From Domains Towards a Logic of Universals: A Small Calculus for the Continuous Determination of Worlds

(Preprint)

Claus Brillowski

Alter Zollweg 108b, 22147 Hamburg, Germany
brillowski@logike.info

Abstract.

At the end of the 19*th* century, 'logic' moved from the discipline of philosophy to that of mathematics. One hundred years later, we have a *plethora of formal logics*. Looking at the situation form informatics, the mathematical discipline proved only a temporary shelter for 'logic'. For there is *Domain Theory*, a *constructive mathematical theory* which extends the notion of computability into the continuum and spans the field of all *possible* deductive systems. Domain Theory describes the space of *data-types* which computers can ideally compute – and *computation* in terms of these types. Domain Theory is constructive but only *potentially* operational. Here *one particular operational* model is derived from Domain Theory which consists of *'universals'*, that is, *model independent* operands and operators. With these universals, Domains (logical models) can be approximated and continuously determined. The universal data-types and rules derived from Domain Theory relate strongly to the first formal logic conceived on philosophical grounds, Aristotelian (categorical) logic. This is no accident. For Aristotle, deduction was type-dependent and he too thought in term of type independent universal 'essences'. This paper initiates the next 'logical' step 'beyond' Domain Theory by reconnecting 'formal logic' with its origin.

Key words: Logic, Domain Theory, Type-2 theory of Computability, Data-types, Modes of being, Aristotelian logic, Universals, Topological Information Storage, Problem of Induction.

Contents

Chapter 1. Introduction 1
1.1. A First Overview ... 1
1.2. ... And Further Details 7

Chapter 2. The Halting Problem Revisited 13
2.1. Tarski's Dilemma 14
2.2. The Hidden Third Assumption 16
2.3. Scott's Solution to the Halting Problem 17

Chapter 3. Preliminary Notes on Algebra and Algebraic Specification 19
3.1. Algebraic Specification of Abstract Data-Types 20

Chapter 4. The Derivation of the Initial Algebra 23
4.1. The Zero-Domain 24
4.2. Sorts and Terms 25
4.3. Virtual- and Actual- Membership 25
4.4. Constructors and Defined Operators 26
4.5. The Signature 27
4.6. The Implementation of the Initial Algebra 28

Chapter 5. Dynamic Domains 33
5.1. Examples 33
5.2. Elements and their Inverses 35
5.3. Deformation in Dynamic Domains 38

Chapter 6. Logic 43
6.1. The Small Calculus as a Deductive System 43
6.2. The Small Calculus and Aristotelian Logic 46

Appendix I: Domain Theory and its Foundation 47
6.3. Order Theory 47
6.4. Domain Theory 48

Bibliography 53

CHAPTER 1

Introduction

1.1. A First Overview ...

According to the manifesto of IFCoLog, "*The International Federation for Computational Logic*".[1], close to ten thousand people are working on the research and development of logic-related methods. From its Greek beginnings until the 19-century, there was only *one* formal logic, Aristotelian logic, in varying stages of formalization and extensions. When Frege and Russell replaced it with the technically much more advanced predicate calculus, they hoped to entrench logic even more.[2] But today there is a multitude of logics, and no *single* logic rules undisputed. This matters. The very idea of logic is, that we can address legitimately everything and anything in *one* way, 'logically'.[3] Today this idea has been abandoned, or is, at least, under threat. It is under threat, not because of the multitudes of logics, but by *Domain Theory*. It is intended by its designer Dana Scott, to be a *theory of types*; the theory of all (data-) types which can ideally be computed. But it is also an advanced theory of computation where 'computation' – deduction under finitary conditions – is defined purely *in terms of* data-types.[4] The central idea of logic is threatened by Domain Theory, because it is so general that it neither has data-types of its own, nor does it possess any kind of 'deductive engine'.[5] The significance of this development can intuitively be gauged, by turning to the original definition of 'deduction' through Aristotle. "A deduction is a discourse in which, certain things being stated, something other than what is stated follows of necessity from their being so."[6] The "certain things" one can state or assert, are various. One can assert, for example, the existence of physical objects, mathematical truths, or someone's belief in ghosts, to name but a few. In each case, that what "follows of necessity" will change accordingly. Now let us reconnect to the present. A *possible assertion* turns, after it has been asserted, into a piece of *data*. Deductions are therefore *naturally data-type dependent*. And, when we take Domain Theory into account, *totally* so: the moment a formal system turns operational,

---

[1] See `http://www.ifcolog.net` under the heading "About" and "Manifesto".

[2] Both subscribed to *logicism*, the doctrine, that mathematics is an extension of logic. See [**Irv10**].

[3] This general theses is expressed throughout the history of logic in different ways. For an overview see [**Hei89**] chapter 4.

[4] "The main purpose of the Theory of Domains ... is to give a mathematical model of a system of *types* – including function space types – and to explain a notion of *computability* with respect to these types. There are many possible theories of types, but the construction of domains is mean to justify *recursive definitions* at all types, and – most essentially – to justify recursive definitions *of* types." [**Sco82**]

[5] It is a conditional theory and based on a *condition of continuity*. S.Abramsky interprets this condition as follows: "The continuity condition ... reflects the fact that a computational process will only have access to a finite amount of information at each finite stage of the computation. If we are provided with an infinite input, then any information we produce as output at any finite stage can only depend on some finite observation we have made of the input." [**Abr08**, p.495]

[6] *Prior Analytics* 24b. Translated by A. J. Jenkinson in [**Ari84b**]

its (data-) objects and rules of derivation turn parochial. Thus by 'deduction', Aristotle has defined what we call 'formal reasoning' today, the very notion which is codified precisely in Domain Theory. But unlike in earlier times, 'logical generality' today is a matter of a mathematical understanding and no longer bound to the operational level. Is the situation terminal and must we renounce the founding idea of 'logic'?

