one of which is very likely to be familiar to you, the use of linear functions, but the other idea, steganography, is probably much less so. The spark of David's idea involves an interplay of both of these notions working together. At its core is the idea that basic unadorned addition is one of the simplest possible invertible arithmetic operation useful for enciphering anything.

However, there does need to be a bit more to it than that! To be useful, any enciphering needs to be invertible, but as said earlier, just adding a simple constant value is far too naïve in practice.

At the very least, something having more than one parameter would naturally provide a bit more of a challenge. To that end, consider here enciphering numbers by instead using a more general linear function such as:

$$y = 4 * z + 27$$

might be more effective, as there would then be two parameters (i.e. gradient and offset) needed to encode and decode the message. Decoding involves applying its inverse, also a linear function:

$$z = (y - 27) / 4$$

Traditionally speaking, linear functions like the above are generally regarded as an anathema amongst professional cryptographers and the like. Linear functions are typically to be avoided at all costs for encryption purposes. The reason for using them will become much clearer in due course – suffice it to say that:

*In general, linear functions should **not** be used within secure encryption methods!*

## 2.1 What is Steganography?

Quite often, steganography is thought of as perhaps a trivial way of *disguising* or *concealing* information by "hiding in plain sight". However, the dictionary definition of steganography suggests it has a more subtle rôle, in which steganography is used to *conceal* the very presence of a message within some other data. This is rather like writing in invisible ink, for one example, or by using the low-order bits to encode messages in digital streaming content, for another.

A particularly nice illustration of "hiding in plain sight" was told to me by a friend. Back in the 70s, he and his family had moved into a new area and unfortunately they got burgled – twice, in quick succession! My friend fairly promptly installed improved security measures, including a burglar alarm needing to be disarmed through the use of a key within 30 seconds of entry. However, to mitigate the risk of younger members of the family losing the alarm key, my friend hung a bunch of similar keys nearby, only one of which disarmed the alarm. Of course, all the family knew which key was the right one to use – and no further burglaries occurred thereafter!

There is a general impression that steganography is somehow particularly weak and therefore shouldn't ever be used. This is somewhat misleading – it all very much depends on what it is that needs protecting, how steganography is to be used and what other techniques steganography is combined with. For example, it is quite acceptable to combine the use of steganographic techniques with some stronger forms of encryption, to disguise or *camouflage* the presence of a more-strongly encrypted message.

## 2.2 Formally defining Linear Steganography

In this section. *Linear Steganography* is formally introduced as a *reversible* linear process for obfuscating and disguising (numerical) information. This is said to be *linear* because it just uses ordinary linear functions, and it's *steganography* because numbers are used to disguise and conceal other numbers<sup>1</sup>.

Our linear enciphering and deciphering functions are as follows:

$$\begin{aligned} \text{Linear\_enc}(\alpha, \beta) &: \mathbb{Z} \rightarrow \mathbb{Z} \\ \text{Linear\_enc}(\alpha, \beta) &\hat{=} \lambda z : \mathbb{Z} \cdot \alpha * z + \beta \end{aligned}$$

where  $\alpha \neq 0$ , and  $\beta$ , are both integers. Let's also define the integer subset  $Rcode(\alpha, \beta)$  to be the *range* of function  $\text{Linear\_enc}(\alpha, \beta)$  by:

$$Rcode(\alpha, \beta) \hat{=} \{ \alpha * z + \beta \in \mathbb{Z} \mid z \in \mathbb{Z} \}$$

Clearly,  $Rcode(\alpha, \beta) \subseteq \mathbb{Z}$  which implies that  $\text{Linear\_enc}(\alpha, \beta) : \mathbb{Z} \rightarrow Rcode(\alpha, \beta)$ .

Because linear functions like this are one to one and onto their range,  $Rcode(\alpha, \beta)$ , there is an *inverse* function,  $\text{Linear\_dec}(\alpha, \beta)$ , defined as follows:

$$\begin{aligned} \text{Linear\_dec}(\alpha, \beta) &: Rcode(\alpha, \beta) \rightarrow \mathbb{Z} \\ \text{Linear\_dec}(\alpha, \beta) &\hat{=} \lambda z : Rcode(\alpha, \beta) \cdot \frac{(z - \beta)}{\alpha} \end{aligned}$$

The inverse  $\text{Linear\_dec}(\alpha, \beta)$  is defined over the integer subset  $Rcode(\alpha, \beta)$  – but seemingly defined here using *division*, an operation borrowed from the rationals,  $\mathbb{Q}$ . This all works out, of course, because when  $z \in Rcode(\alpha, \beta)$ , then  $\alpha$  does exactly divide into  $(z - \beta)$ , by definition. A perhaps less incongruous way of defining this function is to use *integer division* (`div`) instead and then define the function  $\text{Linear\_dec}(\alpha, \beta)$  by this equation:

$$\text{Linear\_dec}(\alpha, \beta) = \lambda z : Rcode(\alpha, \beta) \cdot (z - \beta) \text{ div } \alpha$$

This describes exactly the same function as before, albeit expressed without directly appealing to the operation of division over the rationals. Note that the definition above could equally provide a function that is well-defined over all integers, not merely the subset  $Rcode(\alpha, \beta)$ .

---

<sup>1</sup>Although not discussed further here, this approach could just as easily be applied to reals (i.e. floating point) as to integers.

### 2.2.1 Worked example

To briefly illustrate all of this, consider the following worked example.

As is traditional, Alice wishes to send Bob a message - in the form of a number, naturally<sup>2</sup>. So, let's imagine that Alice and Bob have already agreed upon the parameters  $\alpha = 14$  and  $\beta = -9$  - they don't need to be positive.

Suppose that the number Alice wishes to send to Bob is 23. Alice calculates the encoded number as  $Linear\_enc(14, -9)(23) = (14 * 23) - 9 = 313$ . Alice then concocts a seemingly innocuous text message containing the encoded number 313, sending it to Bob. He then reads the message and then extracts and decodes the number Alice sent by:  $Linear\_dec(14, -9)(313) = (313 + 9) / 14 = 23$ . In this way, Alice and Bob can then exchange data, preferably varying the key parameters  $\alpha$  and  $\beta$  every time they do so, according to some scheme agreed in advance. . It is worth noting that there is nothing particularly unusual about the number 313 on its own - nothing to suggest it carries any different meaning in context.

## 2.3 Encoding vs Enciphering vs Encrypting

What's the difference between the encoding, enciphering and encrypting of information? The broad distinctions taken here are as follows:

**Encoding:** This involves taking information expressed in one form and reexpressing that data (or parts thereof) in an alternative form for the purpose of communication and/or processing. As such, virtually all processing of data involves encoding in one form or another.

**Encrypting:** This is clearly a form of encoding. However, the intent of encryption is to ensure some form of secure information sharing within a select grouping of two or more participants only. This means that the encryption process has to prevent access to the encrypted information by anyone not within the select group. Many types of encryption schemes are known (e.g. symmetric, asymmetric public-key, etc.) but all necessarily rely on members of the select grouping knowing some private credential information in one form or another and using that to access and exchange information within the grouping. It is therefore access to their private information that determines if access to the encrypted content is successful.

Conceptually, every encryption scheme can be split into *two* processes having distinct responsibilities:

**Data encipherment:** The process of rendering given information into an opaque form which then needs some additional (so-called *key*) information to recover the original information as given.

---

<sup>2</sup>The characters Alice, Bob and Mallory commonly appear in the literature as idealised protocol participants, or *principals*, that have particular rôles within security protocols. Alice and Bob stand for participants wishing to exchange messages securely and privately between themselves, and Mallory is someone wanting to intercept, read and potentially even *insert* messages, by masquerading as one of the other participants. The intention is to have protocols that enable both Alice and Bob to communicate privately, while at the same time frustrating Mallory's aims as far as possible.

**Key management:** The process with responsibility for controlling the key information needed to access the enciphered information.

**Enciphering:** This is also a form of encoding, of course. As said above, it is the process of opaquely encoding given information using some other additional (key) information in such a way that the original information can't be recovered without using this key information. Another way of thinking of encipherment is that it is just like encryption, but without any key management!

Although some may consider the distinctions here somewhat subtle, discussion here mainly considers *enciphering* as opposed to *encrypting* data.

### 2.4 More practical matters

Clearly this method is not particularly 'efficient', in the sense that the encoded output is typically larger in terms of digits than the given input. If  $size(n)$  gives the number of bytes needed to represent the number  $n$  (broadly  $1 + floor(log_{256} n)$ ) so that if  $n < B < 2 * n$  and  $0 < A \leq 255$  then:

$$size(A * n + B) \approx 1 + size(n) \text{ bytes.}$$

Furthermore, the parameter  $B$  should also be so that  $size(B) \approx size(n)$  and have various one's sprinkled through its binary representation to 'disrupt' the plain-text,  $n$ , by forcing many carries.

This means that to encode a seven-byte number, one needs eight bytes. To encode a short text string of 15 ASCII characters, one needs 16 bytes (or 128 bits).

For example, to send the 12 character string "Hello World!" with key parameters  $\alpha = 131$  and  $\beta = 21343045860346512224743294573$ , first convert the string into its constituent 12 hex bytecodes:

48 65 6c 6c 6f 20 57 6f 72 6c 64 21

which naturally corresponds to the hex number 0x48656c6c6f20576f726c6421, and in base 10 is: 22405534230753928650781647905. Enciphering this value with the above key parameters gives the 13 byte output as (base 16):

25 50 dd 09 3f 53 e4 33 c0 14 ae c3 50

which in base 10 is: 2956468030089111165477139170128.

### 2.5 A tougher variant?

Consider the following variant having instead *three* integer key parameters  $\alpha \neq 0$ ,  $\beta$  and  $\gamma$ :

$$\lambda_z : \mathbb{Z} \cdot A * (z + B) + C$$

Some might wonder if this is any harder to analyse than before.

On the contrary, basic algebra reveals that, instead, this is trivially *equivalent* to the original two-parameter method, and is merely a bit more involved to calculate, while providing no additional difficulty for analysis:

$$\alpha * (z + \beta) + \gamma = \alpha * z + (\alpha * \beta + \gamma)$$

### 3. Homomorphic Encipherment

What prompted my interest in this was reading about something called “*Homomorphic Encryption*”<sup>[1]</sup>.

The basic concept seems straightforward enough – as illustrated in Figure 1. The input data is first enciphered *locally* in such a way that a third-party remote service can perform some meaningful data operation upon the *enciphered* data, all *without* first deciphering that data. The resulting data from the operation can then be returned to the local client, at which point the returned data is locally decoded.

The important point is that the remote service appears to have operated upon the encoded input as if it had been given in the clear – and yet the remote service remains completely unaware of what the actual input (or indeed, the output) was! At no point does the remote service have access to the actual input or output values.

From a more mathematical point of view, this process follows the somewhat idealised commuting diagram as shown in Figure 2, where the encoding and decoding functions, *Enc* and *Dec* perform translations between a given input domain, *A*, and some data domain *B*. These morphisms are composed<sup>3</sup> with the operation *Opn* acting upon the data domain so as to *induce* a corresponding operation, *Opn'* acting upon the input domain.

#### 3.1 Functional Conjugation

Seeing this diagram, this strongly resonated with an idea familiar from elementary Group Theory, and elsewhere – it is the notion of a *conjugate* operation. We can use the pair of morphisms *Enc* and *Dec* so that the operation *Opn'* over the input domain is defined in terms of *Opn* over the data domain, where:

$$\begin{aligned} id_A &= Dec \circ Enc \\ Opn' &= Dec \circ Opn \circ Enc \end{aligned}$$

In other words, the operation *Opn'* is defined as a *functional conjugate* of an operation *Opn* where a conjugate operation firstly maps input data using morphism *Enc* into the domain of *Opn*, and then applies operation *Opn*. The result is then mapped back to the original space using the morphism *Dec* to obtain the output. In this way, functional conjugates can be viewed as representing *isomorphic 'clones'* of other operations.

