An Informatics*Lexicon†

Dines Bjørner

DTU Compute, Technical Universty of Denmark, DK-2800 Kgs. Lyngby, Denmark
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

December 9, 2025

Contents

*[ι 43,π 10]
†With due respect to [19]

Preface

- Over the years, and in my more than 120 publications I have tried to be careful with consistent use of terms of my scientific field and its related fields: mathematics, philosophy and discrete, manifest and human-made and -assisted fields.
- Hence this lexicon.
- The lexicon entries range from characterizations to definitions.
- Besides terms directly of the field of computing, this lexicon also lists terms from mathematics, philosophy and *domains* – where You are kindly asked to look up entries related to entry 24 on page 7.
- In composing this lexicon I often thought of
  - *Denis Diderot* [1713–1784] *Encyclopédie (1751–1772)* and
  - *Dr. Samuel Johnson* [1709–1784] *A Dictionary of the English Language (1755).*
- But, as noted on the cover of this lexicon, my real "predecessor" and master, is *Michael A. Jackson [19, 20].*
- In conceiving of the Domain Science and Engineering [3, 4, 5, 6] – between the years 2010–2025 – I had to use terms most of which were not [yet] in the conventional 'informatics' vocabulary; terms such as

– domain	[ι 24,π 7][1]	– part set	[ι 73,π 13]
– entity	[ι 32,π 8]	– internal quality	[ι 48,π 10]
– endurant	[ι 30,π 7]	– unique identifier	[ι 113,π 18]
– external quality	[ι 36,π 9]	– mereology	[ι 65,π 12]
– solid (entity)	[ι 99,π 16]	– attributes	[ι 6,π 5]
– fluid (entity)	[ι 37,π 9]	– perdurant	[ι 74,π 13]
– part	[ι 72,π 13]	– channel	[ι 14,π 6]
– living species	[ι 55,π 11]	– action	[ι 2,π 3]
– atomic part	[ι 5,π 4]	– event	[ι 34,π 8]
– compound part	[ι 15,π 6]	– behaviour	[ι 8,π 5]
– composite (Cartesian part)	[ι 13,π 6]		

- Writing this document started on October 4, 2025.
  I will be updated and extended occassionally !
  Currently many entries have no text other than the term [to be defined].
  And currently many terms are yet to be entered.

[1] [ι *i* π *p*] designates ιtem *i* πage *p*

1 𝔸

1. **Abstraction:**

```
Conception, my boy, fundamental brain-work,
    is what makes the difference in all art
```

D.G. Rossetti[2]*: letter to H. Caine*[3]

> *Abstraction is a tool, used by the human mind, and to be applied in the process of describing (understanding) complex phenomena.*
>
> *Abstraction is the most powerful such tool available to the human intellect.*
>
> *Science proceeds by simplifying reality. The first step in simplification is abstraction. Abstraction (in the context of science) means leaving out of account all those empirical data which do not fit the particular, conceptual framework within which science at the moment happens to be working.*
>
> *Abstraction (in the process of specification) arises from a conscious decision to advocate certain desired objects, situations and processes as being fundamental; by exposing, in a first, or higher, level of description, their similarities and — at that level — ignoring possible differences.*

[14, C.A.R. Hoare *Notes on Data Structuring*][4]

Example 1: **Road Net:** A road net of street segments and street intersections can be abstracted as a graph: segments as edges and intersections as vertices .

2. **Action:** An action is 'something' that potentially changes the state [of a domain] [ι 102,π 17].

3. **Axiom:** An axiom, postulate, or assumption is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.

Example 3: **Stacks:**

Narrative:

The meaning of designations:

1. Stacks, s:S, have elements, e:E;
2. the empty_S operation takes no arguments and yields a result stack;
3. the is_empty_S operation takes an argument stack and yields a Boolean value result.
4. the stack operation takes two arguments: an element and a stack and yields a result stack.

[2]Dante Gabrielli Rosetti, 1828–1882, English poet, illustrator, painter and translator
[3]T. Hall Caine, 1853–1931, British novelist, dramatist, short story writer, poet and critic.
[4]We shall bring another quote of Tony Hoare as the last proper text of this paper, see Sect. **??** on page ??.

5. the unstack operation takes an non-empty argument stack and yields a stack result.
6. the top operation takes an non-empty argument stack and yields an element result.

The consistency relations:

7. an empty_S stack is_empty, and a stack with at least one element is not;
8. unstacking an argument stack, stack(e,s), results in the stack s; and
9. inquiring the top of a non-empty argument stack, stack(e,s), yields e.

Formalization:

The meaning of designations:

type
1. E, S

value
2. empty_S: **Unit** → S
3. is_empty_S: S → **Bool**
4. stack: E × S → S
5. unstack: S $\xrightarrow{\sim}$ S
6. top: S $\xrightarrow{\sim}$ E

The consistency relations:

axiom
7. is_empty(empty_S()) = **true**
7. is_empty(stack(e,s)) = **false**
8. unstack(stack(e,s)) = s
9. top(stack(e,s)) = e .

4. **Algebra:** By an *algebra* we shall understand (i) a set, $A = \{a_1, a_2, ..., a_n, ...\}$, of values, the *carrier* of the algebra, and (ii) a set, $\Omega = \{\omega_1, \omega_2, ..., \omega_m\}$, usually a finite set, of operations:
$$\omega_i(a_{j_1}, a_{j_2}, ..., a_{j_n}) = a_k$$
where $\omega_i : \Omega$ and $\{a_{j_1}, a_{j_2}, ..., a_{j_n}\} \subseteq A$.