Here the very existence of the Theory of Domains will be transformed into grounds for a *particular* logic with *model-independent* and thus *universal* data-types and operations. The existence of Domain Theory has two aspects, a formal, *notational* and an informal, *logical* one. To obtain an operational calculus for *model-independent* data-types and operations from Domain Theory, two strategic moves are needed. The basic definitions of Domain Theory make up a finite notational system. *Informatics* has ways to extract algebraic, structural rules from such systems, *irrespective* of the mathematical or logical ideas they are designed to communicate. The first strategic move consists in applying these techniques to the basic definitions of Domain Theory. The second strategic move concerns the intuitive base of Domain Theory. Scott's achievement lies in logic. He introduced a completely new kind of logical object, $\perp$, the 'undefined data-type' into logic. It is the only object which is required to exist by the axioms of Domain Theory. But didn't I say above, that Domain Theory has no data-types of its own? And now $\perp$, which is not part of any other mathematical theory, is required to exist? We just had a glimpse of the very special nature of this new logical object. From 'outside', $\perp$ is unique to Domain Theory. But from 'outside' there are no things as data-types. From 'inside', all data types are *at least* $\perp$. And if all data-types are $\perp$, $\perp$ alone has no power to differentiate. $\perp$ is a kind of logical zero. Domain Theory as it stands has only one zero-object. In the second strategic move, *many* zero-objects will be introduced. The algebraic approach together with the generalization of Scott's basic idea interact and support each other. This leads directly to the Small Calculus, which contains the rules for another minimum: *homeomorphic operations*, by which Domains can be approximately and continuously *deformed*. There is one *particular* operational model with *universal* properties waiting to be discovered, but a price has to be payed. The operands and operators in finite scenarios are only partially defined and can always be extended. For want of a better name, I will call this feature *approximately operational*.

The model-independent, universal data-types and rules thus derived from Domain Theory are strongly related to the first formal logic, Aristotelian (categorical) logic. How can this be understood? We have seen that Aristotle's notion of 'deduction' is 'type-dependent'. The philosophical counterpart to a (data-) type is a *mode of being*, *Seinsweise*. Formal categorical Aristotelian logic is a logic of universal 'types'. In the 20th century, philosophy parted ways into the 'analytic' and the 'continental' philosophic tradition of Europe.[7] And the tradition of thinking in terms of modes of being in philosophy is only alive in continental philosophy. Thus the question: how can the connection between the universal data-types and Aristotelian logic be understood? – has two answers, depending of the viewers 'camp'. In modern textbooks of logic, Aristotelian logic, if mentioned at all, is depicted as imprecise, incomplete and of historical value only.[8] That the old logic is 'history' and 'deficient', is a position

[7] See [**Fri00**].

[8] "Predicate calculus is to syllogism what a precision tool is to a blunt knife. (Non the less, whenever a new piece of equipment is introduced, there will always be found those, who prefer the outdated

of the analytical camp. In the continental tradition, Aristotle is correct and it is modern mathematical logic which is deficient. It is deficient, because it is based on a superficial and insufficient analysis of the notion of proposition and a misconceived role of mathematics versus language.[10] Both approaches lead to formal systems. The formal relationship between Aristotelian and predicate logic is studied in the *history of logic*. Aristotle put down his 'formal' logic in natural language sentences. Everything depends, on how these sentences are formalized. Most logicians follow Frege and transcribe the Aristotelian sentential forms into monadic predicate calculus. For example, the sentential form 'all S are P' is transcribed as $\forall(x)S(x) \to P(x)$. But when *all* deductive rules of Aristotelian logic are taken into account, this transcription fails. Is there a faithful transcription of Aristotelian forms into monadic predicate calculus at all? This question is one of (finite) combinatorics.[**Gla06**] There is exactly *one* possibility in which Aristotelian logic can be imbedded into monadic predicate calculus (barring cases of symmetry). However this embedding does not sustain the informal semantics of Aristotelian logic. For example, 'all S are P' maps to a normal form with 16 terms. Aristotelian logic can be transcribed into monadic predicate calculus, but this transcription is not a translation because no connection can be established between this transcription and the texts accompanying Aristotelian logic, or the intuitive semantics of the sentential forms. The damning conclusion: "Since the times of Frege and Russell, most textbooks have not treated the subject of transcription of Aristotelian logic into predicate logic adequately"[11]

Aristotelian logic is a deductive system which is best viewed as *unrelated* to mathematical logic. Here, Aristotelian logic is understood as a formal system which arose from the *philosophical* analysis of language. In contrast, modern logic arose from a *mathematical* analysis of language. That 'language' can always be viewed in these essentially different ways is a thesis of continental philosophy and cannot be treated here.[12] Both approaches have their own criteria and lead to *different sets of primitives of* formal logic. The starting point however is the same: typical simple propositions of natural language, like (S1) "this blackboard is black" or (S2) "all bodies are extended". But then the approaches diverge.

machinery with which they are familiar; and predicate calculus is unquestionably harder to learn.) There are no reasons other than historical ones for studying the syllogism; but this theory has been of importance in the history of both logic and philosophy, and perhaps therefore deserves a place in a modern logic course."[9]

[10] "With the help of mathematical methods people attempt to calculate the system of the connectives between assertions. For this reason, we also call this logic 'mathematical logic'. It proposes to itself a possible and justified task. However, what symbolic logic achieves is anything but logic, i.e., a reflection upon *logos*. Mathematical logic is not even logic of mathematics in the sense of defining mathematical thought and mathematical truth, nor could it do so at all. Symbolic logic is itself only a mathematics applied to propositions and propositional forms. All mathematical logic and symbolic logic necessarily place themselves outside of every sphere of logic, because, for their very own purpose, they must apply *logos*, the assertion, as a mere combination of representations, i.e., basically inadequately. The presumptuousness of logistic in posing as the scientific logic of all sciences collapses as soon as one realizes how limited and thoughtless its premises are." [**Hei67**] p.156

[11] [**Gla06**], Chapter 5.

[12] Finite parts of a language ('words') 'point' into a human dimension of understanding which is uniquely personal. But at any time the very same words can be taken *as determined* and free of indication.See *formale Anzeige* 'formal indication' in [**Hei92**].

Frege introduced predicates as truth-functions which characterize 'sets' of objects. The technical devices needed to make this machinery work are truth-functions, variables, logical connectors and quantifiers. They were obtained by Frege by analyzing the notation of mathematical statements and proofs. A natural-language predicate is assimilated to such a function which takes objects as arguments and returns a truth value. Grammatically (S1) and (S2) are both simple, but the Fregean approach enforces a *differentiation.* "This blackboard" is mapped to a constant in the range of the monadic predicate "black". (S2) is formalized into a complex proposition, an implication of the form $\forall(x)S(x) \rightarrow P(x)$.[13] The similarity of (S1) and (S2) in natural language is interpreted as merely linguistic and rejected on 'logical grounds'.