Functional conjugation provides a powerful technique for reusing operations defined over one domain and then applying appropriate conjugates over another. Conjugation commonly arises both in mathematics and in programming, and can be characterised by the phrase:

Map – Operate – Unmap

This idea crops up again later on.

---

<sup>3</sup>Recall that functional composition is defined as follows:  $(g \circ f) \hat{=} \lambda x \cdot g(f(x))$  i.e. the function *f* is applied first, followed by applying *g*.



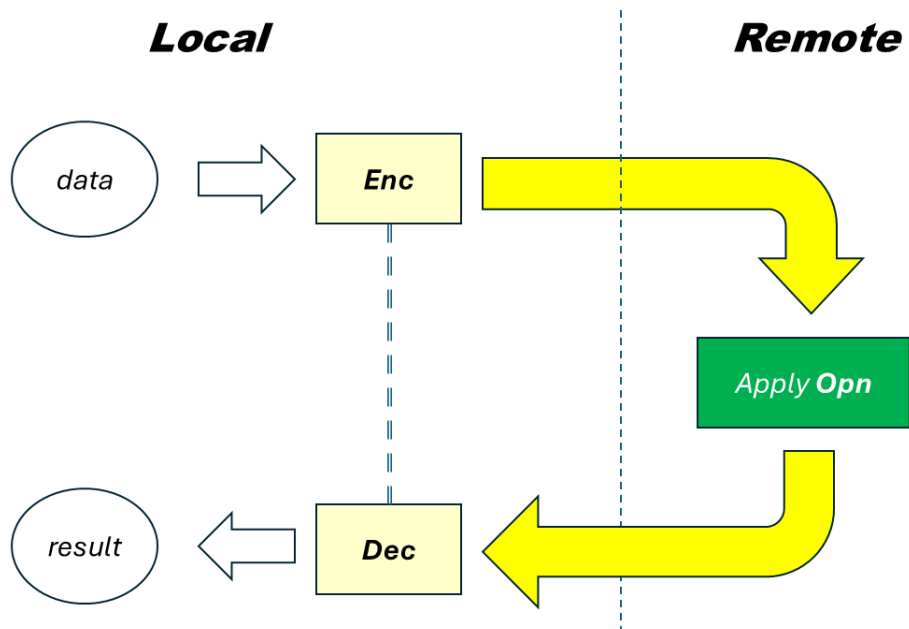


Figure 1: Outline of Homomorphic Encipherment

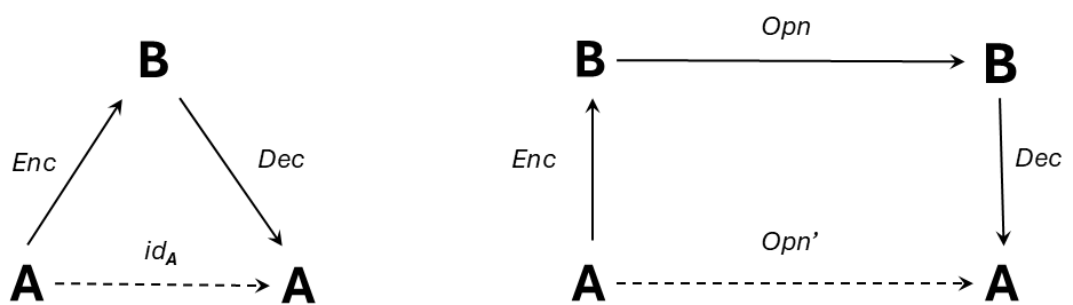


Figure 2: Homomorphic Encipherment in the form of a commuting diagram

### 3.2 Discussion

To be clear, this all struck me as not only a somewhat strange thing to want to do, it also seemed quite unclear to me how it could even be done at all! After all, arbitrarily modifying or interfering with enciphered data in any way generally completely mangles it beyond all possible use, preventing useful deciphering.

Fortunately, the article mentions *privacy-preserving calculations* as a potentially motivating application area, which provides a credible motive for implementing what would otherwise seem quite unlikely to succeed and rather pointless.

However, there could also be some advantages in terms of a *separation of concerns*. For instance, both the local sender and the remote service each benefits from this privacy – the local client keeps their data private while gaining the benefit of whatever the remote service provides computationally. Conversely, the remote service offers a service publicly without needing to disclose specific details of how that service operates internally. This enhances trust for the local client since the remote server would gain nothing by acting incorrectly or dishonestly, as the remote server has no idea of what the data is that they are processing or even what it signifies.

In this way, the use of homomorphic enciphering provides a means in which both parties can retain confidential access to their own proprietary content, whilst at the same time enabling an efficient service to operate between them.

Having seen that there might be perfectly sensible reasons to want provide this kind of service, there only remains the difficulty of seeing how to actually do it! This is the topic of the next section.

## 4. A homomorphic enciphering scheme

I now turn to formally introducing a scheme providing a simple form of homomorphic enciphering.

Before going any further, I should point out that the fully general homomorphic enciphering problem is considered to be extremely challenging and is not addressed here. The techniques that are proposed and illustrated here instead provide for *restricted* application which nevertheless includes a broadly useful class of operations.

Recalling Figure 1, we need to find a pair of enciphering / deciphering functions (*Enc*, *Dec*) which commutes appropriately with some suitable class of operations. The key point is that the class of operations somehow needs to be *compatible* with the enciphering and deciphering methods used.

Now, from the earlier discussion, Linear Steganography seems like a promising candidate for providing enciphering and deciphering. Furthermore, it seems natural to take the class of operations as the *polynomials having integer coefficients* – that is, functions having the structural form, for integer  $N \geq 1$ :

$$\lambda_z : \mathbb{Z} \cdot \sum_{i=0}^N c_i * z^i$$

where the coefficients  $c_i$  are integers, and  $c_N \neq 0$ .

The following convention is adopted for the remainder of this article: standard (unenciphered) input values are denoted by  $n$  and are known here as *natural inputs*, whereas  $x$  typically denotes enciphered input values.

#### 4.1 The problem

However, it turns out that there is a serious problem with this suggested approach to homomorphic enciphering, since it doesn't work out quite as naïvely imagined above.

The issue is that combining a linear enciphering with a polynomial,  $p$ , in this way doesn't immediately yield a result that can be systematically related back, or 'deciphered', to the required value,  $p(n)$ , for natural input  $n$ .

To illustrate algebraically what goes wrong here, let's consider the following example. Suppose we take the data operation to be a quadratic polynomial called  $p$ , such as:

$$p \hat{=} \lambda z : \mathbb{Z} \cdot a * z^2 + b * z + c$$

The idea is that, given a natural input<sup>4</sup>  $n$ , then the local client enciphers that as  $x = \alpha * n + \beta$ , for some  $\alpha$  and  $\beta$ , which is then sent to the remote service for further processing.

However, when the remote service applies polynomial  $p$  to the enciphered value,  $x = \alpha * n + \beta$ , here is algebraically what will be computed<sup>5</sup>:

$$\begin{aligned} p(x) &= a * x^2 + b * x + c \\ &= a * (\alpha * n + \beta)^2 + b * (\alpha * n + \beta) + c \\ &= a * (\alpha^2 * n^2 + 2\alpha\beta * n + \beta^2) + b\alpha * n + b\beta + c \\ &= a\alpha^2 * n^2 + 2a\alpha\beta * n + a\beta^2 + b\alpha * n + b\beta + c \\ &= a\alpha^2 * n^2 + (2a\alpha\beta + b\alpha) * n + (a\beta^2 + b\beta + c) \end{aligned}$$

This shows how the polynomial's coefficients  $a, b$  and  $c$ , get combined and "muddled up" with the enciphering parameters  $\alpha$  and  $\beta$ , in a complicated way that isn't immediately separable. This means that this calculation is not easily related back algebraically to the hoped-for result.

What is needed locally is some way to decipher and extract the desired result from output provided by the remote server – which forms the subject of the next section.

#### 4.2 Using modular arithmetic for Homomorphic Encipherment

Fortunately, there is a way to rescue this approach by judiciously using *modular arithmetic* as part of the *deciphering* process.

<sup>4</sup>To simplify the account, the input  $n$ , as well as all the coefficients for  $p$ , are here assumed to be non-negative to ensure that the output value is non-negative. In section 5.1, it is discussed how such limitations can be mitigated.

<sup>5</sup>For the sake of simplicity, clarity and readability, some of the multiplies symbols have been suppressed.

Making this work involves the local client making a savvy choice of key parameters,  $\alpha$  and  $\beta$ , for enciphering the natural input  $n$ . Crucially, the key step is for the local client to first choose a (large) *modulus* value,  $M > 0$ . This value is initially used in calculating the key parameters,  $\alpha$  and  $\beta$ , and is also used later in the deciphering process.

Using this modulus value,  $M$ , the value returned from the remote service can be deciphered by exploiting the homomorphic properties of modular arithmetic. Accordingly, it is important that the remote service neither needs nor requires knowledge of the modulus chosen by the local client.

However, in order for that choice of modulus to be sensibly made by the local client, there are some constraints on  $M$  to be met that involves knowing something further about the polynomial. This is discussed in the following section.

**4.2.1 Constraints on selecting the modulus,  $M$**

$p$  be a polynomial with integer coefficients such that it is strictly positive over some given range of (natural) inputs:  $a \leq n \leq b$ , where  $a < b$ . This range of natural input values is selected by the remote service, based on what is known about the polynomial  $p$ .

The choice of modulus  $M$  by the local client is then made, informed by characteristics of the polynomial  $p$ , which in turn is only known specifically by the remote service. So, the local client is given the following information by the remote service:

Over the (natural) input range from  $a$  to  $b$ , the polynomial is strictly positive and its maximum value is less than  $R > 0$ .

$$\forall n \in \mathbb{Z} \cdot a \leq n \leq b \Rightarrow 0 < p(n) < R$$

Using this information, the local client can then securely select modulus  $M$  such that:  $R < M$ .

It is perhaps worth emphasising that the value  $R$  given by the remote service to the local client does not need to be in any sense "tight" – its just some value larger than any attained by the polynomial  $p$  over the natural input range:  $a \leq n \leq b$ .

**4.2.2 Selecting the key parameters,  $\alpha$  and  $\beta$**

Now that a modulus  $M$  has been selected, the key parameters,  $\alpha$  and  $\beta$ , can then be selected so that:

$$\begin{aligned} \alpha &\equiv 1 && (\text{mod } M) \\ \beta &\equiv 0 && (\text{mod } M) \end{aligned}$$

That is, by putting  $\alpha \hat{=} 1 + k * M$  and  $\beta \hat{=} h * M$  for some non-zero *co-prime* integers  $k, h$ . This gives, for all  $n$ :

$$\begin{aligned} x &\equiv \alpha * n + \beta && (\text{mod } M) \\ &\equiv 1 * n + 0 && (\text{mod } M) \\ &\equiv n && (\text{mod } M) \end{aligned}$$

Making the integers  $k, h$  be co-prime *systematically* eliminates the unfortunate possibility that, for some  $d > 1$ , that for all values of  $n$  that  $x \equiv n \pmod{d * M}$ .

Why is this? Suppose that for some  $d > 1$ ,  $x \equiv n \pmod{d * M}$ . Then this says that:

$$\begin{aligned} x &\equiv \alpha * n + \beta && \pmod{d * M} \\ &= n + (g * d * M) \end{aligned}$$

for some  $g > 0$ . Then, expanding the definitions of  $\alpha$  and  $\beta$ :

$$\begin{aligned} \alpha * n + \beta &= (1 + k * M) * n + h * M \\ &= n + (g * d * M) \end{aligned}$$

Simplifying and collecting terms, this gives:

$$n + (k * n + h) * M = n + (g * d) * M$$

Simplifying by cancelling  $n$  from both sides and dividing out by  $M$ , this gives:

$$k * n + h = g * d$$

This equation supposedly holds for every value of  $n$ . Putting  $n = 0$ , this means that  $h = g * d$ . Putting  $n = 1$  then shows that  $k = 0$ , contradicting the assumption that  $k > 0$ .