Example 4: **Algebras:** Example 3 is an example of an algebra. So is Example 10, a Boolean Algebra. Others are:

A Queue Algebra: The *queue algebra* has, as *carrier*, the union of the set of all queue element values with the set of all queue values and, for example, create empty queue, enqueue, dequeue, first ("oldest"), last ("youngest"), and is_empty queue as *operations*.

A Directory Algebra: The *directory algebra* has, as *carrier*, the union of the set of all directory entry values (i.e., of value triples of entry name, date and information values) with the set of all directory values and, for example, create empty directory, insert entry in directory, directory look-up, edit directory entry and remove directory entry as *operations*.

A Graph Algebra: A *graph algebra* has, as *carrier*, the union of the set of all nodes, and the set of all edges, where nodes and edges are all distinctly labelled, and, for example, create empty graph, insert_node in graph, insert_edge in graph [between one or two nodes], depth_first_ search in graph and breadth_first_ search in graph, as [some of its] *operations*.

5. **Atomic Part:** By an *atomic part* we shall understand a part which the domain analyser considers to be indivisible in the sense of not meaningfully divisible, for the purposes of the domain under consideration, that is, to not meaningfully consist of sub-parts.

6. **Attribute:** [5] We can roughly distinguish between two kinds of attributes: those which can be motivated by **physical** (including chemical) **concerns**, and those, which, although they embody some form of 'physics measures', appear to reflect on **event histories**: *"if 'something', ϕ, has 'happened' to an endurant, e_a, then some 'commensurate thing', ψ, has 'happened' to another (one or more) endurants, e_b."* where the *'something'* and *'commensurate thing'* usually involve some 'interaction' between the two (or more) endurants. It can take some reflection and analysis to properly identify endurants e_a and e_b and commensurate events ϕ and ψ.

   Example 5: **Attributes:** Example attributes of (i) *road sections (links)* are (i.1) `length`, (i.2) `open-in-one-or-the-other-or-both directions`, etc., (ii) *banks* are (ii.1) `account number`, (ii.2) `balance`, etc.

2 𝔹

7. **B:** A formal program development and specification language [1]

8. **Behaviour:** By a behaviour we shall understand a set [ι 85,π 14] of sequences [ι 94,π 16] of actions [ι 2,π 3], events [ι 34,π 8] and behaviours.

9. **Boole, George:** [2 November 1815 – 8 December 1864] Born english. Professor at Queen's College, Cork, Ireland. Known for Boolean Algebra [ι 10,π 5]. `https://en.wikipedia.org/wiki/George_Boole`.

10. **Boolean Algebra:** We exemplify a Boolean Algebra with three values: **Bool:** {**true, false, chaos**} and the following opeations:

$$\begin{array}{ll} \sim : \mathbf{Bool} \to \mathbf{Bool}, & = : \mathbf{Bool}\times\mathbf{Bool}\to\mathbf{Bool} \\ \wedge : \mathbf{Bool}\times\mathbf{Bool}\to\mathbf{Bool} & \neq : \mathbf{Bool}\times\mathbf{Bool}\to\mathbf{Bool} \\ \vee : \mathbf{Bool}\times\mathbf{Bool}\to\mathbf{Bool} & \Rightarrow : \mathbf{Bool}\times\mathbf{Bool}\to\mathbf{Bool} \end{array}$$

∨, ∧, and ⇒ Syntactic Truth Tables

∨	**true**	**false**	**chaos**
true	**true**	**true**	**true**
false	**true**	**false**	**chaos**
chaos	**chaos**	**chaos**	**chaos**

∧	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**false**	**false**	**false**
chaos	**chaos**	**chaos**	**chaos**

⇒	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**true**	**true**	**true**
chaos	**chaos**	**chaos**	**chaos**

[5] Parts [ι 72,π 13] and fluids [ι 37,π 9]are typically recognised because of their spatial form and are otherwise characterised by their intangible, but measurable attributes. That is, whereas endurants, whether solid (as are parts) or fluids, are physical, tangible, in the sense of being spatial [or being abstractions, i.e., concepts, of spatial endurants], attributes are intangible: cannot normally be touched[6], or seen[7], but can be objectively measured[8]. Thus, in our quest for describing domains where humans play an active rôle, we rule out subjective "attributes": feelings, sentiments, moods. Thus we shall abstain, in our domain science also from matters of aesthetics.

3 $\mathbb{C}$

11. **Calculation:** A calculation is a deliberate process that transforms one or more inputs into one or more results. The term is used in a variety of senses, from the very definite arithmetical calculation of using an algorithm, to the vague heuristics of calculating a strategy in a competition, or calculating the chance of a successful relationship between two people [Wikipedia].

12. **Cartesian Enumeration:** A Cartesian [ι 13,π 6] enumeration of n elements, expressed by expressions $e_1, e_2, ..., e_n$, for $n > 1$[9], is:

    - $(e_1, e_2, ..., e_n)$

    where, of course, the elipses, ..., must be filled in with proper expressions. Cf. [ι 54,π 11], [ι 61,π 11] and [ι 87,π 15].

13. **Cartesian Part:** A Cartesian part is a compound part [ι 15,π 6] which consists of an "indefinite number" of two or more parts of distinctly named sorts .

14. **Channel:** A medium for communicating messages between behaviours, as expressed in CSP [16].