The philosophical analysis is based on *modes of being* of *concrete single objects* like 'Socrates'. I will give an example of the basic idea with help of a thought experiment. A plane explodes in mid-flight and the debris scatters. One unhappy passenger is on a trajectory towards the ground. The passenger *is* a body, that is, a physical object, and as such he moves in the trajectory determined by his mass and physical laws. But there are properties which do not belong to this physical type of level. Further differentiation is needed. Some physical objects are alive, some are not. The passenger in the trajectory is 'alive' at the beginning and 'dead' at the end of the trajectory when he hits the ground. The passenger *is* a living being and *as such is* a physical body. 'Living being', 'physical body' are *modes* of his being. Another mode is 'human'. The passenger *is* human, and because of that, his death is mourned by his loved ones. These modes by no means do 'exhaust' a particular concrete object. 'Socrates' as individuum has *infinite* many properties, *some* of which are *universal* ('human', 'living being').[14] On the technical level 'Socrates' corresponds to *one* value which has infinite many *valences.* 'Socrates' is *polyvalent.* Some of the valences are model-independent, *universal* and *stable*, some are not. Aristotelian formal propositions are specific binary relations between a 'subject' (S) and a 'predicate' (P). They are traditionally classified across two independent dimensions. Propositions are *universal* or *particular* ('quantity') and *affirmative* or *negative* ('quality'). In the Aristotelian tradition simple propositions (S1) and (S2) are deliberately formalized as *universal affirmative* propositions.[15] Aristotelian logic rests on the assumption, that the 'is' in 'Socrates is human' and 'a human is a living being' is *logically one and the same.* In other words, it rests on the thesis, that we can speak of and about everything in a valid way, irrespective of its mode/type of existence, *with help of the word 'is'.*[16] The activity of asserting such propositions was called (logical) *determination*: The subject (S) was being circumscribed better with the help of a predicate (P). Heidegger provided a very detailed *philosophical* interpretation of Aristotelian propositions.[17]. But on the formal side, everything is open: how do the rules of Aristotelian logic relate to polyvalent objects and relations? And especially:

[13] See [**Fre77**], "Begriffsschrift" p.23

[14] "For there is nothing to prevent the same thing from being both man and white and countless other things: but still if one asks whether it is true to call this a man or not our opponent must give an answer which means one thing, and not add that it is also white and large. For, besides other reasons, it is impossible to enumerate the accidents, which are infinite in number;" *Metaphysics* 1007a, Translated by *W.D. Ross* [**Ari84a**]

[15] [**Jos16**], chapter IX.

[16] The title of the overview of the thesis of [**Hei89**] chapter 4 is: "Die These der Logik: Alles Seiende läßt sich unbeschadet der jeweiligen Seinsweise ansprechen und besprechen durch das 'ist'."

[17] [**Hei95**] part B.

how do the values obtain their valences and what *are* the basic valences? This question is especially important, as traditionally Aristotelian logic was seen as a tool to organize and *store* knowledge or, (to use a modern word) *information*. The following lengthy passage by J.St. Mill will provide a suitable impression of this forgotten usage of Aristotelian logic. And it will articulate a key problem associated with this usage:

> "It must be granted that in every syllogism, considered as an argument to prove the conclusion, there is a *petitio principii*. When we say,
>
> All men are mortal,
> Socrates in a man,
> therefore
> Socrates is mortal;
>
> it is unanswerably urged by the adversaries of the syllogistic theory, that the proposition, Socrates is mortal, is presupposed in the more general assumption, All men are mortal: that we cannot be assured of the mortality of all men, unless we are already certain of the mortality of every individual man: that if it be still doubtful whether Socrates, or any other individual we choose to name, be mortal or not, the same degree of uncertainty must hang over the assertion, All men are mortal: that the general principle, instead of being given as evidence of the particular case cannot itself be taken for true without exception, until every shadow of doubt which could affect any case comprised with it, is dispelled by evidence *aliunde*; and then what remains for the syllogism to prove? That, in short, no reasoning from generals to particulars can, as such, prove anything, since from a general principle we cannot infer any particulars, but those which the principle itself assumes as known.
> This doctrine appearers to me irrefragable; and if logicians, though unable to dispute it, have usually exhibited a strong disposition to explain it away, this was not because they could discover any flaw in the argument itself, but because the contrary opinion seemed to rest on arguments equally indisputable. In the syllogism last referred to, for example, or in any of those which we previously constructed, is it not evident that the conclusion may, to the person to whom the syllogism is presented, be actually and *bona fide* a new truth ? Is it not matter of daily experience that truths previously unthought of, facts which have not been, and cannot be, directly observed, are derived at by way of general reasoning? We believe that the Duke of Wellington is mortal. We do not know this by direct observation, so long as he is not yet dead. If we were asked how, this being the case, we know the Duke to be mortal, we should probably answer, Because all men are so. Here, therefore, we arrive at the knowledge of a truth not (as yet) susceptible of observation, by a reasoning which admits of being exhibited in the following syllogism:
>
> All men are mortal,
> The Duke of Wellington is a man,
> therefore
> The Duke of Wellington is mortal.

> And since a large portion of our knowledge is thus acquired, logicians have persisted in representing the syllogism as a process of inference or proof, though none of them has cleared up the difficulty which arises from the inconsistency between that assertion and the principle that if there be anything in the conclusion which was not already asserted in the premises, the argument ia vicious."[18]

What does this circle signify? Seen from the scientific and analytic 'camp' the *problem of induction*, so forcefully stated by Mill, is genuine, but its crisp formulation is due to the lack of differentiation between (S1) and (S2).[19] But from the philosophical positions of Aristotle[20] and Heidegger, there is *no vicious circle*.[21] For Heidegger, the 'problem of induction' is created by a superficial understanding of 'proposition'. So far the situation. How does the Small Calculus relate to this problem? What creates the problem of induction is the assumption, that everything *is determined*. If one starts from determined 'existing' individuals, what is left is 'bottom -up' abstraction. The Small Calculus proceeds top-down by 'construction'. It functions with partial values and operators which can always be extended by *further determination*. The 'construction' is a *deformation*, a notion of topology. What is the connection between topology and logic?