Admittedly, any divisor,  $d$  of modulus  $M$  does necessarily have the property that  $x \equiv n \pmod{d}$  – but this is of less concern since, firstly,  $x \pmod{d}$  is clearly less than  $M$  and probably less than  $|n|$ , and secondly, none of  $M, d$  or  $n$  are commonly known, except by the local client. In any event, if there was any concern of that sort, the local client could always choose  $M$  to be some appropriately large enough prime<sup>6</sup>.

With  $\alpha$  and  $\beta$  chosen in this way then:

$$\begin{aligned} a * x^2 + b * x + c & \\ &\equiv a * (\alpha * n + \beta)^2 + b * (\alpha * n + \beta) + c && \pmod{M} \\ &\equiv a \alpha^2 * n^2 + (2a \alpha \beta + b \alpha) * n + (a \beta^2 + b \beta + c) && \pmod{M} \\ &\equiv a(1) * n^2 + (2a(1)(0) + b(1)) * n + (a(0) + b(0) + c) && \pmod{M} \\ &\equiv a * n^2 + b * n + c && \pmod{M} \end{aligned}$$

---

<sup>6</sup>It is perhaps worth noting that there is a further refinement of this idea exploiting the use of *two* co-prime moduli  $M_1$  and  $M_2$  – where  $M_1$  is used to encipher the value  $n$  (i.e. where  $M_1 > n$ ), and  $M_2$  enciphers the additive constant. Briefly, put  $\alpha = k * M_1 + 1$  and  $\beta = h * M_2$  and then encipher  $n$  as  $x = \alpha * n + \beta$ , as before. Encoded values would be subsequently deciphered by first reducing by modulus  $M_2$  to eliminate the additive constant, and then further reducing by modulus  $M_1$  to extract the value of  $n$  from the enciphering. For this to work satisfactorily, it is clearly necessary to have  $M_2 > \alpha * g$ , where  $g$  is the greatest natural value in range. All this ensures that  $((\alpha * n + \beta) \pmod{M_1}) \pmod{M_2} = n$  for any  $n$  in range.

Naturally, all of this generalises to arbitrary polynomials  $q$  over the integers, since modular arithmetic is mathematically a *ring congruence* and is therefore substitutive:

$$\begin{aligned} q(x) &\equiv q(\alpha * n + \beta) \pmod{M} \\ &\equiv q(n) \pmod{M} \end{aligned}$$

Furthermore, the remote service has no clue about the values of  $\alpha$  and  $\beta$  or indeed the value of modulus  $M$ , all of which are *only* known to the local client.

### 4.3 A worked example of homomorphic enciphering

Suppose that the remote service's polynomial  $p$  is:

$$p(n) \hat{=} 41 + 3 * n + 2 * n^2$$

with advisory input range  $0 \leq n \leq 100$ .

The only information given by the remote service to clients is that, over the advisory input range from 0 to 100, the operation is less than 30,000, but greater than 0.

Let's take the natural input to be  $n = 23$ .

Choosing modulus  $M = 56734$  (i.e. greater than 30000) and suppose we make  $\alpha = 4 * M + 1 = 226937$  and  $\beta = 3 * M = 170202$ . The enciphered value of natural input 23 is then given by  $x = \alpha * 23 + \beta = 5389753$ , and this value<sup>7</sup> is then sent to the remote service, which then calculates:

$$\begin{aligned} p(5389753) &= 2 * 5389753^2 + 3 * 5389753 + 41 \\ &= 58098874802018 + 16169259 + 41 \\ &= 58098890971318 \end{aligned}$$

The remote service sends this final value back to the local client, which then decipheres the final result using modulus  $M = 56734$ , by calculating  $58098890971318 \bmod 56734 = 1168$ .

Just to check, let's ordinarily evaluate the polynomial  $p$  using the value 23 as input:

$$\begin{aligned} p(23) &= 2 * 23^2 + 3 * 23 + 41 \\ &= 1058 + 69 + 41 \\ &= 1168 \end{aligned}$$

<sup>7</sup>The enciphered input value,  $x$ , is typically *much* different to its natural value,  $n$ , that  $x$  enciphers. This value of  $x$  will almost inevitably lie outside the advisory input range as advertised by the remote service.

This is entirely normal and is to be expected. The remote service appreciates that the values it is asked to compute with have been enciphered. The advisory input range needs to be given not to restrict the natural input values, but to provide guidance to assist clients in making judicious selections for their modulus values.

This is precisely as expected.

#### 4.4 Commonalities and Differences

It is worth noting some helpful commonalities and differences between the two techniques (*Linear Steganography*, and *Modular Enciphering*) examined here. It will no doubt be appreciated that each technique has its own distinctive use-case.

- Although both techniques use the *same* enciphering method e.g.  $\alpha * n + \beta$ , each uses a different *deciphering* method.

The reason these techniques are different is because they are performing different jobs. Accordingly, the key parameters ( $\alpha$ ,  $\beta$ ) involved in each technique are chosen differently according to different criteria.

- The Linear Steganography technique is for exchanging data using a *shared key*. The key parameters are necessarily shared and should not be repeated in a predictable fashion.
- The Modular Enciphering technique is for privately performing a particular kind of calculation, but using public infrastructure. The key parameters here are **not** shared beyond the local client, but typically need to be carefully chosen depending upon the range of possible natural inputs and other details about the function being calculated remotely.

### 5. Mitigating extensions

Previous sections have discussed the basic ideas behind linear steganography and the proposed method for homomorphic enciphering. The objective of this section is to describe some ways to mitigate certain limitations that were highlighted earlier. All of these limitations are due to the need to use modular arithmetic to decipher the output of homomorphic enciphering.

#### 5.1 Handling negative outputs by using offsets

Because our homomorphic enciphering technique makes use of modular arithmetic in its deciphering phase, this creates ambiguity if representing negative values e.g.  $-4 \equiv 7 \pmod{11}$ , occurring on the output.

The solution is for the remote service to *offset* the operation (i.e. add a constant) so that over its advisory input range, the polynomial doesn't go negative. This naturally requires the local client to perform a compensating correction (i.e. subtract that constant) *after* deciphering in order to compute the correct result<sup>8</sup>.

This means the local client needs to know the offset that the remote service applies to the polynomial, so that the local client can deduct it later.

---

<sup>8</sup>Technically, the modulus  $M$  has to be chosen large enough by the local client so that when the offset is subtracted off, the result is valid and lies within the specified bounds. This broadly means that the modulus  $M$  has to be chosen greater than the bound given by the remote service, plus the offset.

### 5.1.1 Worked example – handling offsets

Let's assume the polynomial is given by:

$$p(n) \hat{=} n^3 - n^2 - 12 * n$$

Let's suppose that the remote service selects the input range to be from -10 to 10, and over this range, determines that the minimum value of  $p$  is -980 and that the maximum is 780 (each attained at the endpoints).

To ensure a non-negative value for the output over this range, the remote service adds a suitable offset value - for example, 1010. This offset can be arbitrarily chosen, just so long as the calculation over the advisory input range would be non-negative.<sup>9</sup>

The remote service then advises local clients that for the advisory input range from -10 to 10, the polynomial value lies between -1000 and 1000. In addition, the offset applied is 1010.

Let's assume the local client now chooses a modulus  $M = 5121$  which is greater than the maximum bound + offset, and then put  $\alpha = 3 * M + 1 = 15364$  and  $\beta = 5 * M = 25605$ . Let's also assume that the natural input value is  $n = -6$ . The corresponding enciphered input is  $x = 15364 * (-6) + 25605 = -66579$ . Despite this enormous negative quantity, this will all work out as expected – have courage!

The remote service takes the enciphered input value,  $x = -66579$  and computes the value of  $p'(x) = p(x) + 1010$  i.e. the polynomial plus offset, as follows:

$$\begin{aligned} p'(-66579) &= (-66579)^3 - (-66579)^2 - 12 * (-66579) + 1010 \\ &= -295128943822539 - 4432763241 + 798948 + 1010 \\ &= -295133375785822 \end{aligned}$$

Therefore,  $p(-66579) + 1010 = -295133375785822$ , and this value is sent back to the local client, which decipheres it by reducing it using the modulus 5121:

$$-295133375785822 \bmod 5121 = 1320$$

Finally, the local client computes the end-result by subtracting off the offset 1010 that the remote server had earlier added on, to obtain the result, -180.

Just to check this result, lets compute  $p(-6)$  directly:

$$\begin{aligned} p(-6) &= (-6)^3 - (-6)^2 - 12 * (-6) \\ &= -216 - 36 + 72 \\ &= -180 \end{aligned}$$

<sup>9</sup>As promised earlier, here is another instance of the *functional conjugation* pattern: Map – Operate – Unmap.



This is therefore as expected.

Now, was adding the offset really necessary? Let's briefly look at what happens if no offset had been involved. Recalling that  $p(x) = p(-66579) = -295133375786832$ , this can be deciphered by taking the modulus:

$$-295133375785332 \text{ mod } 5121 = 4941$$

Although clearly  $5121 = 4941 + 180$ , the clear advantage of using an offset is that it is arithmetically correct – and gets the sign right!

Finally, would it have mattered if the offset had been different? Let's instead assume that the offset was, say, 1653. Then  $p(x) + 1653 = -295133375785179$ . The local client then decipheres this:  $-295133375785179 \text{ mod } 5121 = 1473$ , and then subtracting off the offset 1653 obtains -180, as before.

**5.1.2 Handling rationals by using *Decimal Scaling***

Integers are useful, but rationals are typically even more so in applications. Unfortunately, rational numbers and modular arithmetic don't really work together in a natural manner, making rational numbers hard to use in this form of homomorphic enciphering. Rationals can work perfectly well with Linear Steganography, meaning that enciphering and deciphering naturally extends to the rationals entirely as one expects. However, in the context of homomorphic enciphering, using rationals directly becomes confusing and makes little sense, mainly because of the need to use modular arithmetic for deciphering – and that requires integers, not rationals.

However, by going back to first principals with rationals, this yields an approach to using rational inputs and coefficients in a way that is compatible with homomorphic enciphering. In particular, this approach illustrates how to support decimal fractions of fixed-length significance, such as 3.14 (to two significant figures). As such it represents a pragmatic choice, while also simplifying some of the complications of the necessary scaling involved<sup>10</sup>.

Let's illustrate the approach through some examples. The rational number 3.14159 has five significant figures following the decimal point and is of course equivalent to:

$$3.14159 = \frac{314159}{100000} = \frac{314159}{10^5}$$

In terms of calculating with rationals, lets consider how a simple decimal calculation works, such as:

$$4.71 * 3.14159 - 8.617 = 14.7968889 - 8.617 = 6.1798889$$

---

<sup>10</sup>A more comprehensive approach involving general rational scaling is of course entirely feasible, but a bit more complex to describe. The present account represents a reasonable compromise that is straightforward and useful for practical calculation, and as such, demonstrates feasibility.

Treating this as a decimal scaled integer computation gives here:

$$\begin{aligned}
 4.71 * 3.14159 - 8.617 &= \frac{471}{10^2} * \frac{314159}{10^5} - \frac{8617}{10^3} \\
 &= \frac{471 * 314159}{10^7} - \frac{8617 * 10^4}{10^7} \\
 &= \frac{147968889 - 86170000}{10^7} \\
 &= \frac{61798889}{10^7} = 6.1798889
 \end{aligned}$$

In effect, this illustrates how all of the separate rational divisions can be bundled up into a *single* division. This is an important observation for combining rationals with this form of homomorphic enciphering. It demonstrates how calculations involving scaled decimals can become integer calculations involving a single division by a power-of-ten constant, illustrating the basic principles needed – and the next section gives an extended worked example illustrating how this might work in practice.