15. **Compound Part:** Compound parts are those which either are Cartesian- [ι 13,π 6] or are set-oriented parts [ι 89,π 15] .

16. **Computation:** A computation is any type of calculation that includes both arithmetical and non-arithmetical steps and which follows a well-defined model (e.g. an algorithm).

Mechanical or electronic devices (or, historically, people) that perform computations are known as computers [ι 17,π 6]. An especially well-known discipline of the study of computation is computer science [ι 18,π 6] [Wikipedia].

17. **Computer:** A computer is a collection of *hardware* and *software*, that is, is a machine that can be instructed to carry out sequences of arithmetic and logical operations automatically via computer programming [Wikipedia].

18. **Computer Science:** is the study and knowledge of the abstract phenomena that "occur" within computers [DB].

As such computer science includes *theory of computation, automata theory, formal language theory, algorithmic complexity theory, probabilistic computation, quantum computation, cryptography, machine learning and computational biology.*.

19. **Computing Science:** is the study and knowledge of how to construct "those things" that "occur" within computers [DB].

As such computing science embodies *algorithm and data structure design*, *functional-, logic-, imperative-* and *parallel programming*; *code testing, model checking* and *specification proofs*. Much of this can be pursued using *formal methods* [ι 38,π 9].

[9] – it makes no sense to express (), for $n = 0$, or (a) for $n = 1$

20. **Conservative Extension:** An extension of a logical theory is conservative, i.e., conserves, if every theorem expressible in the original theory is also derivable within the original theory [en.wiktionary.org/wiki/conservative_extension].

21. **CSP:** A conceptual program specification language [16].

4 𝔻

22. **Discrete Endurant:** By a *discrete* [or *solid*] endurant [ι 30,π 7] we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing: have 'body' [or magnitude] of three-dimensions: length, breadth and depth [21, Vol. II, pg. 2046].

Same as *solid endurant* [ι 101,π 17].

23. **Divide and Conquer:** In computer science, divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly `[Wikipedia]`.

24. **Domain:** By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* fragment of a *human assisted* reality, i.e., of the world. It includes its *endurants*, i.e., *solid and fluid entities* of *parts* and *living species*, and *perdurants* .

25. **Domain Analysis:** Inquiry into a domain as to its endurants [ι 30,π 7] and perdurants [ι 74,π 13]. .

26. **Domain Analysis & Description Ontology:** An ontology that "guides" the *domain analyzer cum describer* in systematically analysing and describing domains, such as we charaterize domains, cf. [ι 24,π 7], [ι 25,π 7] and [ι 27,π 7]. See Fig. 1 on the following page.

27. **Domain Description:** The description of a domain – as to its endurants [ι 30,π 7] and perdurants [ι 74,π 13].

28. **Domain Engineering:** is the engineering of *domain descriptions* based on the engineering of *domain analyses* [DB].

29. **Domain Modeling:** Same as domain analysis & description – [ι 25,π 7] & [ι 27,π 7].

5 𝔼

30. **Endurant:** Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persist, endure .