In the philosophical tradition of logic, properties come in two flavors: *imperishable* (*aphthartos)* or stable (universal) like 'living being' or *perishable* (*phthartos*) and

[18][**Mil84**] p.120-121

[19]"Until about the middle of the previous century induction was treated as a quite specific method of inference: inference of a universal affirmative proposition (All swans are white) from its instances (a is a white swan, b is a white swan, etc.) . . . It is no longer possible to think of induction in such a restricted way; . . . One powerful force driving this lexical shift was certainly the erosion of the intimate classical relation between logical truth and logical form; propositions had classically been categorized as universal or particular, negative or affirmative; and modern logic renders those distinctions unimportant. . . . The classical problem if apparently insoluble was simply stated, but the contemporary problem of induction has no such crisp formulation." From The *Stanford Encyclopedia of Philosophy* 'induction', [**Vic06**]

[20]The same criticism as Mill's can be found in *Sextus Empiricus* (600AC) who ascribes it to *Pyrrho of Elis* (ca. 360-270BC), a contemporary of Aristotle: "And I will deal at present with the axiomatic arguments; for if these are destroyed all the rest of the arguments are overthrown as well, since it is from these that they derive the proof of their deductions. Well then, the premiss 'Every man is an animal' is established by induction from the particular instances; for from the fact that Socrates, who is a man, is also an animal, and Plato likewise, . . . So whenever they argue 'Every man is an animal, and Socrates is a man, therefore Socrates is an animal,' proposing to deduce from the universal proposition 'Every man is an animal' the particular proposition 'Socrates therefore is an animal,' which in fact goes (as we have mentioned) to establish by way of induction the universal proposition, they fall into the error of circular reasoning, since they are establishing the universal proposition inductively by means of each of the particulars and deducing the particular proposition from the universal syllogistically."[**Emp90**] Book II, chapter XIV, 196 Aristotle deals with this argument in the first chapter of the first book of *Posterior Analytics*.

[21]"One tends to translate the word *epagôgê* with 'induction'; this is nearly literal and correct, but from the point of the matter, that is, in its interpretation, it is totally wrong. *Epagôgê* does not mean to run through single facts and series of facts, in order to deduce from similar properties so encountered, common and 'universal' ones. *Epagôgê* means the introduction of that which comes into view by bending over the single object towards that which was there before – which is what? That which it is. Only if e.g. 'treehood' is already in view, can we detect single trees. Taking into view that and making manifest that which was there before like 'treehood', is *epagôgê*. *epagôgê* is a two-way 'accounting': first lifting the gaze and at the same time fixing onto that which comes into view. *Epagôgê* is that, what is alien to the scientific mind; scientists suspect a 'petitio principii'; [**Hei39**] p.242; translated by the author.

accidental like 'white'.[22] Topological properties like 'line-length', 'angle' and 'surface-area' stay invariant (stable) under rigid transformations.[23] Projective transformations preserve properties like 'incidence' and 'cross-ratio'. But both rigid as well as projective transformations are special cases of so called 'topological' or *homeomorphic* transformations. These transformations are so drastic, that the objects in question lose all their 'accidental' properties and only keep the *most stable* ones. Homeomorphic transformations can be intuitively envisioned as *elastic deformations*.[24]. Let us call that, what stays stable and thus *identical* in possible transformations 'the object'. Objects in different states of deformation are *different* and *non-equivalent but* identical. For example, "The path up and down is one and the same".[25] The "path up" and the "path down" are *different*; one is more arduous than the other. Yet 'they' are *one and the same – the path*. 'Objects' in this sense are *focus-points of identity*. What is being focused on in deformations is *identity through differences* and this identity is classically expressed through universal affirmative propositions.

Now everting is in place for a modern interpretation of Mill's "petitio". Asserting a concrete universal affirmation, a Domain is topologically deformed in a finitary way. This logical determination/deformation works 'two-ways': it is reflexive and partially constructs (extends) the objects it speaks about. For example, *before* the assertion: 'Socrates is human', the information cannot be derived and is merely potentially true. *After* the assertion, the logical space is deformed, the object 'Socrates' has become extended by the property 'human', which from now on is one of its 'imperishable' properties. In such a way a universal affirmation functions as a topological and logical *information-storing* operation. When Domains are deformed as described, the distinctions of 'lowest' and 'highest' predicates turn into *post hoc* distinctions. In any stage of the deformation, there will be 'predicates' which approximate only themselves and no other object (in the calculus). These are traditionally names of individuals ('ultra-filters'). In the other direction, there will be'highest' predicates, which are not a model of anything, (except $\perp$). The information-storing-process can go on infinitely, in which case an infinite limit will be reached. But for any finite number of universal affirmations as input, the recursion will terminate. The "petitio principii" is an infinite, but finitary approximable recursion.

1.2. ... And Further Details

A (universal) Turing Machine formally models the computing capabilities of a general-purpose computer. It can be studied for the class of integer functions over $\mathbb{N}$, called partial recursive functions, or for the class of languages L defined over finite strings $L \subseteq W(\Sigma)$. The strings appearing in the latter are called recursively enumerable sets (r.e. sets) or formal languages. Roughly thirty years ago, Dana Scott extended the theory of computability from functions over $\mathbb{N}$ to functions

[22] See [**Jos16**] chapter IV.

[23] A transformation is said to be rigid if it preserves the relative distance.

[24] For example, a cup with a handle can be elastically deformed into a 'doughnut' (torus) (and curiously, the human shape). 'Cup' and 'doughnut' are in *different* phases, yet from a topological point of view they are *identical* and *one and the same*. For an animated example see `http://en.wikipedia.org/wiki/Homeomorphisms`

[25] Heraclitus, quoted according to [**KRS99**] fragment 200.

between topological spaces. Any object x of Domain Theory is defined as the limit of a sequence $(x_i)_{i\in\mathbb{N}}$ of finite objects, i.e. $x = \lim x_i$. An x_i is seen as *approximation* to the (infinite) object x. If x approximates y, $x \geq y$, y is the *same or a better version of* x in terms of *qualitative information.* Why was the theory of computability extended?