### 5.1.3 Worked example - Decimal scaling

This example illustrates both decimal scaling for handling decimals, together with the use of offsets for handling negatives. Let's assume that the polynomial  $p$  is:

$$p(n) \hat{=} 14.57 + 2.6 * n - 4.51 * n^2$$

where the natural input  $n$  can be a decimal to two significant figures. Let's assume that the remote service selects an advisory input range from -10.0 to 10.0. The maximum value of this quadratic over this range is attained at  $n = 2.6/9.02 = 0.29$  with value  $p(n) = 14.95$ . The minimum value over the range is attained at endpoint  $n = -10.0$  with value  $p(n) = -462.43$ .

As the minimum value is negative, this implies that an offset needs to be given, and, since decimal values are involved, a decimal scaling factor also needs to be worked out. This implies that the remote service must determine the scaled version of polynomial  $p$  it uses to perform calculations. The process of determining the offset, the scaling factor and the scaled version of the polynomial is briefly illustrated here.

Assume that natural input  $n = t/10^2$ , for some integer  $t$  - since  $n$  is assumed to be a decimal with (at most) two significant figures, lying in the range from -10 to 10, which means that integer  $t$  lies in the range from -1000 to 1000.

Next, the remote service chooses an offset value ensuring that polynomial output values are positive. Given that the minimum value is -462.43, lets assume the offset is chosen equal to 510. Furthermore, lets amend polynomial  $p$  by adding on the offset, naming this amended polynomial  $p'$ :

$$\begin{aligned}
 p'(n) &\hat{=} \text{offset} + p(n) = 510 + p(n) \\
 &= 510 + 14.57 + 2.6 * n - 4.51 * n^2 \\
 &= 524.57 + 2.6 * n - 4.51 * n^2
 \end{aligned}$$

Substituting now for  $n$  by putting  $n = t/10^2$ :

$$\begin{aligned}
 p'(t/10^2) &= 514.57 + 2.6 * (t/10^2) - 4.51 * (t/10^2)^2 \\
 &= \frac{52457}{10^2} + \frac{260}{10^2} * \frac{t}{10^2} - \frac{451}{10^2} * \frac{t^2}{10^4} \\
 &= \frac{52457 * 10^4}{10^6} + \frac{260 * t * 10^2}{10^6} - \frac{451 * t^2}{10^6} \\
 &= \frac{524570000 + 26000 * t - 451 * t^2}{10^6}
 \end{aligned}$$

Now, by defining scaled integer polynomial  $q(t)$  by:

$$q(t) \hat{=} 524570000 + 26000 * t - 451 * t^2$$

this gives the following:

$$p'(n) = p'(t/10^2) = q(t)/10^6$$

With all of this in place, the remote service can then say that the polynomial has an advisory input range from -10.0 to 10.0, supports decimals up to two significant figures, and has upper and lower bounds between 20 and -500 over this range. The offset has been set to 510 and the decimal scaling factor is  $10^6$ .

Given the above info, the local client then chooses modulus  $M$  greater than the maximum bound (20) plus the offset (510) together, but also necessarily scaled by  $10^6$ , the decimal scaling factor. This means the modulus  $M$  needs to be *greater than*  $530 * 10^6$ . The local client now (randomly) chooses modulus  $M = 530067217$ . With this value, now choose the enciphering parameters  $\alpha = 3 * M + 1 = 1590201652$ , and  $\beta = 2 * M = 1060134434$ .

Let's now assume that the natural input chosen by the local client is  $n = 2.67 = 267/10^2$ , meaning that  $t = 267$ .

Given these key parameters, the enciphering value  $x$  corresponding to  $t$  becomes:

$$\begin{aligned}
 x &\hat{=} \alpha * t + \beta \\
 &= 1590201652 * 267 + 1060134434 \\
 &= 425643975518
 \end{aligned}$$

The enciphering input value  $x$  is then sent to the remote service, which then computes the value of  $q(x)$ :

$$\begin{aligned} q(x) &= q(418955177435) \\ &= 524570000 + 26000 * 418955177435 - 451 * 418955177435^2 \\ &= -81708930035473526411076124 \end{aligned}$$

This computed value is sent back to the local client for deciphering.

The local client can then use their modulus, computing:

$$-81708930035473526411076124 \bmod 530067217 = 499360661$$

However, because the remote service introduced a scaling factor and offset when computing this value, this must also be compensated for to obtain the final result.

$$\begin{aligned} \left( \frac{499360661}{10^6} \right) - 500 &= 499.360661 - 500 \\ &= -0.639339 \approx -0.64 \quad (\text{to 2 decimal places}) \end{aligned}$$

As a check, let's compute the value of  $p(n)$ , for  $n = 2.67$ :

$$\begin{aligned} p(2.67) &= 14.57 + 2.6 * 2.67 - 4.51 * 2.67^2 \\ &= 14.57 + 6.942 - 32.151339 \\ &= -10.639339 \approx -10.64 \quad (\text{to 2 decimal places}) \end{aligned}$$

Again, this is as expected.

This example might seem overly laboured perhaps, but it demonstrates that the approach tangibly works, not just as an abstract construction, but also as a concrete calculation.

## 5.2 Handling large moduli – using the Chinese Remainder Theorem (CRT)

From the earlier worked examples, it seems that the moduli involved may become somewhat large and rather cumbersome, which then leads to larger numbers being used throughout. As seasoned number theorists will have surely noted and anticipated, the celebrated *Chinese Remainder Theorem*<sup>11</sup> could be deployed to help tackle this issue, by effectively replacing some very large moduli by a series of smaller, more manageable values.

The Chinese Remainder Theorem (CRT) says that, given a series of modular congruences of the form:

---

<sup>11</sup>This theorem is so-called because its earliest known statement was found in a Chinese mathematical treatise written during the Tang dynasty (3<sup>rd</sup> to 5<sup>th</sup> century CE).

$$\begin{aligned} X &\equiv a_1 \pmod{M_1} \\ X &\equiv a_2 \pmod{M_2} \\ &\dots \\ X &\equiv a_n \pmod{M_n} \end{aligned}$$

for unknown  $X$ , where the moduli  $M_i$  are pairwise co-prime, then there is a unique solution for  $X$  modulo  $(M_1 * M_2 * \dots * M_n)$  satisfying all of those congruences simultaneously[3, 4, 5]. Fortunately, the CRT is widely implemented and solutions can be readily computed.

**5.2.1 Two worked examples – revisited**

Let’s illustrate how the CRT could be used by revisiting a couple of the examples given earlier.

**Revisiting example: Handling offsets**

Looking back at the worked example from section 5.1.1, the polynomial  $p$  used there is given by:

$$p(n) = n^3 - n^2 - 12 * n$$

and where the natural input was  $n = -6$ . From the earlier example, the offset was taken to be 1010, making the polynomial  $p'$ :

$$\begin{aligned} p'(x) &\hat{=} p(x) + 1010 \\ &= x^3 - x^2 - 12 * x + 1010 \end{aligned}$$

Earlier the modulus chosen was  $M = 5121$  – this is greater than the maximum bound of  $p$  (i.e. 800) plus the offset (1010), each specified by the remote service. However, the effective modulus only needs to be greater than  $800 + 1010 = 1810$ .

Instead of doing that, the local client could instead have chosen smaller moduli and then used the CRT to combine the results. The chosen moduli must be pairwise co-prime to each other and where their product exceeds the 1810 lower-bound. Let’s choose the following smaller moduli to be:

$$93, 67$$

which are both co-prime and their product is 6231, well above the 1810 value.

The same calculation performed earlier is now repeated, but instead for each of these smaller moduli in turn. These calculations are tabulated below, where  $n = -6$ :

$M$	$\alpha = 3 * M + 1$	$\beta = 5 * M$	$x = \alpha * n + \beta$	$p'(x)$	$p'(x) \pmod{M}$
93	280	465	-1215	-1795074010	86
67	202	335	-877	-675283728	26

The local client then applies the CRT:

$$\begin{aligned} R &\equiv 86 \pmod{93} \\ R &\equiv 26 \pmod{67} \end{aligned}$$

The resulting solution value for  $R$  is 830, the same as the deciphered value found previously. Deducting the offset value of 1010, the final value of the polynomial  $p$  is -180, as expected.

**Revisiting example: Decimal scaling**

In the example from section 5.1.3, the local client needed to choose a huge modulus  $M = 521737456$ .

Again, lets apply the CRT and (randomly) choose a number of smaller, pairwise co-prime moduli such as:

$$1357, 3971, 4792, 7893$$

Their product is 203816174974632, which is certainly (much) greater than the modulus used earlier, 521737456.

As before, the usual calculation is then repeated, but instead for each of these smaller moduli in turn. These calculations are then tabulated below where the integer input  $t$  is 267 (rescaled from  $n = 2.67$ ) and the (scaled) polynomial  $q(x)$  is given by:

$$q(x) = 524570000 + 26000 * x - 451 * x^2$$

$M$	$\alpha = 3 * M + 1$	$\beta = 2 * M$	$x = \alpha * t + \beta$	$q(x)$	$q(x) \pmod{M}$
1357	4072	2714	1089938	-535743291615644	945
3971	11914	7942	3188980	-4586403213570400	3440
4792	14377	9584	3848243	-6678746789471099	717
7893	23680	15786	6338346	-18118592825521916	2123

The local client then applies the CRT with this information:

$$\begin{aligned}
 R &\equiv 945 \pmod{1357} \\
 R &\equiv 3440 \pmod{3971} \\
 R &\equiv 717 \pmod{4792} \\
 R &\equiv 2123 \pmod{7893}
 \end{aligned}$$

Solving for  $R$ , this gives 499360661, which is the same as the deciphered value computed earlier. Since it is exactly as before, the final computed value is also identical.

It is worth noting that using fewer moduli would also have sufficed – the same result could have been reached by instead using any three of the four congruences to solve for  $R$ . Additionally, there is nothing special about the choice of these moduli – other values of the moduli could have been used.

These two examples illustrate how some of the large number processing can be replaced by a series of similar calculations, but using smaller moduli instead.

## 6. Application: Multi-Party Computation

One of the goals of privacy-preserving computation is showing how to perform calculations that involve several independent sources while retaining confidentiality. This area is known as (secure) Multi-Party Computation (MPC) (see [2]). Perhaps unsurprisingly, the technique very briefly described here for homomorphic encipherment can also be adapted for this application too.