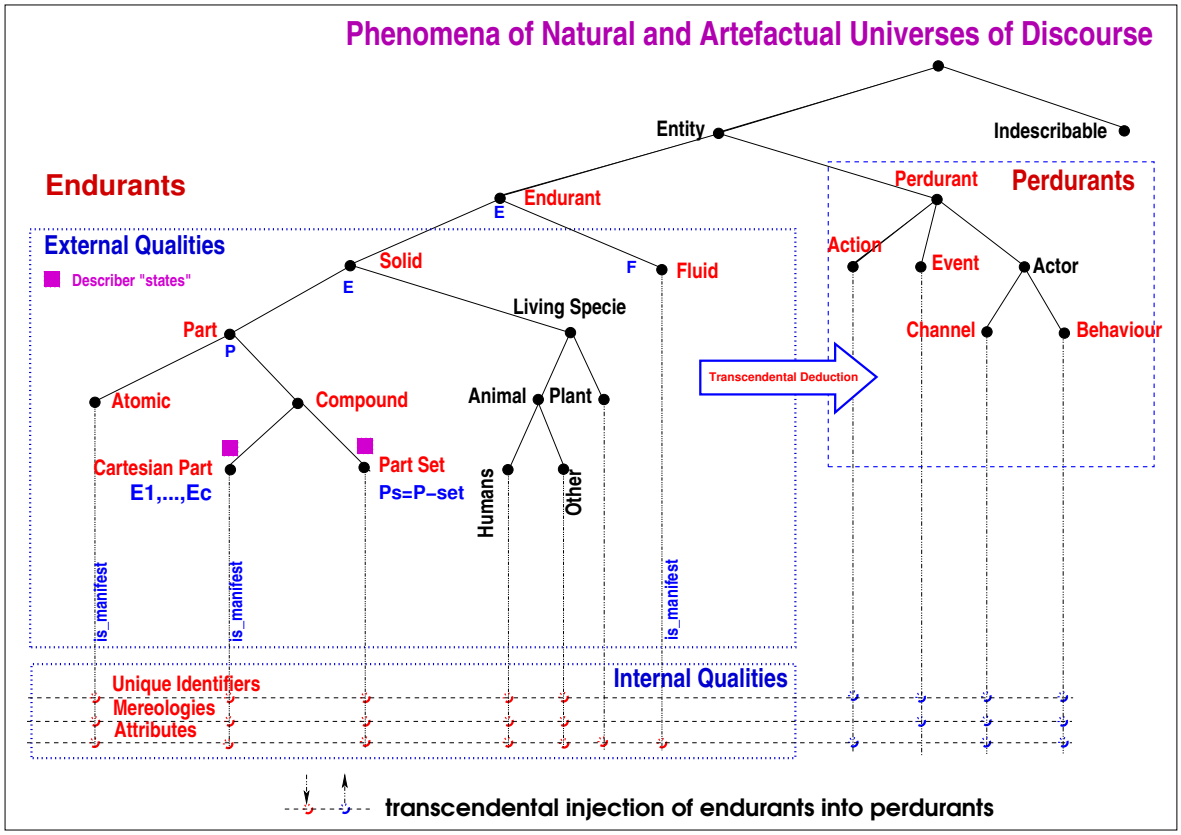

Figure 1: A Domain Analysis & Description Ontology

31. **Engineering:** is the use of scientific principles to design and build machines, structures, and other items, including bridges, tunnels, roads, vehicles, and buildings [Wikipedia].

The engineer *walks the bridge between science and technology:* analysing man-made devices for their possible scientific properties and constructing technology based on scientific insight.

32. **Entity:** By an entity we shall understand a phenomenon, i.e., something that can be observed, i.e., be seen or touched by humans, *or* that can be conceived as an abstraction of an entity; alternatively, a phenomenon is an entity, *if it exists, it is "being", it is that which makes a "thing" what it is: essence, essential nature*. If a phenomenon cannot be so **observed and described** then it is not en entity .

33. **Epistemology:** is the branch of philosophy concerned with the theory of knowledge – and is the study of the nature of knowledge, justification, and the rationality of belief [Wikipedia].

34. **Event:**

35. **Event B:** A formal program specification language [2], See also [ι 7,π 5].

36. **External Qualities:** External qualities of endurants [ι 30,π 7] of a manifest [ι 58,π 11] domain [ι 24,π 7] are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form.

6 𝔽

37. **Fluid:**

38. **Formal Method:** By a *formal method* [ι 66,π 12] we shall here understand a *method* whose techniques and tools can be understood mathematically [ι 63,π 12].

    For *formal domain, requirements* or *software engineering* methods *formality* means the following:

    - There is a set, one or more, specification languages – say for domain descriptions, requirements prescriptions, software specifications, and software coding, i.e., programming languages.[10]
    - These are all to be formal, that is, to have a formal syntax, a formal semantics, and a formal, typically *Mathematical Logic* proof system.
    - Some of the *techniques* and *tools* must be supported by a mathematical understanding.

39. **FORTRAN** A programming language [23].

40. **Fluid Endurant** By a *fluid endurant* we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [21, Vol. I, pg. 774] .

41. **Function** A function is a mathematical object which, when *'applied'* to a [mathematical] value, called its **argument**, potentially yields a [mathematical] value, called its **result**. If it fails to yield a result, the function is said to be undefined for that argument, and is then called a **partial function**, otherwise, if it yields results for any argumenent it is said to be a **total function**.

We refer to the Lambda Calculus [ι 50,π 10].

7 𝔾

8 ℍ

42. **Hardware:** The physical components of a computer: electronics, mechanics, optics, etc. `[Wikipedia]`.

[10] Most formal specification languages are textual, but graphical languages like `Petri nets [26], Message Sequence Charts[18], Statecharts [12], Live Sequence Charts [13]`, etc., are also formal.

9 𝕀

43. **Informatics:** By *informatics* we shall understand ***
44. **Instruction:** By an *instruction* we shall understand ***
45. **Intentional Pull:** The concept of intentional pull is a wider notion than that of invariant. Here we are not concerned with pre-/post-conditions on operations. Intentional pull is exerted between two or more phenomena of a domain when their relation can be asserted to **always** hold.
46. **Invariants:** The concept of invariants in the context of computing science is most clearly illustrated in connection with the well-formedness of data structures. Invariants then express properties that must hold, i.e., as a **pre-condition**, before any application of an operation to those data structures and shall hold, i.e. as a **post-condition** after any application of an operation to those data structures.
47. **Interface Requirements:** are those requirements which can be expressed in a combination of both domain and machine concepts. They are so because certain entities, whither endurants or perdurants, are ***shared** between the domain and the machine.* Cf. [ι 81,π 14].
48. **Internal Quality:**

10 𝕁

11 𝕂

12 𝕃

49. **Language:** By *language* we shall, with `[Wikipedia]`, mean a *structured system* of communication. Language, in a broader sense, is the method of communication that involves the use of – particularly human – languages. The 'structured system' that we refer to has come to be known as *Syntax, Semantics* and *Pragmatics*.
50. **Lambda Calculus:** In mathematical logic, the lambda calculus (also written as λ-calculus) is a formal system for expressing computation based on function abstraction and application using variable binding and substitution.
51. **Linguistics:** By *linguistics* we shall mean the scientific study of language.
52. **List:** By *list* we shall mean a finite, or possibly infinite sequence, i.e., an ordered set of elements[11].