The need for an extension arises, when one takes a *logical view* of the activity of programming. For practical purposes, computer languages can be (and most often are) defined *operationally only.* That is, by (precisely) cataloguing the effects of (basic) operators on data and the state of a computer. Ignoring many intermediate levels, the 'programmer' develops an intuitive understanding of these instructions and maps every-day chores onto them . This path leads to a culture of program design and an 'art of programming'. But from a logical point of view, the symbols of programming languages have *no meaning at all.* They are formal languages *only.* Logically, they are *undefined object-languages* in need of a *formal semantical definition.* For the latter, a *meta-language* is needed. Domain Theory was created to provide a mathematical space of values for 'the meanings' of operations which can be performed by computers, that is, as a *meta-language.* Formal languages are, when they have been defined in terms of Domain Theory, *models of* Domain Theory. A collection of techniques and a methodology for logically defining a programming language in terms of the denotations provided by Domains Theory, is called *Denotational Semantics.* 'Denotational Semantics' is not only used to define a *canonical standard model* of a particular programming language. Depending on the object-language, this logical definition is usually not operational. In this case a series of *non-standard models* are designed which are congruent to the standard semantic, each containing more implementation detail than the previous one. The connection between the abstract Domain Theory and a concrete computer is often depicted by a 'semantic bridge'[26].

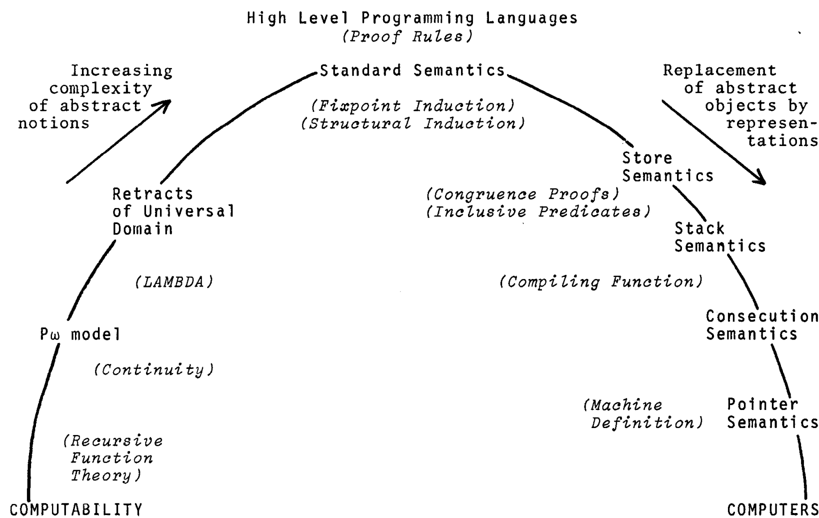


Fig. 14.2. The Semantic Bridge.

The semantic bridge connects the shores of computational *theory* and *practice.* A 'computer' is defined by its operations and data-types. Programming languages are

[26][**Sto77**] p.386.

conceived to bridge the gap between the machine and the way we naturally think. But "the way we naturally think" is not a logical category. Seen from logic, programming languages are conceived for arbitrary reasons and on intuitive and practical grounds. Only when furnished with a denotational canonical semantics have they graduated to the logical level. There are infinitely many possible programming languages and all have, in principle, a canonical model. But we desperately want models which are not only canonical but also operational, *'concrete'*. Flexible as we are, we don't really care, what canonical model we have, *as long as it is operational.*

Now operations can not only be defined by recording in a table their effects on objects. They can also be defined in a totally different way, namely *relative* to each other. When we look at how operations are related relative to each other, we take the 'algebraic view' of things. In extreme cases, as with the initial definitions of category theory, this has the effect, that objects can be done away with completely. But we do not need to go that far. From the algebraic viewpoint, mathematical theories range over a space of models. If, for example, the Peano axioms define the theory in case, in the range are all the different numerical systems (Roman, decimal, binary, . . .) which faithfully realize or model the properties put down by the axioms. The operations of each of these notational systems are very specific. On the other hand, algebraic operations are operations not for *one model only*, but for whole regions of models. And there are algebraic objects and operations which are *common to all models* of a given theory. They are the objects and operations of the *initial algebra.* The initial algebra makes the finitary 'minimal core' of a theory manifest. It is often called its *standard model.* The objects and operations of an initial algebra are unique and canonical, yet universal. Domain Theory allows for infinite limits. But they are not assumed or required. If they exist, they are *reducible* to (unions of) *finite* objects and can be *constructed.* The algebraic approach to Domain Theory will yield universal data-objects and operations of a *finitary and terminating* notational system, by which infinite objects (limits) can be generally approximated.

There is one more reason why the algebraic approach recommends itself. At the bottom of the left side of the semantic bridge are a small number of axioms which determine Domains, their objects and relations. When moving to the other shore over the bridge, the complexity increases dramatically. On the right side of the bridge are programs called 'compilers' which easily consist of tenths of thousandths of lines of code. The algebraic operators of the Small Calculus can be coded in around a hundred lines.

An initial algebra can be *derived* from the basic definitions of a mathematical theory. Initial algebras are isomorphic to term algebras. 'Term' refers to a *constant*, a *variable* or the *application* on an operator to a list of argument-terms of a *notational presentation* of a mathematical theory. The concrete derivation of an initial algebra from a mathematical theory consists to a large extent in the classification of the terms into appropriate algebraic sorts. The rest is a kind of mechanical computation, which the notations of the theory are subjected to: a *derivation.* The result of such a derivation is an *algebraic specification of abstract data types.* That is, an *algebraic theory* with a *signature*, *sorts* a *set of equations*, *membership-predicates* and *constructors.* In the case of Domain Theory there are obstacles. From functional programming we know, that functions can be viewed alternatively as operations or data. This is

but a secondary effect of a much deeper property. At the heart of Domain Theory lies a (con-) *fusion* between the notions of operation and operand. Algebra, on the other hand, relies on a clear distinction between operands and operators. Thus, when Domain Theory is turned into an algebraic theory, the two approaches clash and the 'confusion' between operand and operators must be dealt with explicitly. In an algebraic specification of abstract data-types, membership-predicates determine, whether a term is a member of a specific algebraic sort or not. Each canonical term may only appear *exactly once*. By partitioning Domain Theory into algebraic sorts, it is unavoidable, that sometimes *one* term of Domain Theory legally becomes a member of *two* or more algebraic *sorts*. This kind of 'cross-sort identity' cannot be allowed. To handle these cases, new *actual*- and *virtual*- membership predicates are introduced. With the help of these predicates, the derivation can be performed straightforwardly. But the result are objects, which lie 'halfway' between traditional objects and operators.