The set up is that there is a collection of *sources* numbered 1,  $\dots$ ,  $N$  that provides input, and a *receiver* acting as coordinator and receives output from the remote server. None of the sources should be able to decipher the inputs given by the other sources. Naturally, the receiver can decipher all inputs from the sources - but not necessarily be able to distinguish which input was given by which source. This is illustrated in Figure 3. To support multi-party computation,

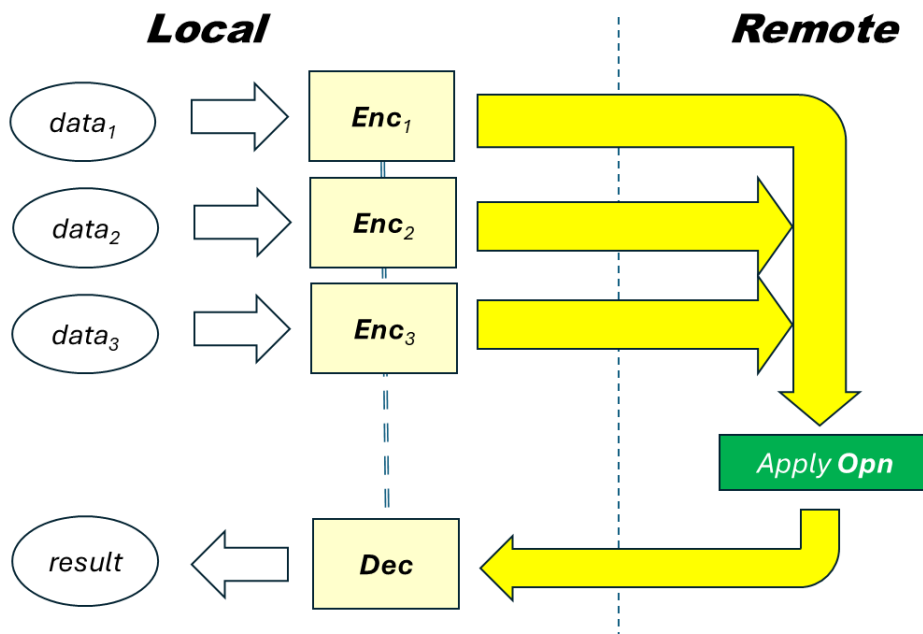


Figure 3: Multi-party computation

the receiver acts as coordinator and privately chooses a *base modulus*  $M$ , known only to the receiver, who then chooses a set of pairwise *co-prime* factors  $\{k_1, k_2, \dots, k_N\}$ . The  $i^{\text{th}}$  source is then given their own private modulus  $M_i = k_i * M$  by the receiver, obviously without knowledge of the base modulus  $M$ .

Each source  $i$  then independently enciphers their inputs using its own specific modulus,  $M_i$  and passes that data to the remote server. The remote server performs a calculation (typically summation) and passes the enciphered result back to the receiver, who uses the base modulus  $M$  to decipher the overall result.

Because each source uses a modulus  $M_i$  divisible by base modulus  $M$  to encipher their input, the receiver can use  $M$  to correctly decipher the result in the same way as for homomorphic

encipherment.

Other than the receiver, none of the sources can correctly decipher the final result. This is because none of the sources know the base modulus  $M$  used by the receiver and, moreover, there is no incentive to determine what it is (e.g.  $M = g.c.d(M_i, M_j)$  for  $i \neq j$ ). Doing so would most likely globally reveal what everyone's input was, negating independent confidentiality for the participants.

Moreover, assuming that each of the  $k_i$ 's is greater than one and co-prime to each other, the separate moduli  $M_i$  ensure that everyone's actual inputs remain plausibly confidential. Of course, there is nothing to prevent the receiver disclosing the base modulus and the set of actual (anonymised) inputs at some later date. Such a disclosure would have the benefit of allowing each participant to check that their own input had been received and to independently test the outcome.

## 7. Conclusion

This article has proposed the use of some elementary arithmetical techniques for the purpose of providing a form of homomorphic enciphering in a practical manner that is clearly scalable. This scalability derives from that fact that no particular special purpose hardware or software is required to implement these techniques. This is largely because arbitrary-precision integer arithmetic has become very widely supported these days by languages such as Python. A reference implementation in Python of the various ideas and concepts described can be downloaded from here:

**Download:** [LS-RH-HE-code+Feb2025.zip](#)

**SHA 256:** [62a26d1c1780123d97d9ef629ff9ae4b3bf6a50b474d75d16695af85d3c4b441](#)

Despite the techniques themselves being commonplace and mathematically elementary, their application to homomorphic enciphering don't appear to be particularly well-known or commonly available within the public domain. The purpose of this paper is simply to make the ideas better known. Because of their overall simplicity and broad applicability, these techniques are quite likely to be well-known in various circles. As such, I'm probably not the first to have worked them out for myself!

It is quite delightful to have seen how these humble and well-known arithmetic techniques can so surprisingly and comprehensively contribute to the field of homomorphic enciphering. Not only that, but those very same ideas also contribute elsewhere within privacy-preserving computation, as can be seen by their application to Multi-Party Computation in particular.



## References

- [1] Wikipedia, *Homomorphic Encryption*,  
[https://en.wikipedia.org/wiki/Homomorphic\\_encryption](https://en.wikipedia.org/wiki/Homomorphic_encryption)
- [2] Wikipedia, *Secure Multi-Party Computation*,  
[https://en.wikipedia.org/wiki/Secure\\_multi-party\\_computation](https://en.wikipedia.org/wiki/Secure_multi-party_computation)
- [3] Wikipedia, *Chinese Remainder Theorem*,  
[https://en.wikipedia.org/wiki/Chinese\\_remainder\\_theorem](https://en.wikipedia.org/wiki/Chinese_remainder_theorem)
- [4] S. MacLane and G. Birkhoff *Algebra*, 3<sup>rd</sup> Edition, AMS Chelsea, 1999
- [5] Ivan Niven, Herbert S. Zuckerman, Hugh L. Montgomery *An Introduction to The Theory of Numbers*, 5<sup>th</sup> Edition, Wiley, 1991

## BCS-FACS Peter Landin Semantics Seminar 2024

# Inferentialism, Proof-theoretic Semantics, and Computation

David Pym  
*University College, University of London*

December, 2024

**Reported by:** Brian Monahan and Bill Stoddart

### Brian's thoughts:

The BCS FACS Peter Landin Semantics Seminar is our leading event of the year. It is generally given by someone who has, like Peter Landin himself, broadly contributed to our understanding of programming languages – perhaps through mathematical semantics, or how languages are processed, or indeed, how languages are thought about, applied and used. We have so far been fortunate to continue the tradition of hosting speakers who knew Peter Landin personally and were influenced by his work – and this year is again no exception!



*(Photo courtesy of Jonathan Bowen)*

We were fortunate to have David Pym, Professor of Information, Logic, and Security at UCL speak to us. Following several personal anecdotes about Peter Landin and interactions with him at Queen Mary College, London, David Pym moved swiftly to his core topic – discussing the relationship between inferentialism, semantics, and computation.

The main philosophical ideas behind Inferentialism are not only reasonably straightforward but also quite foundational and liberating. Its primary thesis is that whatever is meant linguistically is governed and determined by how statements are used, particularly with respect to the reasoning supporting

that use. In short, meaning is governed by inference and how one reasons using it – and not the other way around. This further hints at how programming languages might acquire meaning through how they are used and reasoned about. As it was later implied, that was something that Peter Landin would most likely have strongly approved of! Inferentialism itself has been emerging elsewhere within standard philosophy and linguistics in, for example, the work of the philosopher Robert Brandom (see [2]).

For me personally, some early form of inferentialism, as David Pym describes it, has been with me, in one form or another, for quite some time. I started out as a postgraduate student at Edinburgh in the late 70s/early 80s, working with Edinburgh LCF (Logic of Computable Functions) [1] for my doctorate. The LCF system is an early precursor of modern-day theorem-proving environments such as HOL, Isabelle, Coq and Lean. Around that time in '81, several “inferentialist” ideas were undoubtedly “in the air” at Edinburgh. For example, Gordon Plotkin formulated his Structural Operational Semantics, [3, 4] in which he applied a strongly inferentialist approach to programming language semantics, initially formulated as a set of influential lecture notes. In addition to Milner's LCF and Plotkin's SOS, one might also add the whole transition systems approach to

*process algebras*, as pioneered by Robin Milner, firstly with CCS and then later with SCCS and pi-calculus. Inferentialism has deep roots, indeed!

The talk's clear focus on *intuitionistic propositional logic* might have initially seemed a bit unusual - but in fact, David Pym's focus here cuts straight to the core of what inferentialism is about. Firstly, a focus on a form of propositional logic rather than predicate calculus meant that logical structure in terms of sequents and connectives would be the primary concern, without the distraction of variables, equality, and quantifiers cluttering the discussion (Note that in any case, these can be added later without further controversy.)

By taking intuitionistic logic as the main example, this emphasised the need for a primarily *proof-theoretical* account of the logical system. Intuitionistic logic has its roots in the mathematical philosophy of the eminent constructivist L.E.J. Brouwer (1881-1966). In Brouwer's view, mathematical proof should be as direct as possible, requiring explicit construction wherever necessary. Therefore, certain classical inferences such as *excluded middle* ( $\vdash p \vee \neg p$ ) and *double negation* ( $\vdash p \Leftrightarrow \neg(\neg p)$ ) would be deemed strictly superfluous.

However, eliding these classical inferences changes the logic's underlying denotational or model-theoretic semantics. For instance, classical propositional logic is denotationally representable in terms of Boolean algebras, but intuitionistic logic is instead representable in terms of something more general, such as *Heyting algebras*, which characterise *distributive lattices* more generally. As one might expect, every Boolean algebra is a Heyting algebra, but not vice-versa. In practice, this means that all theorems in intuitionistic logic are also classical, but not the other way around.

However, in the talk, David Pym neatly avoided any such model-theoretic discussions by staying strictly within a proof-theoretic account that takes inference as primary and where "meaning is given in terms of inference". For me (Brian), the most interesting part of the talk was the introduction of *Proof-theoretic Validity* and *Base-extension Semantics* as an alternative to the Tarskian approach of using *Model-theoretic semantics* to provide semantic validation of logical systems. As remarked in the talk, a significant disadvantage of model-theoretical analysis is its propensity to produce something of even greater mathematical complexity than the system one started out from.

As briefly explained, the notions of Proof-theoretic Validity and Base-extension Semantics consider different ways to validate putative proofs by examining internal meta-logically inspired conditions and constraints over the provability relation itself. From the way that David Pym sets this up, Proof-theoretic Validity is primarily about validity of *proofs* and, in contrast, Base-extension Semantics is primarily about validity of *formulae*. A brief example of this was given by considering *Sandqvist's semantics for IPL*, as presented in Figure 4 (see [6, 8, 9, 10, 11]).

The remainder of David Pym's talk included a brief but again intriguing discussion of *reductive logic*, an area that David Pym has pioneered with Eike Ritter[5]. In this approach, one works *backwards* from a goal statement towards known results. If this search process can find known results that match each of the outstanding subgoals, the proof-search finishes, and the proof is constructed by going forward, proving the goal. This form of backwards proof-search was pioneered and implemented by Robin Milner in the context of his LCF system and is now commonly known as *tactical reasoning*. This approach to proof construction is very commonly employed today in all modern theorem-proving systems (see [7, 12]).

### Sandqvist's Semantics for IPL

Base rules  $\mathcal{R}$ , application of base rules, and satisfaction of formulae in a (possibly finite) countable base  $\mathcal{B}$  of rules  $\mathcal{R}$  are defined as follows:

$\frac{[P_1] \quad \dots \quad [P_n]}{q_1 \quad \dots \quad q_n} \mathcal{R}$	<p>(Ref) <math>P, p \vdash_{\mathcal{B}} p</math></p> <p>(App<math>\mathcal{R}</math>) if <math>((P_1 \Rightarrow q_1), \dots, (P_n \Rightarrow q_n) \Rightarrow r)</math> and, for all <math>i \in [1, n]</math>, <math>P, P_i \vdash_{\mathcal{B}} q_i</math>, then <math>P \vdash_{\mathcal{B}} r</math></p>
<p>(At) for atomic <math>p</math>, <math>\Vdash_{\mathcal{B}} p</math> iff <math>\vdash_{\mathcal{B}} p</math></p> <p>(<math>\supset</math>) <math>\Vdash_{\mathcal{B}} \phi \supset \psi</math> iff <math>\phi \Vdash_{\mathcal{B}} \psi</math></p> <p>(<math>\wedge</math>) <math>\Vdash_{\mathcal{B}} \phi \wedge \psi</math> iff <math>\phi \Vdash_{\mathcal{B}} \psi</math> and <math>\psi \Vdash_{\mathcal{B}} \phi</math></p>	<p>(<math>\vee</math>) <math>\Vdash_{\mathcal{B}} \phi \vee \psi</math> iff, for every atomic <math>p</math> and every <math>\mathcal{C} \supseteq \mathcal{B}</math>, if <math>\phi \Vdash_{\mathcal{C}} p</math> and <math>\psi \Vdash_{\mathcal{C}} p</math>, then <math>\Vdash_{\mathcal{C}} p</math></p> <p>(<math>\perp</math>) <math>\Vdash_{\mathcal{B}} \perp</math> iff, for all atomic <math>p</math>, <math>\Vdash_{\mathcal{B}} p</math></p> <p>(Inf) for <math>\Theta \neq \emptyset</math>, <math>\Theta \Vdash_{\mathcal{B}} \phi</math> iff, for every <math>\mathcal{C} \supseteq \mathcal{B}</math>, if <math>\Vdash_{\mathcal{C}} \theta</math> for every <math>\theta \in \Theta</math>, then <math>\Vdash_{\mathcal{C}} \phi</math></p>

There is a substitution (cut) operation on bases that maps derivations  $P \vdash_{\mathcal{B}} p$  and  $p, Q \vdash_{\mathcal{B}} q$  to a derivation  $P, Q \vdash_{\mathcal{B}} q$ .