[11] In mathematics, a sequence is an enumerated collection of objects in which repetitions are allowed and order matters. Like a set, it contains members (also called elements, or terms). The number of elements (possibly infinite) is called the length of the sequence. Unlike a set, the same elements can appear multiple times at different positions in a sequence, and unlike a set, the order does matter. Formally, a sequence can be defined as a function from natural numbers (the positions of elements in the sequence) to the elements at each position. The notion of a sequence can be generalized to an indexed family, defined as a function from an arbitrary index set. `[Wikipedia]`

53. **List Comprehension:** Let `TE` be a type name or type expression, `i, li, ui` be natural numbers and $\mathcal{B}$`(e(i))` be a predicate expression then

    $\langle$ e(i) | e:TE,i:**Nat** • li≤i≤ui ∧ $\mathcal{B}$(e(i)) $\rangle$

    is a list comprehension: the natural numer index ordered sequence of all `e(i)` in `TE` that satisfies predicate $\mathcal{B}$`(e(i))`.

54. **List Enumeration:** A List [ι 52,π 10] enumeration of n elements, expressed by expressions $e_1, e_2, ..., e_n$, for $n > 0$, is:

    $\langle e_1, e_2, ..., e_n \rangle$

    where, of course, the elipses, ..., must be filled in with proper expressions.Cf. [ι 12,π 6], [ι 61,π 11] and [ι 87,π 15].

55. **Living Species:**

13 𝕄

56. **Machine:** By a *machine* we shall understand a combination of software and hardware.

57. **Machine Requirements:** are those requirements which can be expressed sôlely in terms of machine concepts.

58. **Manifest Part:** By a manifest part we shall understand a part which 'manifests' itself either in a physical, visible manner, "occupying" an 𝔸ℝ𝔼𝔸 or a 𝕍𝕆𝕃𝕌𝕄𝔼 and a ℙ𝕆𝕊𝕀𝕋𝕀𝕆ℕ in 𝕊ℙ𝔸ℂ𝔼, or in a conceptual manner forms an organization in Your mind !.

59. **Map:** By a map we shall understand a effectively enumerable function:

    $[a_1 \mapsto b_1, a_2 \mapsto b_2, ..., a_n \mapsto b_n, ...]$,

    where a_i, b_i are map domain[12], **dom**, definition set, respectvely map range, **rng**, set value (expression)s.

60. **Map Comprehension:** Let `d(i), r(i)` be expresions of **type** `D`, respectiely **type** `R`, and $\mathcal{B}$`(d(i),r(i))`, then:

    [d(i)↦r(i) | d(i):D, r(i):R • $\mathcal{B}$(d(i),r(i))]

    is a map comprehension expression which denotes the map from definition set values, `d(i)`, to map range set value, `r(i)`, for which $\mathcal{B}$`(d(i),r(i))` holds.

61. **Map Enumeration:** A map [ι 59,π 11] enumeration of n map elements, expressed by expressions $a_1 \mapsto b_1, a_2 \mapsto b_2, ..., a_n \mapsto b_n$, for $n \geq 0$, is:

    $[\ a_1 \mapsto b_1, a_2 \mapsto b_2, ..., a_n \mapsto b_n\ ]$

    where, of course, the elipses, ..., must be filled in with proper expressions. Cf. [ι 12,π 6], [ι 54,π 11] and [ι 87,π 15].

62. **Map Operation:** These are the map operations

[12] Beware of the possible confusion of names: Domain as in [ι 24,π 7] and **dom**ain - as here !

- `m_a` ∪ `m_b` map union[13],
- `m_a` † `m_b` map overide[14],
- `m\s` map restriction[15],
- `m/s` map restriction[16],
- **dom** `m` map definition set[17],
- **rng** `m` map range set[18], and
- `m(x)` application[19].

63. **Mathematics:** By *mathematics* we shall here understand a such human endeavours that makes precise certain facets of language, whether natural or 'constructed' (as for mathematical notation), and out of those endeavours, i.e., mathematical constructions, also called theories, build further abstractions.

64. **Metaphysics:** is the branch of philosophy that examines the fundamental nature of reality, including the relationship between mind and matter, between substance and attribute, and between potentiality and actuality [22] `[Wikipedia]`.

    One may claim that this paper is [also] about metaphysics.

65. **Mereology:** is the theory of part-hood relations: of the relations of part to whole and the relations of part to part within a whole [28, 27, 10].

    The concept of 'mereology' and its study is accredited to the Polish mathematician, philosopher and logician Stanisław Leśniewski (1886–1939).

66. **Method:** By a method we shall understand a *set of principles* for *selecting* and *applying* a *set of techniques* *using* a *set of tools* in order to *construct* an *artefact* [DB].

67. **Methodology:** is the comparative study and knowledge of methods [DB].

    [The two terms: 'method' and 'methodology' are often confused, including used interchangeably.]

68. **Model:** A mathematical model is a description of a system using mathematical concepts and language. We shall include descriptions[20], prescriptions[21] and specifications[22] using formal languages in presenting models.

69. **Modelling:** Modelling is the act of creating models, which include discrete mathematical structures (sets, Cartesians, lists, maps, etc.), and are logical theories represented as algebras. That is, any given `RSL` text denotes a set of models, and each model is an algebra, i.e., a set of named values and a set of named operations on these. Modelling is the engineering activity of establishing, analyzing and using such structures and theories. Our models are established with the intention that they "model" "something else" other

[13] **dom** `m_a` and **dom** `m_b` do not overlap.
[14] **dom** `m_a` and **dom** `m_b` may overlap: resulting map "obeys" the overlap mapping.
[15]
[16]
[17]
[18]
[19]
[20] as for domains
[21] as for requirements
[22] as for software

than just being the mathematical structure or theory itself. That "something else" is, in our case, some part of a reality[23], or of a construed such reality, or of requirements to the, or a reality[24], or of actual software[25].