The derivation of the initial algebra yields a general mechanism for the approximative construction of any Domain. Yet the notational system so obtained cannot be used for anything, least of all for the approximation of infinite limits. An initial algebra *faithfully* caries over the objects of the mathematical theory into the realm of algebra. And the axioms of Domain Theory require only *one* such object. Thus we are rather short of material to approximate anything by.

This one object, $\perp$, is however, very, very special. It is a logical *zero-object*. What does this mean? Turing Machines are about *partially* defined functions on $\mathbb{N}$, but Domain Theory is about *totally* defined functions on spaces of the continuum. If f is a partial function which diverges on the argument w, we define $f(w) = \perp$. By doing so we treat $\perp$ not as a value of the object-language, but as a value of the meta-language. If it *were* be a value of the object language, there would be no halting problem. But in Domain Theory $\perp$ *is* an item of Domain Theory.[27] How is that possible? Has Scott has solved the halting problem?

The answer is 'no' and 'yes'. Mathematicians – and, since Scott, also logicians – think in terms of value-spaces and operations. If an operation does not have a solution in a certain value-space, a new value space may be set up, in which the problem does not appear. For example, the equation $x^2 + 1 = 0$ does not have a solution in the value-space of reals and this has given rise to the field of complex numbers. Scott applied this mathematical approach to algorithms and computability. There is no solution to the halting problem in $\mathbb{N}$ or $W(\Sigma)$. But in the realm of Scott-Domains, the halting problem does not arise, due to the way its objects are defined. We saw, that they are defined in terms of limits and approximations, i.e., $x = \lim x_i$. And at index 0, $x_0 = \perp$ for *any* Domain object. Aside from $x_0 = \perp$, the existence of other, *extended objects* in Domain Theory is *conditional*. To understand this aspect let us take a look at logic.

Logic is about theorems and there are two classes of them. One class is made up of theorems which are called tautologies; they require *no assumptions* for their proofs.

[27] G.Plotkin writes in *The Pisa Lectures*: "As it is well-known, Scott introduced an element $\perp$ into the value domain V, rather than worrying about partial functions and undefined results: to coin a phrase, he objectifies (some of) the metalanguage." [**Plo83**] p.3

And there are theorems which need *assumptions* to be proven. These theorems are only true on *condition* that some other assumptions hold. In the case of Domain Theory and Denotational Semantics, it is the programming language which is defined, which provides the conditions for the extended objects of Domains. Programming languages vary greatly, but each comes with its own kinds of objects, like *booleans* or *integers*. Denotational Semantics translates them into existing extended objects of Domains.

When assumptions are introduced in logic, we commit ourselves to models, where these assumptions hold; and others are excluded, where they do not hold. Let us now look at $\perp$ in terms of this 'modeling' activity. Because *all* elements of Domains 'are' $\perp$ at index zero, $\perp$ does not determine or exclude *any* particular object/model. $\perp$ is a *logical zero*.

An initial algebra is about a global, minimal core of a mathematical theory. And Scott introduced a logical zero-object into logic. The Small Calculus rests on a combination of the two 'minimalist' approaches. We are short of data-material, but we can introduce as many zero-objects as we need! They are collected in the *Zero-Domain*. What is the result of this move? In normal calculi, the assertion of relations requires objects. Zero-objects are not objects in this sense, they precede them. But if we have several, they can be used in assertions of relationships. And the assertion of an approximation relation between Zero-objects leads to *better defined* objects. In formal systems, when virtual objects/facts are asserted, they are endowed with assertive 'reality' in one single step. The new assertions do not have the weight of these one-step assertions. They produce more determined versions of objects/relations *on the way to* a 'final', possible infinite, object. By using zero-objects, models are can now be determined *gradually* and *continuously* by ever increasing amounts data from a Domain of zeros. The class of Domains which are computed by the The Small Calculus in such a way are called *Dynamic Domains*. Some properties of Dynamic Domains:

(1) The $\geq$ operand-operator is finite and distributes information over the elements (filters) of Dynamic Domains. The rules for the distribution are *reflexive*, *recursive*, *terminating* and *confluent*. An qualitative information closure is computed in finite time.

(2) The $\geq$ operand-operator is *homeomorphic*. That is, Domains which are 'constructed' by asserting information-approximations by closing up under its rules are *deformed* under the increase (decrease) in information.

In chapter 6 the Small Calculus is interpreted as a deductive system. The Small Calculus now 'deduces' all the finite information from a finite set of logico-algebraic premisses/assumptions. The feedback loop between premisses ('relations') and terms ('objects') makes for a stark difference to traditional formal systems. In the Small Calculus, in the course of deduction, *additional terms* may appear. What does that mean? The expressions of a conventional formal system are split into two: *possible* and *actual* theorems. The connection is provided through *proofs*. Possible theorems are all defined *statically in advance*. Possible theorems must be well-formed, and this question can always be decided, as it does not depend on the deductive properties of the formal system. In the Small Calculus *it does*. Terms – which correspond to possible

theorems – are created *dynamically*. Correspondingly, the set of all deductions of the Small Calculus is called *dynamic closure*. The set of deductions with a predefined set of terms is called *static closure*. The rules, by which the *static closure* of the Small Calculus is computed is congruent to the core rules of *Aristotelian syllogistic*.

Replacing Aristotelian logic by predicate calculus paved the way to Domain Theory. In Domain Theory, approximation is a primitive and is interpreted as qualitative information. By this feature, Domain Theory reconnects to Aristotelian logic. Aristotelian propositions are in subject-predicate form, *because* 'approximation' is a primitive. Properties come in 'flavors' of stability, *because* of the topological nature of Domains. The most stable ones are homeomorphic properties, which are universal and common to all models. By deforming Domains, information is stored in them topologically. Relative to the information stored already (starting with the Zero-Domain), information is either *new* and ('synthetic') or 'analytic', that is, deducible in finite space and time. If logic is conceived as a formal language with a static set of possible theorems, all these distinctions are meaningless. The Small Calculus is a first step beyond formal languages towards a class of non-recursively enumerable, but approximable languages.