Figure 4: Sandqvist's semantics for IPL (Slide 11)

Following this discussion of proof-search, David Pym elegantly and cleanly applied modern proof theory to understanding the process of executing a logic program. Briefly, a logic program provides the set of basic facts (in an extended clausal form known as the *hereditary Harrop formulae*) over which the proof-search then operates reductively by matching goals and subgoals as indicated above, until no sub-goals remain, establishing the original goal.

Overall, David Pym gave an excellent introduction to his current research in modern proof theory. By exploring and highlighting inferentialist and proof-theoretic approaches, this programme of work may help to reduce the need for utilising complex model-theoretic validations, which will surely simplify and ease the broader application of formal logic.

### Bill's thoughts: A subjective review of the talk

When I offered to write this review, it certainly wasn't from a standpoint based on any knowledge of the subject, quite the opposite, in fact. Nevertheless, the abstract intrigued me, and the talk itself had a profound effect on my mathematical imagination. Here, we had a talk from a philosopher of mathematics who worked in the parallel tradition of constructive mathematics and intuitionistic logic. Those of us working with classical mathematics generally pick up the bare minimum - constructivists aren't satisfied to know something must exist; they want to know what it is. This is reflected in intuitionistic logic, which rejects the law of the excluded middle and consequently 'proof by contradiction' in its classic form (but I learnt that constructivists have a more restricted form of such proof in their toolkit). Proof-theoretic semantics offers an alternative view of mathematical truth. In the classical view, we are constrained to think of proof in some new formalism as an abstract game with symbols, which only connects to truth once we have constructed a model and linked our new formalism to a classical formalism (set theory) that has stood the test of time. After creating a model, our theory is said to be sound, and the results we

prove in it can be considered “true”. I’ve gone along with that as a framework but always found the explanation a little dry because the subjective feeling of truth comes from the mathematical intuitions obtained from *doing* mathematics, from carrying out proofs.

In the talk, proof-theoretic semantics was discussed at two levels. First at the level of rules of inference and second at the level of proof construction. My ignorance of the field meant the first part was somewhat baffling, though sprinkled with intriguing insights, but the second part completely gripped my imagination. A proof is often thought of formally as a deductive progression in which rules are applied to argue from some premises to a conclusion. In the talk, David contrasted this with an approach in which the proof process begins by taking some putative goal (the result to be proved) and choosing an inference rule to apply backwards to obtain one or more sufficient sub-goals. This process then continues, generating a proof tree, with proof being complete when we arrive at leaves that can be discharged directly without generating further sub-goals. As this process proceeds, we typically argue under certain assumptions. For example, if our goal is to prove  $P \Rightarrow Q$ , we will have a rule that adds  $P$  to the hypotheses under which we are arguing and generates  $Q$  as the new sub-goal. The backward reasoning process is intertwined with forward reasoning, where we look at the set of hypotheses under which we carry out the proof and use them to generate further hypotheses. For example, if one such hypothesis is  $A \wedge B$ , we can add  $A$  and  $B$  as separate hypotheses, or if we have both  $P$  and  $P \Rightarrow Q$ , we can add  $Q$ . The way the leaves of the proof tree are discharged is when the goal is prove’s one of our assumptions, in which case it can simply be ticked off. This description of proof in terms of sequent calculus is being built into the formulation of proof-theoretic semantics.

The talk then moved on to resolution theorem provers. Here, the takeaway for me was that this technique is still viable with intuitionistic logic.

The end of the talk left me needing a glass of wine or two, and it was only on the way to the bus stop that I started thinking of questions I might have asked.

For example, how can we reject the law of the excluded middle when we can clearly demonstrate it in operation by wiring up a NOT gate and an OR gate? In fact this little example was quite illuminating for me. In intuitionistic logic we can prove both  $A \Rightarrow A \vee \neg A$  and  $\neg A \Rightarrow A \vee \neg A$ . We can thus prove our little circuit will have a *true* on its output line whatever signal is placed on its input line, but we must assume some signal is placed there in order to reason constructively.

However, on the bigger question of mathematical truth, does proof-theoretic semantics have a view of truth that pays more heed to its subjective but shared apprehension by practitioners?

The after-effect of the talk was to leave me with an increased curiosity for the alternative tradition. For example, I was intrigued to see that there is a quantum physicist from this tradition, Nicolas Gisin, a professor at Geneva University, who says that any deterministic view of the universe is undermined by the unreality of real numbers used in defining initial conditions.

### Link to slides and video:

<https://www.bcs.org/events-calendar/2024/december/hybrid-event-facs-agm-and-peter-landin-semantics-seminar/>

## References

- [1] M. Gordon, R. Milner, and C.P. Wadsworth, *Edinburgh LCF: A Mechanized Logic of Computation*, LNCS 78, Springer-Verlag, 1979
- [2] Robert Brandom, *Making It Explicit: Reasoning, Representing & Discursive Commitment*, Harvard University Press, 1994
- [3] Gordon Plotkin, *A Structural Approach to Operational Semantics*, Technical report FN-19, Computer Science Dept., Aarhus University, 1981 Reprinted in *Journal of Logic and Algebraic Programming*, vol 60-61, pp 17-139, 2004  
[https://homepages.inf.ed.ac.uk/gdp/publications/sos\\_jlap.pdf](https://homepages.inf.ed.ac.uk/gdp/publications/sos_jlap.pdf)
- [4] Hans Hüttel. *Transitions and Trees - An introduction to Structural Operational Semantics*, Cambridge University Press, 2010
- [5] David Pym, Eike Ritter, *Reductive Logic and Proof-Search: Proof Theory, Semantics, and Control*, Oxford Logic Guides 45, Oxford University Press, 2004
- [6] A. Gheorghiu, D. Pym, *Definite formulae, Negation-as-Failure, and the Base-extension Semantics of Intuitionistic Propositional Logic*, Bulletin of the Section of Logic, Czech Academy of Sciences, 2023  
<https://repozytorium.uni.lodz.pl/xmlui/handle/11089/48072?show=full&locale-attribute=en>
- [7] A. Gheorghiu, S. Docherty, and D. Pym, *Reductive Logic, Proof-search, and Coalgebra: A Perspective from Resource Semantics*, Revised version in *Samson Abramsky on Logic and Structure in Computer Science and Beyond*, Alessandra Palmigiano and Mehrnoosh Sadrzadeh (editors), Outstanding Contributions to Logic, Springer, 2023. Manuscript: <http://www.cs.ucl.ac.uk/staff/D.Pym/rlp-sc.pdf>
- [8] D. Pym, E. Ritter, and E. Robinson, *Categorical Proof-theoretic Semantics*, Studia Logica, 2024  
<https://link.springer.com/article/10.1007/s11225-024-10101-9>
- [9] A. Gheorghiu, T. Gu, and D. Pym, *A note on the practice of logical inferentialism: the state-effect interpretation, definitional reflection, and completeness*, 2nd Logic and Philosophy: Historical and Contemporary Issues Conference, Vilnius, Lithuania, May 2024 Manuscript: <https://arxiv.org/pdf/2403.10546>
- [10] A. Gheorghiu, T. Gu, and D. Pym, *Inferentialist Resource Semantics*, To appear, Proc. Mathematical Foundations of Programming Semantics (MFPS), Oxford, June 2024. Electronic Notes in Theoretical Informatics and Computer Science (ENTICS)  
<https://arxiv.org/abs/2402.09217>
- [11] A. Gheorghiu, T. Gu, and D. Pym, *Proof-theoretic Semantics for Intuitionistic Multiplicative Linear Logic*, Studia Logica, Dec. 2024,  
<https://link.springer.com/article/10.1007/s11225-024-10158-6>
- [12] A. Gheorghiu, D. Pym, *Semantic Foundations of Reduction Reasoning*, Manuscript: [http://www.cs.ucl.ac.uk/staff/D.Pym/Semantic\\_Foundations\\_of\\_Reduction\\_Reasoning.pdf](http://www.cs.ucl.ac.uk/staff/D.Pym/Semantic_Foundations_of_Reduction_Reasoning.pdf)

## FME/BCS-FACS Joint Webinar

# The Self-Aware Digital Twin

Einar Broch Johnsen  
*University of Oslo*

15/10/2024

**Reported by:** Keith Lines

### Abstract

Digital twin applications use digital artifacts to twin physical systems. The purpose is to continuously mirror the structure and behaviour of the physical system, such that stakeholders can analyse the physical system by means of the digital twin for, e.g., decision support, scenario exploration, model-based control, systematic reconfiguration, etc. In this talk, we discuss the basic concepts of a digital twin, and how digital twins differ from models and control systems. We show how digital twins can be realized in a framework that integrates models at runtime, semantic technology and simulation models, in order to leverage domain knowledge in model-based analysis driven by live data.

### Biography

Einar Broch Johnsen is a professor at the Department of Informatics, University of Oslo. He is active in formal methods for distributed and concurrent systems, including object-oriented and actor languages, cloud computing, robotics and digital twins. He is one of the main developers of the SMOL programming language [1] for digital twins.

Currently, Einar leads a project on a digital twin of the Oslo fjord. He is also involved in the MSCA network REMARO [2] on reliable AI for marine robotics and the HEU project NebulOuS, in which his group develops a digital twin for a meta-operating system.

## 8. Introduction

The 2024 Formal Methods Europe (FME) / BCS FACS joint seminar presented a very interesting overview of work developing digital twins that contain self-awareness, i.e., the ability to automatically reconfigure themselves according to changes in the corresponding physical system. As noted during the Q+A that completed the seminar, an important aim is to make twins of systems that are not easily understood, such as the Oslo fjord.

### 8.1 Background: What are digital twins?

The seminar began with definitions of the term digital twin presented over the years, along with impressive outcomes that are claimed to have resulted from their use (increased energy production, more efficient use of medical facilities, boosted earnings, etc.).

The origins of digital twins lie with NASA’s space programs [3]. They can mean many different things to many different organisations. The US National Academies of Sciences, Engineering, and Medicine (NASEM) defines a digital twin as follows (highlights were noted in bold by the speaker) [4]:

A digital twin is a set of **virtual information constructs** that **mimics the structure, context, and behaviour** of a natural, engineered, or social system (or system-of-systems), is dynamically updated with data from its physical twin, has a **predictive capability**, and informs decisions that realize value.

Broch Johnsen presented his own explanation, see Figure 1. He is of the opinion that models that underpin digital twins should be as abstract as possible, and concern aspects of the system that are of direct interest. This formal methods-like approach contrasts with the approach that the models should be as realistic as possible (and therefore more complicated than they might otherwise need to be).