14 ℕ

70. **Narration & Formalisation:** To communicate what a domain "is", one must be able to narrate of what it consists. To understand a domain one must give a formal description of that domain. When we put an ampersand, &, between the two terms we mean to say that they form a whole: not one without the other, either way around! In our domain descriptions we enumerate narrative sentences and ascribe this enumeration to formal expressions.

15 𝕆

71. **Ontology:** is the branch of metaphysics dealing with the nature of being; a set of concepts and categories in a subject area or domain that shows their properties and the relations between them [7, 8] `[Wikipedia]`. An ontology identifies and distinguishes concepts and their relationships; it describes content and relationships [Bob Bater]. Ontologies *specify*.

    The two terms, taxonomy and ontology relate. We refer to the term 'taxonomy', Item 104 on page 17.

16 ℙ

72. **Part:** By a *part* we shall understand a solid endurant existing in time and subject to laws of physics, including the *causality principle* and *gravitational pull*[26].

73. **Part Set:** A compound part [ι 15,π 6] is a part set, same as a set part, [ι 89,π 15], consists of a [finite or infinite] set of parts.

74. **Perdurant:** Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time*.

75. **Phenomenon:** By a phenomenon we shall understand a fact that is observed to exist or happen .

Some phenomena are rationally describable – to a large or full degree – others are not.

76. **Philosophy:** is the study of general and fundamental questions about existence, knowledge, values, reason, mind, and language. Such questions are often posed as problems to be studied or resolved `[Wikipedia]`.

[23]— as in domain modelling
[24]— as in requirements modelling
[25]— as in software design

77. **Pragmatics:** studies the ways in which context contributes to meaning.

    Pragmatics encompasses speech act theory, conversational implicature, talk in interaction and other approaches to language behaviour in philosophy, sociology, linguistics and anthropology [25, 24] [Wikipedia].

78. **Principle:** By a *principle* we shall, loosely, understand *(i) elemental aspect of a craft or discipline, (ii) foundation, (iii) general law of nature, etc* [www.etymonline.com].

79. **Program:** By a *program* we shall understand ***

80. **Programming:** By *programming* we shall understand ***

17 ℚ

18 ℝ

81. **Requirements:** By a requirements we understand (cf., [17, IEEE Standard 610.12]): *"A condition or capability needed by a user to solve a problem or achieve an objective"*

    In *software development* the requirements explain what properties the desired software should have, not how these properties might be attained. In our, the *triptych* approach, requirements are to be "derived" from domain descriptions.

82. **Requirements Engineering:** is the engineering of constructing requirements [DB].

    The aim of requirements engineering is to **design the machine.** Chapter **??** covers requirements engineering.

83. **Requirements Prescription:** By a requirements prescription we mean a document which outlines the requirements that some software is expected to fulfill.

84. **Requirements Specification:** By a requirements specification we mean the sames as a requirements prescription.

19 𝕊

85. **Set:** In mathematics [ι 63,π 12], a set is a collection of different things; the things are elements or members of the set and are either domain objects [ι 24,π 7], or are mathematical objects: numbers, symbols, points in space, lines, other geometric shapes, variables, or other sets. A set may be finite or infinite. There is a unique set with no elements, called the empty set; a set with a single element is a singleton.

"Examples" of sets:

- a band of musicians
- a gang of outlaws
- a pack of dogs
- a swarm of flies
- a group of people
- a flock of geese
- a bunch of crooks
- a herd of cattle
- a pride of lions
- a crew of sailors
- a a mob of hair
- a a school of dolphins

86. **Set Comprehension:** Let TE be a type name or type expression, and $\mathcal{B}(e)$ be a predicate expression then

    $$\{\ e \mid e : TE \bullet \mathcal{B}(e)\ \}$$

    is a set comprehension: the set of all e in $e : TE$ that satisfies predicate $\mathcal{B}(e)$.

87. **Set Enumeration:** A set [ι 85,π 14] enumeration of n elements, expressed by expressions $e_1, e_2, ..., e_n$, for $n > 0$, is:

    $$\{\ e_1, e_2, ..., e_n\ \}$$

    where, of course, the elipses, ..., must be filled in with proper expressions.Cf. [ι 12,π 6], [ι 54,π 11] and [ι 61,π 11].

88. **Set Operation:** These are the operations on sets, s:

    - $s_1 \cup s_2$: union
    - $s_1 \cap s_2$: intersection
    - $s_1 = s_2$: equal
    - $s_1 \neq s_2$: inequal
    - $s_1 \subset s_2$: proper subset
    - $s_1 \subseteq s_2$: subset
    - **card** s: cardinality
    - $e \in s$: membership

89. **Set Part:** A compound part [ι 15,π 6] is a set part, same as a part set, [ι 73,π 13], consists of a [finite or infinite] set of parts.

90. **Set Type:** If T is a type name or type expression, then T-**set** is a type expression and denotes the set of all subsets of T.

91. **Science:** is a systematic activity that builds and organizes knowledge in the form of testable explanations and predictions about the universe `[Wikipedia]`.

    Science is the intellectual and practical activity encompassing the systematic study of the structure and behaviour of the physical and natural world through observation and experiment.

92. **Semantics:** is the linguistic and philosophical study of meaning in language, programming languages, formal logics, and semiotics. It is concerned with the relationship between signifiers — like words, phrases, signs, and symbols — and what they stand for in reality, their denotation [9] `[Wikipedia]`.