CHAPTER 2

The Halting Problem Revisited

The 'halting problem' is a special kind of effective method or algorithm. To define 'algorithm', a normalization-system for algorithmic notation must be set up. There are many different such systems which were found to be translatable into each other (e.g. Universal Turing Machine, Lambda Calculus, ...). What is common to all such normalization-systems is the following: (1) Each such algorithm is written down as a finite string (in which the particular effective method is described). (2) There is a finite set of basic or prime-methods, which all the complex methods expressed by the strings must be composed of. (3) All these primary methods are called 'algorithmic'. (4) There are rules for combining methods to new methods and these rules preserve the property 'algorithmic'. (5) By virtue of this setup, the set of all algorithmic methods can be defined inductively. Such a set of algorithms is enumerable. The various normalization-systems differ in the kind of data they allow and the sort and amount of basic operations they start from, and the combination rules. Any definition of 'algorithm' happens on the floor of such a concrete normalization system. These concrete systems are called *models of computability.* A physical computer can be seen as a deficient realization of such a specific model. Its actual computations take time and the storage space is limited. The *theory of computability* is about properties common to all models of computability. One such property is 'termination'.

A 'computation' is the actual performance or 'execution' of an algorithm. A computation may stop after some finite time – 'terminate' – or not. This distinction plays an essential role in the notion of computable function.'Computable function' is an amalgamation of the notion of'algorithm' and 'formal language'. To describe the notion of computable function, we first will set up a formal language which describes functions. A function is a special kind of binary relation. A binary relation is a subset of a cartesian product of two sets, $\rho \subseteq X \times Y$. We write $x\rho y$ instead of $(x, y) \in \rho$. A function or *mapping* from X to Y is a triple

$$f = (X, Y, \rho)$$

where X, Y are sets and ρ is a relation such that $(x\rho y)$ *and* $(x\rho y')$ *then* $y = y'$ for *all* $x \in X$ *and* $y, y' \in Y$. The relation ρ is called *graph* of f.

Now that we know what functions are, we can define a formal language for them and integrate the notion of algorithm in such a way, that it 'functions' as generator for the graph.

Definition 2.0.1 (Computable function). Let $f : L \to L'$ with $L \subseteq W(\Sigma_1)$ and $L' \subseteq W(\Sigma_2)$. f is totally computable, *iff* there exists a terminating algorithm for f which computes the mapping $f(w)$ for all $w \in L$. If such an algorithm exists only for a true

subset $L'' \subset$L, then f is called partially computable. If $w \in L''$ is such an argument, we say that $f(w)$ is undefined or diverges and write $f(w) = \perp$.

'Algorithm' and 'function' are connected by the idea, that an x, when handed over to an algorithm as input, will be transported to the output y by an algorithm on termination. The result of this fusion is the notion of *computable function*. Let us call the functions which can be defined in such a way, *Formal Language Definable* (FLD) functions. It would be very satisfactory if the set of FLD- and computable functions would coincide; or at least, that the set of computable functions could be clearly demarcated from non-computable ones. Famously, neither is the case. The reason lies in the 'halting problem' which defines a 'non-halting' function for any given normalized computational model. Here the halting problem is defined with Turing Machines.

Theorem 2.0.2 (Halting Problem)**.** *Let M_y be the $y-th$ Turing Machine in a standard enumeration and f_y the function which is computed by M_y. No Turing Machine can compute the total function $g : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ which is defined as:*

$$g(x, y) = \textbf{\textit{if}}\ f_y(x) \neq \perp\ \textbf{\textit{then}}\ 1\ \textbf{\textit{else}}\ 0.$$

Proof. By contradiction. Suppose that g is computable. Thus the partial function h, defined as

$$h(x) = \text{ if } g(x, x) = 0 \text{ then } 1 \text{ else } \perp,$$

is computable. h 'switches' the result of g. Once we have a Turing Machine M_y which computes g, we can modify it to M_x to compute h. To derive M_x from M$_y$, it is sufficient to transform the result 0 of M_y into 1 and to force the new machine never to halt, when M_y would produce 1. Let f_x be the corresponding function for M_x with $f_x = h$ in the standard enumeration. Then by definition of h, if $h(x) = f_x(x)$ is defined, $h(x) = 1$ and $g(x, x) = 0$. But by definition of $g(x, y)$, this implies $f_x(x)$ to be undefined. Otherwise, if $h(x)$ is undefined, $g(x, x) = 1$ which in turn means that $f_x(x) = h(x)$ is defined. In both cases we obtain a contradiction and we must conclude that g is not computable. □

This result fits well with other known facts of mathematics. The cardinality of all functions $\{f | f : \mathbb{N} \to \mathbb{N}\}$ is $2^{\aleph_0}$, the cardinality of the continuum. The computable functions on the other hand are recursively enumerable. Thus the majority of functions cannot be enumerated – and g happens to be a concrete instance of a *non-computable* function.

This is how the halting problem is explained in most books on computability. From this position, Scott's contribution to computability and logic is very hard or impossible to understand. That is because this description makes use of an unnecessary assumption. Dropping this assumption opens the way to Domain Theory and a completely different perspective. In order to 'flush out' the unnecessary assumption, we move to logic.

2.1. Tarski's Dilemma

With the 'halting problem', we have encountered a general recipe for the construction of problems which are known as 'antinomies'. An antinomy is a collision between two possible assertions, of which each on its own seems true, provable or plausible, but when taken together express an impossibility. From this situation, the logician A. Tarski draw conclusions which are widely accepted today but which must be revised

in the light of Scott's work. In the paper: *The Semantic Conception of Truth*[**Tar86**], Tarski constructs an antinomy according to the method of the halting problem above.[1] Instead of numbers $0, 1$, truth-values *true* and *false* are used. With truth values, the halting problem turns into the *antinomy of the liar.* In chapter eight, Tarski analyzes the implicit assumptions which lead to the antinomy. He finds three assumptions, of which the following two are necessary for the antinomy to occur.