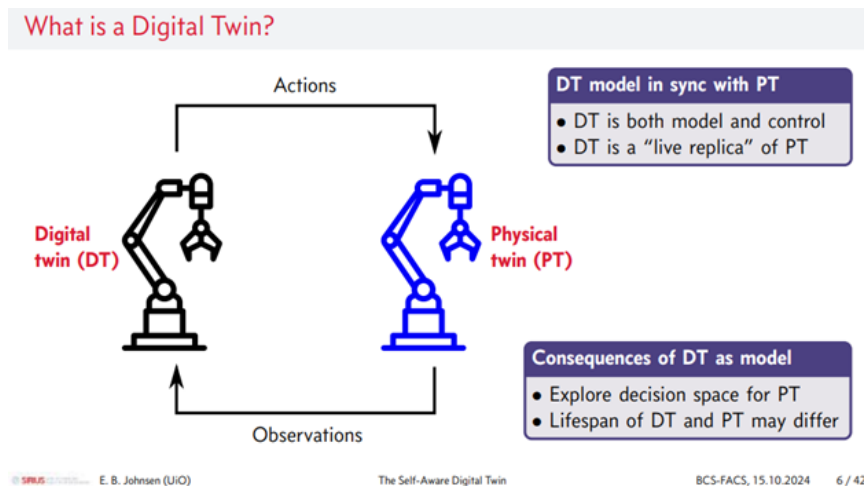


Figure 1: Explanation of digital twins

### 8.2 Background: Reflective programming, SMOL and conceptual layers

Reflective programming, also known as reflection, is one approach being explored to implement self-aware digital twins. The Wikipedia definition [5] of this concept was used, i.e., "... the ability of a process to examine, introspect, and modify its own structure and behaviour". As



Wikipedia notes, reflection goes back to assembly language programming but fell out of use with the introduction of higher-level languages. More modern object-oriented languages, such as Java, reintroduced the concept.

The **Semantic Micro Object Language (SMOL)** [1] is an object-oriented language customised for programming digital twins. It supports reflection and has a formal semantics that helps with reasoning about programs, such as proving correctness.

Organising digital twins into **conceptual layers** is a key concept. The **data layer** concerns input from the physical twin, such as sensors. The **structural twin** organises this data into structured information and holds domain knowledge, structured using the web ontology language (OWL) [6]. The **behavioural twin** concerns simulations and analysing the behaviour of the system. The interaction between the top two layers is the main area of interest of this work.

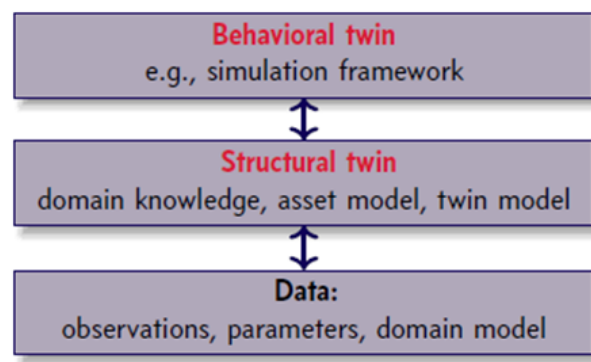


Figure 2: Conceptual layers

## 9. Case studies

Case studies of digital twins illustrated the concepts. The first concerned temperature control within a building:

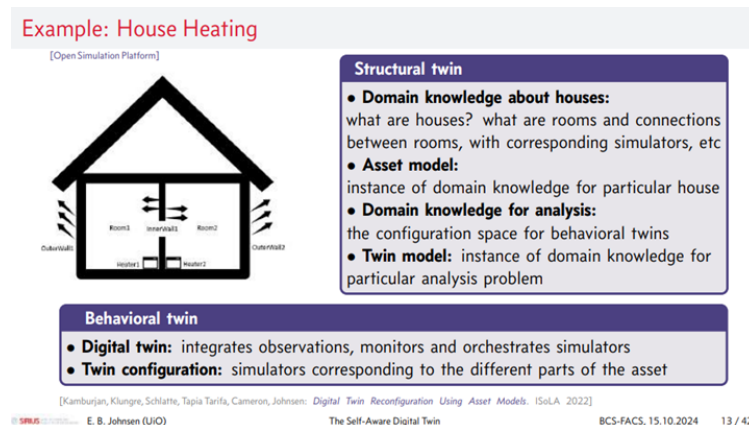
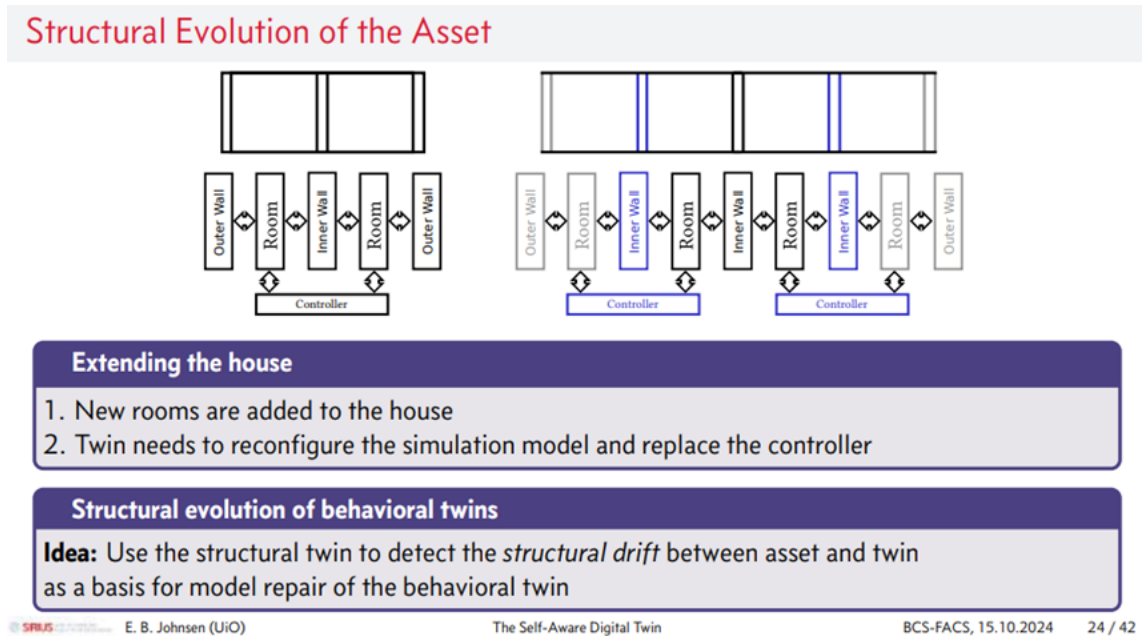


Figure 3: Digital twin case study: Building

The structure of the building can be modified, such as by adding new rooms. The asset model

should be kept up to date by the structural engineers. The behavioural twin detects differences, or “drift”, between the asset model and twin model (see Figure 4) and modifies the twin model (such as by adding another control) and its behaviour accordingly.



**Figure 4:** Digital twin case study: Modification of building

A further case study concerned a biological system [7], a greenhouse for growing basil. Humidity levels and watering are adjusted according to the lifecycle stage and health of the plants.

## 10. Question and answers

The presentation ended with a Q+A session. Topics covered included:

- SMOL is not a language for programming models. It uses the Functional Mock-up Interface (FMI) standard [8] to call functional mock-up units (FMUs). FMUs contain models that are defined using software tools that can export FMI-compatible code. Similarly, SMOL programs can upload and use time series data.
- What does correctness mean for programs that are self-adapting?
- Could an inappropriate choice of model have a negative effect on the physical system?
- The final question was whether uncertainties, from sources such as sensor data, are being taken into account. Uncertainties can influence decisions made using digital twins. The answer was not yet, but the importance of uncertainties is appreciated. Approaches are being considered, such as integrating them within the underlying knowledge graphs. There are interesting discussions to be held on this matter.

## Slides and Video:

<https://ebjohnsen.org/talk/the-self-aware-digital-twin/2024-ebjohnsen-bcs-facs.pdf>

<https://www.youtube.com/watch?v=QBaFoQENSQo>

## References

- [1] Semantic Micro Object Language (SMOL): <https://smolang.org>
- [2] Reliable AI for Marine Robotics (REMARO): <https://remaro.eu/>
- [3] Digital Twins and Living Models at NASA:  
<https://ntrs.nasa.gov/citations/20210023699>
- [4] National Academies of Sciences, Engineering, and Medicine (NASEM) definition of digital twin:  
<https://nap.nationalacademies.org/resource/26894/RH-digital-twins.pdf>
- [5] Wikipedia entry on reflective programming:  
[https://en.wikipedia.org/wiki/Reflective\\_programming](https://en.wikipedia.org/wiki/Reflective_programming)
- [6] OWL, web ontology language: <https://www.w3.org/OWL/>
- [7] Digital Twin Lab: <https://smolang.org/lab.html>
- [8] Functional Mock-up Interface (FMI) standard:  
<https://fmi-standard.org/>

# Festschrift reviews: Cliff Jones and Jim Woodcock

January 2025

Reviewed by: Jonathan P. Bowen



Covers of the three Festschrift volumes under review

The field of formal methods is now sufficiently well-established that many key researchers are reaching the end of their “formal” career or beyond. Celebrations are typically held when leaders in the field reach their retirement or a significant birthday, normally by decade. For example, in 2023, Jifeng He, formerly a collaborator with Tony Hoare at the Programming Research Group in Oxford, reached his 80th birthday with a celebratory event in Shanghai, China, where he is now based [1, 2] and an associated Festschrift volume was published in the Springer *Lecture Notes in Computer Science* (LNCS) series [3]. At the start of 2024, Tony Hoare reached his 90th birthday and we celebrated this in an issue of *FACS FACTS* with contributions of reminiscences by a number of his colleagues [6]. In this current article, three recent Festschrift volumes in the LNCS series celebrating formal methods researchers are reviewed.

The first pair of Festschrift volumes (LNCS 14780 [4] and LNCS 14781 [5]) both celebrate the 80th birthday of Cliff Jones, known for his contributions of VDM early in his career. He gained his doctorate under Tony Hoare at Oxford. This is his first ever Festschrift collection of papers, hence the double volume, with contributions by 90 scientists spread across 20 countries in 30 papers. These are ordered alphabetically by the family name of the first author, so there is no favouritism in the volume where papers are placed. The academic papers are preceded by several shorter and more personal testimonials. All the contributions provide evidence of Cliff Jones’s wide influence on computer science, especially its more formal aspects. Most of the papers start with a tribute to Cliff Jones, relating the content of the paper to his foundational work. I confess that I am a co-author of one of the papers, by chance of co-authorship towards the end of the second volume.

This is the first time to my knowledge where a Festschrift celebration has taken two volumes, but I believe it to be justified in this case. As the preface notes, there has not been a previous Festschrift for Cliff Jones, perhaps due to his modest nature, and this set of two volumes makes up for that omission. The fact that the papers are ordered by author name means that the subject matter varies considerably from paper to paper if tackled sequentially. That said, the variation shows the wide influence of Cliff Jones worldwide. For navigation by readers, it could have been helpful to have some guidance on the varied subject matter in the preface or perhaps a brief index at the end with important keywords. I expect that nowadays most readers will find papers by keyword search online, so this could be seen increasingly as less of an issue where “papers” are often accessed and read in electronic form online rather than on paper. In any case, the two volumes provide a wonderful additional resource for formal method researchers. The fact that Cliff Jones is mentioned and cited so often across the papers demonstrates his research influence as well as the level of affection in which he is held.

One of the papers in the LNCS 14781 volume is by Jim Woodcock. He recounts how, early in his career, on completion of his PhD at the University of Liverpool in 1980, his doctoral supervisor gave him a copy of the then new book *Software Development: A Rigorous Approach* (on VDM) authored by Cliff Jones. Soon after, Jim Woodcock attended Cliff Jones’s BCS-FACS Christmas lectures at Imperial College in London. FACS itself had only been established shortly before that in 1978. The lectures influenced Woodcock to use a VDM “rely and guarantee” approach, as advocated by Cliff Jones. Fast forward in time to the present and Jim Woodcock has himself now reached retirement age, which leads us to the final Festschrift volume under review.