    > There are basically three kinds of semantics, expressed somewhat simplistically:

- Denotational Semantics model-theoretically assigns a *meaning*, a *denotation*, to each *phrase structure*, i.e., *syntactic category*.
- Axiomatic Semantics or Mathematical Logic Proof Systems is an approach based on mathematical logic for proving the correctness of specifications.
- Algebraic Semantics is a form of axiomatic semantics based on algebraic laws for describing and reasoning about program semantics in a formal manner.

93. **Semiotics:** is the study and knowledge of sign process (semiosis), which is any form of activity, conduct, or any process that involves signs, including the production of meaning `[Wikipedia]`.

    A sign is anything that communicates a meaning, that is not the sign itself, to the interpreter of the sign. The meaning can be intentional – such as a word uttered with a specific meaning, or unintentional – such as a symptom being a sign of a particular medical condition. Signs can communicate through any of the senses, visual, auditory, tactile, olfactory, or gustatory `[Wikipedia]`. The study and knowledge of semiotics is often "broken down" into the studies, etc., of *syntax*, *semantics* and *pragmatics*.

94. **Sequence:** By a *sequence* we shall mean the same as a list $[\iota\, 52,\pi\, 10]$.

95. **Software:** is the is the set of all the documents that have resulted from a completed *software development: domain analysis & description, requirements analysis & prescription, software: software code, software installation manuals, software maintenance manuals, software users guides, development project plans, budget, etc.*

96. **Software Design:** is the engineering of constructing software [DB].

    Whereas software requirements engineering focus on the logical properties that desired software should attain, software design, besides focusing on achieving these properties *correctly*, also focus on the properties being achieved *efficiently*.

97. **Software Engineering:** to us, is then the combination of domain and requirements engineering with software design [DB].

    This is my characterisation of software engineering.

98. **Software Development:** is then the combination of the development of domain description, requirements prescription and software design [DB].

    This is my characterisation of software engineering.

99. **Solid:**

100. **Syntax:** is the set of rules, principles, and processes that govern the structure of sentences (sentence structure) in a given language, usually including word order `[Wikipedia]`.

We assume, as an absolute minimum of knowledge, that the reader of this paper is well aware of the concepts of `BNF` (*Backus Normal Form*) `Grammars` and `CFG`s (*Context Free Grammars*).

101. **Solid Endurant:** By a *solid* [or *discrete*] endurant [ι 30,π 7] we shall understand an endurant which is separate, individual or distinct in form or concept, or, rephrasing: have 'body' [or magnitude] of three-dimensions: length, breadth and depth [21, Vol. II, pg. 2046].

Same as *discrete endurant* [ι 22,π 7].

102. **State:** A state is here sen as a non-empty collection of endurants [of a domain] [ι 30,π 7].

103. **Syntax, Semantics and Pragmatics:** With the advent of computing and their attendant programming languages these concepts of semiotics has taken on a somewhat additional meaning. When, in computer & computing science and in software engineering, we speak of syntax, we mean a quite definite and (mathematically) precise thing. With the advent of our ability to mathematically precise describe the semantics of programming languages, we similarly mean quite definite and (mathematically) precise things. For natural, i.e., human languages, this is not so. As for pragmatics there is this to say. Computers have not pragmatics. Humans have. When, in this paper we bring the term 'pragmatics' into play we are referring not to the computer "being pragmatic", but to our pragmatics, as scientists, as engineers.

20 𝕋

104. **Taxonomy:** is the practice and science of classification of things or concepts, including the principles that underlie such classification `[Wikipedia]`.

     A taxonomy formalizes the hierarchical relationships among concepts and specifies the term to be used to refer to each; it prescribes structure and terminology.

     Taxonomies *classify.*

105. **Technique:** By a *technique* we shall, loosely, understand *(i) formal practical details in artistic, etc., expression, (ii) art, skill, craft in work"* [www.etymonline.com].

106. **Technology:** is the sum of techniques, skills, methods, and processes used in the production of goods or services or in the accomplishment of objectives, such as scientific investigation `[Wikipedia]`.

Technology can be the knowledge of techniques, processes, and the like, or it can be embedded in machines to allow for operation without detailed knowledge of their workings. Systems (e.g. machines) applying technology by taking an input, changing it according to the system's use, and then producing an outcome are referred to as technology systems or technological systems `[Wikipedia]`.

107. **Tool:** By a *tool* we shall, loosely, understand *(i) instrument, implement used by a craftsman or laborer, weapon, (ii) that with which one prepares something, etc.* [www.etymonline.com].

108. **Triptych:** The *triptych [of software development]* centers on the three 'engineerings': domain, requirements and software [DB].

109. **The Triptych Dogma**

In order to *specify* $\mathcal{S}$oftware, we must understand its requirements.
In order to *prescribe* $\mathcal{R}$equirements, we must understand the $\mathcal{D}$omain.
So we must study, analyse and describe domains.

110. **Type:** By a *type* we shall, loosely, understand a possibly infinite set of values.

Example 110: Examples of types, T, are:

- natural numbers, **Nat**,
- integers, **Int**,
- reals, **Real**,
- Booleans, **Bool**,
- finite sets, T-**set**,
- possibly infinite sets, T-**infset**,
- Cartesians, T1×T2×...×Tn,
- finite lists, T^+,
- possibly infinite lists, T^ω,
- maps, $\mathsf{Td} \overset{m}{\rightarrow} \mathsf{Tr}$,
- total functions, Td→Tr,
- partial functions, $\mathsf{Td} \overset{\sim}{\rightarrow} \mathsf{Tr}$.