The third volume under consideration, LNCS 14900 [7], celebrates the “formal” retirement from the University of York of Jim Woodcock, also a former colleague of Tony Hoare in Oxford, from his professorial position, with an associated Festschrift event with presentations of the papers at the University in York. This includes 14 papers on the application of formal methods by authors from eight countries. Jim Woodcock is known for his strong connections of theory and practice. As one example, he collaborated with IBM on the mathematical modelling of their transaction processing system CICS. This work led to a Queen’s Award for Technological Achievement in 1992 and CICS is still in use today. More recently, he has followed the UTP approach advocated in the 1998 book *Unifying Theories of Programming* by Tony Hoare and Jifeng He [8], relating various formal semantic styles in a common framework. Using this approach, he developed the *Circus* language with a UTP semantics, combining the Z notation, Dijkstra’s guarded command language, and a CSP-style algebra for concurrent systems.

The papers in the LNCS 14900 Festschrift volume reflect Jim Woodcock’s research interests and engineering approach in applying formal methods. The contributions include two papers by Cliff Jones, one as a sole author and one as a co-author, demonstrating the fact that most research-active academics never truly retire. They continue with their research interests even when they are not paid for it! I am sure this will be the case for Jim Woodcock too as he enters his “retirement”.

It is perhaps worth noting that Cliff Jones was the founding Editor-in-Chief of the *Formal Aspects of Computing* journal, associated with the BCS-FACS Specialist Group, and until recently, Jim Woodcock has been the journal’s Editor-in-Chief as well. FACS is grateful to both Cliff Jones and Jim Woodcock for their important roles in ensuring the success of the journal.

All three of these Festschrift volumes under review are likely to include papers of interest to anyone working in formal methods. While I suspect not many people will read these volumes cover to cover, apart perhaps from those to whom they are dedicated, they make a wonderful additional resource of papers for those in the field of formal methods. If you have access to them, no doubt most likely online nowadays, I recommend dipping into their rich resources by many leading formal methods researchers from around the world.

## References

- [1] Bowen, J. P. (2023). Jifeng He's 80th birthday Festschrift Symposium. *FACS FACTS*, **2023**(2):26–30, July. BCS. <https://www.bcs.org/media/10894/facs-jul23.pdf>
- [2] Bowen, J. P. (2024). Trip report: Jifeng He's 80th birthday Festschrift Symposium. *FACS FACTS*, **2024**(1):14–23, January. BCS. <https://www.bcs.org/media/wjqdvedd/facs-jan24.pdf>
- [3] Bowen, J. P., Li, Q., and Xu, Q. (eds.) (2024). *Theories of Programming and Formal Methods: Essays Dedicated to Jifeng He on the Occasion of His 80th Birthday*. Springer, LNCS **14080**, Festschrift. doi:10.1007/978-3-031-40436-8
- [4] Cavalcanti, A. and Baxter, J. (eds.) (2024). *The Practice of Formal Methods: Essays in Honour of Cliff Jones, Part I*. Springer, LNCS **14780**, Festschrift. doi:10.1007/978-3-031-66676-6
- [5] Cavalcanti, A. and Baxter, J. (eds.) (2024). *The Practice of Formal Methods: Essays in Honour of Cliff Jones, Part II*. Springer, LNCS **14781**, Festschrift. doi:10.1007/978-3-031-66673-5
- [6] Denvir, T., He, J., Jones, C.B., Roscoe, A.W., Stoy, J., Sufrin, B., and Bowen, J. P. (2024). Tony Hoare @ 90. *FACS FACTS*, **2024**(2):5–42, July. BCS. <https://www.bcs.org/media/1wrosrpv/facs-jul24.pdf>
- [7] Foster, S. and Sampaio, A. (eds.) (2024). *The Application of Formal Methods: Essays Dedicated to Jim Woodcock on the Occasion of His Retirement*. Springer, LNCS 14900, Festschrift. doi:10.1007/978-3-031-67114-2
- [8] Hoare, C. A. R. and He, J. (1998). *Unifying Theories of Programming*. Prentice Hall Series in Computer Science. <http://www.unifyingtheories.org>
- [9] Jones, C. (1980). *Software Development: A Rigorous Approach*. Prentice Hall International Series in Computer Science.

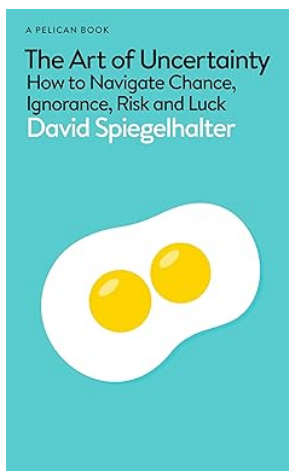
# Book Review: The Art of Uncertainty

David Spiegelhalter

December 2024

**Reviewed by:** Tim Denvir

This is a combined review of David Spiegelhalter's book, *The Art of Uncertainty*, and of his talk to the EMS on the same theme.



On the 25th September I heard a talk by David Spiegelhalter at the Edinburgh Mathematical Society on "Chance, Luck and Ignorance (How to put our uncertainty into numbers)". David Spiegelhalter has written two books, *Art of Statistics* and *Art of Uncertainty*. The talk was strongly based on his second book.

Spiegelhalter has impressive credentials: DSc., FRS, OBE, knighted for services to statistics, and the Royal Society's Michael Faraday Medal.

Spiegelhalter's main thesis in the talk was that probability doesn't exist! That is to say, it is not a property of things in the external world. It is personal, an attribute of one's own knowledge and lack of it. He defined three kinds of luck:

- **Constitutive luck:** what you are born with,
- **Circumstantial luck:** being in the right/wrong place or time,
- **Resultant luck:** how it worked out.

There were some 300 people at the EMS event, I was seated in about the third row, and it was acceptable and feasible to ask questions and challenge during the talk, which I did. The acoustics were such that one could easily be heard by the audience without a microphone. The book consists of an Introduction followed by 16 chapters. Both the book and the lecture contained a quiz, which helps engage the reader/listener.

Spiegelhalter's second thesis is that wherever possible, uncertainty should be put into numbers, not just verbal descriptions, which can be interpreted very differently by different individuals. Indeed, the titles of his first two chapters reflect this: *Uncertainty is Personal* and *Putting Uncertainty into Numbers*. Every chapter, except the final one, named I suspect with a bit of tongue in cheek, *The Future of Uncertainty*, has at its end a summary of its salient points, which I found useful after reading the body of a chapter; I found it helped to gather the ideas together. I've mentioned the first two chapters; the subsequent few were quite thought-provoking: *Taming Chance with Probability, Surprises and Coincidences* and *Luck*. The titles of those chapters are reasonably self-explanatory. Some of the intermediate chapters are less exhilarating, but throughout

the book David Spiegelhalter cites real-life examples and situations. For example, Chapter 10, *What, or Who is to Blame? Causality, Climate and Crime*, Chapter 12, *Risk, Failure and Disaster*: these chapters and others cite and explore real-life events. I did find that in some of these intermediate chapters, a few of Spiegelhalter's elaborations obscured rather than clarified his earlier definitions.

The penultimate two chapters, 14 and 15, *Communicating Uncertainty and Risk* and *Making Decisions and Managing Risks*, are very much concerned with the public and human reactions to information about uncertainty and risk. They were excellent chapters, in my view. Chapter 15 concentrates on "acceptable risk". In the final chapter, 16 *The Future of Uncertainty*, he refers to the anthropic principle in discussing questions like "what's the probability of you being born, or of the human race existing?". He dismisses these questions! He discusses "What is a reasonable probability of the human race existing in the near future?" and in a subsection on AI he writes about Large Language Models without explaining or defining the term. A final subsection, "A Manifesto for Uncertainty", is really in the nature of a summary, or conclusion, about the whole book: a kind of soliloquy on the author's part.

Despite my various criticisms, I'm glad I bought and read this book. It was discounted at the EMS event: £25 down to £18. David Spiegelhalter demonstrates considerable academic expertise. One should not be put off by his, dare I say, showmanship.



## Forthcoming Events

If you have suggestions for future FACS seminar speakers or other events, especially if you are willing to help with co-organisation or even give a talk, please contact Alvaro on [Alvaro.Miyazawa@york.ac.uk](mailto:Alvaro.Miyazawa@york.ac.uk).

### Events Venue (unless otherwise specified):

BCS, The Chartered Institute for IT  
Ground Floor, 25 Cophall Avenue, London, EC2R 7BP

The nearest tube station is Moorgate, but Bank and Liverpool Street are within walking distance as well. The new Elizabeth Line is now very convenient for the BCS London office, by alighting at the Liverpool Street stop and leaving via the Moorgate exit.

Date and Time	Title
Tuesday 11 February 5:30pm - 8:30pm	<b>Hybrid event: Functional Programming and Dependent Types for Metrology</b>

Details of all forthcoming events can be found online here:

[https://www.bcs.org/membership/member-communities/  
facs-formal-aspects-of-computing-science-group/](https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/)

Please revisit this site for updates as and when further events are confirmed.

# FACS Committee



Formal Aspects of Computing  
Science Specialist Group



**Jonathan Bowen**  
*FACS Chair and BCS Liaison*



**John Cooke**  
*Treasurer and Publications*



**Roger Carsley**  
*Secretary and Vice-Treasurer*



**Alvaro Miyazawa**  
*Seminar Organiser and Vice-Secretary*



**Margaret West**  
*Inclusion Officer*



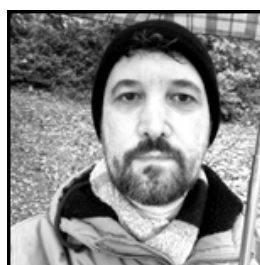
**Keith Lines**  
*Vice-Chair*



**Tim Denvir**  
*Newsletter Co-editor*



**Brian Monahan**  
*Newsletter Co-editor*



**Andrei Popescu**  
*LMS Liaison*



**Ana Cavalcanti**  
*FME Liaison*

FACS is always interested to hear from its members and keen to recruit additional helpers. Presently we have vacancies for officers to help with fund raising, to liaise with other specialist groups such as the Requirements Engineering group and the European Association for Theoretical Computer Science (EATCS), and to maintain the FACS website. If you are able to help, please contact the FACS Chair, Professor Jonathan Bowen at the contact points below:

BCS-FACS  
c/o Professor Jonathan Bowen (Chair)  
London South Bank University  
Email: [jonathan.bowen@lsbu.ac.uk](mailto:jonathan.bowen@lsbu.ac.uk)  
Web: <http://www.jpbowen.com>

You can also contact the other Committee members via this email address.

## Mailing Lists

As well as the official BCS-FACS Specialist Group mailing list run by the BCS for FACS members, there are also two wider mailing lists on the Formal Aspects of Computer Science run by JISCmail.

The main list: [facs@jiscmail.ac.uk](mailto:facs@jiscmail.ac.uk) can be used for relevant messages by any subscriber. An archive of messages is accessible under:

<http://www.jiscmail.ac.uk/lists/facs.html>

including facilities for subscribing and unsubscribing.

The additional list: [facs-event@jiscmail.ac.uk](mailto:facs-event@jiscmail.ac.uk) is specifically for announcement of relevant events.

Similarly, an archive of announcements is accessible under:

<http://www.jiscmail.ac.uk/lists/facs-events.html>

including facilities for subscribing and unsubscribing.

BCS-FACS announcements are normally sent to these lists as appropriate, as well as the official BCS-FACS mailing list, to which BCS members can subscribe by officially joining FACS after logging onto the BCS website.