111. **Type Definition:** Let `T` be a type names, i.e., identifier, and TE a type expression, cf. [ι 112,π 18], then these are type definitions:

type T [further unspecified type], **type** T = TE ["concretized" type]

112. **Type Expression:** Let `T1, T2, ..., Ti, ..., Tn` be type names, cf. Example [ι 110,π 18], then these are type expressions `TE`:

- `Ti`,
- `TEi`-**set** [27],
- `TEi`-**infset** [28],
- `TE1`×`TE2`×...×`TEm` [29],
- `TE`* [30],
- `TE`$^\omega$ [31],
- `TEi` $\overset{m}{\rightarrow}$ `TEj` [32],
- `TEi`→`TEj` [33],
- `TEi`$\overset{\sim}{\rightarrow}$`TEj`, [34] and
- `TE1`|`TE2`|...|`TEm` [35]

21 ∪

113. **Unique Identifier:**

[27] the set of all finite sets of type `TEi`
[28] the set of all finite and infinite sets of type `TEi`
[29] the set of all Cartesians over `TE1, TE2, ..., TEm`
[30] the set of all finite lists over `TE`
[31] the set of all finite and infinite lists over `TE`
[32] the set of all maps from `TEi` to `TEj`
[33] the set of all total functions from `TEi` to `TEj`
[34] he set of all partial and total functions from `TEi` to `TEj`
[35] the union set of types `TE1, TE2,...,TEm`

22 $\mathbb{V}$

23 $\mathbb{W}$

24 $\mathbb{X}$

25 $\mathbb{Y}$

26 $\mathbb{Z}$

27 Conclusion

References

[1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, England, 2009.

[3] Dines Bjørner. Manifest Domains: Analysis & Description. *Formal Aspects of Computing*, 29(2):175–225, March 2017. First Online: 26 July 2016. DOI 10.1007/s00165-016-0385-z.

[4] Dines Bjørner. Domain Analysis & Description – Principles, Techniques and Modelling Languages. www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf. *ACM Trans. on Software Engineering and Methodology*, 28(2):1–67, April 2019. 68 pages.

[5] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. Springer, Fall 2021.

[6] Dines Bjørner. Domain analysis & description – a tutorial. In *ICTAC 2025 Proveedings*, Lecture Notes in Computer Science. Springer, November 2025.

[7] M. Bunge. *Treatise on Basic Philosophy: Ontology I: The Furniture of the World*, volume 3. Reidel, Boston, Mass., USA, 1977.

[8] M. Bunge. *Treatise on Basic Philosophy: Ontology II: A World of Systems*, volume 4. Reidel, Boston, Mass., USA, 1979.

[9] Rudolf Carnap. *Introduction to Semantics*. Harvard Univ. Press, Cambridge, Mass., 1942.

[10] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[11] O.-J. Dahl, E.W. Dijkstra, and Charles Anthony Richard Hoare. *Structured Programming*. Academic Press, 1972.

[12] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[13] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.

[14] Charles Anthony Richard Hoare. Notes on Data Structuring. In *[11]*, pages 83–174, 1972.

[15] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[16] Charles Anthony Richard Hoare. Communicating Sequential Processes. Published electronically: `usingcsp.com/cspbook.pdf`, 2004. Second edition of [15].

[17] IEEE Computer Society. IEEE–STD 610.12-1990: Standard Glossary of Software Engineering Terminology. Technical report, IEEE, IEEE Headquarters Office, 1730 Massachusetts Avenue, N.W., Washington, DC 20036-1992, USA. Phone: +1-202-371-0101, FAX: +1-202-728-9614, 1990.

[18] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992, 1996, 1999.

[19] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England;, 1995.

[20] Michael A. Jackson. *Software Hakubutsushi: Sekai to Kikai no Kijutsu (Software Requirements & Specifications: a lexicon of practice, principles and prejudices)*. Toppan Company, Ltd., 2-2-7 Yaesu, Chuo-ku, Tokyo 104, Japan, 1997. In Japanese. Translated by Tetsuo Tamai (Univ. of Tokyo, tamai@graco.c.u-tokyo.ac.jp) and Hiroshi Sako; ISBN 4-8101-8098-0; xxv + 267 pages.

[21] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.

[22] Michael J. Loux. *Metaphysics, a Contemporary Introduction*. Routledge Contemporary Introductions to Philosophy. Routledge, London and New York, 1998 (2nd ed., 2020).

[23] ANSI X3.9-1966. The Fortran programming language. Technical report, American National Standards Institute, Standards on Computers and Information Processing, 1966.

[24] Jacob Mey. *Pragmatics: An Introduction*. Blackwell Publishers, 13 January, 2001. Paperback.

[25] Charles Sanders Peirce. *Pragmatism as a Principle and Method of right thinking: The 1903 Harvard Lectures on Pragmatism*. State Univ. of N.Y. Press, and Cornell Univ. Press, 14 July 1997.

[26] Wolfgang Reisig. *Understanding Petri Nets Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. 230+XXVII pages, 145 illus.

[27] Barry Smith. Mereotopology: A Theory of Parts and Boundaries. *Data and Knowledge Engineering*, 20:287–âĂŞ303, 1996.

[28] Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.