> "I. We have implicitly assumed that the language in which the antinomy is constructed contains, in addition to its expressions, also the names of these expressions, as well as semantic terms such as the term 'true' referring to sentences of this language; we have also assumed that all sentences which determine the adequate usage of this term can be asserted in the language. A language with these properties will be called "semantically closed."
> II. We have assumed that in this language the ordinary laws of logic hold.
>
> ... Since every language which satisfies both of these assumptions is inconsistent, we must reject at least one of them;"

It is widely agreed, that antinomies must be avoided in formal systems at all costs. They lead to a contradiction, and from a contradiction anything can be proven, nullifying any purpose a calculus might have. As these assumptions together create the antinomy, so Tarski, one must be rejected. It is unacceptable to reject assumption II. Therefore I must be rejected. Thus "...we decide not to use any language which is semantically closed in the sense given".[2] And: "Since we have agreed not to employ semantically closed languages, we have to use two different languages in discussing the problem of the definition of truth and, more generally, any problems in the field of semantics." Now the central logical division is introduced, for which Tarski is justly famous.

> "The first of these languages is the language which is "talked about" and which is the subject matter of the whole discussion; the definition of truth which we are seeking applies to the sentences of this language. The second is the language in which we "talk about" the first language, and in terms of which we wish, in particular, to construct the definition of truth for the first language. We shall refer to the first language as 'the object language', and to the second as 'the meta-language.' "

The distinction between object- and meta-language is standard today. In the context of partial functions, $\perp$ is a value of the meta-language.

'Tarski's dilemma' is, that *either* we must forgo closed (formal) languages (I), *or* we must change the underlying logic (II). Both strategies for handling antinomies have been tried. *Constructive logic* renounces the *tertium non datur* which is essential in

[1] [**Tar86**], chapter seven.

[2] Taski writes: "It would be superfluous to stress here the consequences of rejecting the assumption (II), that is, of changing our logic (supposing this were possible) even in its more elementary and fundamental parts. We thus consider only the possibility of rejecting the assumption (I). Accordingly, we decide not to use any language which is semantically closed in the sense given."

generating the contradiction. Yet the problem of the halting problem runs deeper than Tarski's analysis. Domain Theory introduces a *third* and completely different strategy of handling this logical antinomy. Tarski's dichotomy is incomplete. There is a third assumption involved in the construction of the antinomy, which is a matter of course to Tarski, to such an extent, that he overlooks it.

2.2. The Hidden Third Assumption

In order to make this hidden assumption manifest, we turn to the *Grelling-Nelson antinomy* of heterological terms. We will derive the contradiction in the standard way and then give a contradiction-free interpretation.

The antinomy of Grelling-Nelson works as follows. All predicate-terms of a natural language (here English) are divided into 'syntactic' and 'non-syntactic' expressions. Syntactic predicates are those which apply to language expressions. Non-syntactic are called predicates which do not apply to languages. A syntactic predicate is called *autological* if it applies to itself. An example is 'multisyllabic'. Multisyllabic applies to itself because the word consists of several syllables. Predicates which do not apply to themselves are called *heterological*. The following list shows some examples.

Autological	Heterological
short	long
meaningful	monosyllabic
English	Anglais

First we construct the antinomic contradiction. The question: whether 'heterological' is autological or heterological has two possible answers. First we assume that the answer is 'autological'. Then by definition it must apply to itself. However, heterological are only predicates which do not apply to themselves. Ergo: the answer 'autological' is contradictory. Now we assume that the answer is 'heterological'. Then it can be applied to itself and therefore it is autological. Again we have a contradiction. As 'autological' and 'heterological' are by definition the only two possible answers, we have proved that the antinomy led to a contradiction.

We will now *program* the Grelling-Nelson antinomy. What will the answer of the computer be? In a fictive functional programming language we will define the function *het* (short for heterological) according to the definition as

$$het(y) = \textbf{If } y(y) = true \textbf{ then } false \textbf{ else } true.$$

If this function is applied to 'short' it returns 'false', as $short(short)$ is true. By similar argument, $het(monosyllabic) = true$. Now we apply *het* to itself. *het* is a well defined function and a computer will try to evaluate it. In the course of the evaluation, the functional mechanism will substitute y in the expression

$$\textbf{if } y(y) = true \textbf{ then } false \textbf{ else } true$$

through *het*. That is, we get $het(het) = \textbf{if } het(het) = true \textbf{ then } false \textbf{ else } true.$

In order to compute the outer *het*(*het*) is expanded to *het*(*het*) of the next level. In order to compute that, the functional mechanism will substitute y in the definition with *het* ... and so on. *The process of evaluation never ends.* The computation does *not* lead a *contradiction* but an *infinite regress.* What does this signify?

In order to 'see' the contradiction, we must 'jump' across an *infinite gap* from the beginning of the computation and its fictive result: we must assume, that the *infinite* evaluation sequence has one of two predefined *finite* values. Therein lies the hidden assumption which is necessary to arrive at the contradiction. Without this assumption that one can jump across the infinite gap, the contradiction of the antinomy is *purely fictitious.* As the reason for introducing object-/meta-level distinctions lies in the conception that antinomies are destructive, there is no need to make use of this distinction any longer. So, instead of renouncing assumption I that languages cannot be semantically closed, we renounce the idea, that the antinomy of the liar leads to a contradiction.

2.3. Scott's Solution to the Halting Problem

Once we realize, that the meta-language value $\perp$ functions as a fictitious finite *value* of an *infinite process*, we can see that the contradiction itself is fictitious. It only 'arrives' at $\perp$, when the infinite sequence is completed. And that is, by definition, never. Let us now make a thought experiment and introduce a new value $\perp'$ in the object-language. $\perp'$ is supposed to correspond to $\perp$ of the meta-level. If $\perp'$ is added to the range of finite values of FLD-functions, it *closes up* and logically complements the value space for computable functions. Either a function is defined on the original value-space and terminates, or it does not terminate, $\perp'$. *Tertium non datur.* But of course, this is purely fictitious. We have no way of using $\perp'$ as the value of a function because of the halting problem. We now came to the truly ingenious part of Scott's approach. In order to force $\perp'$ into a value of elements of the object-language, he changed and generalized the notion of 'value'. The objects of standard mathematics possess exactly one value, they are *monovalent.* For example, the booleans, natural numbers, fractions, ... Not so the values of Domains. They can possess more than one valency, they are polyvalent, and these valencies of *a* value are arranged in a partial order. By using that device Scott solved the halting problem. On the meta-level $\perp$ is the finite value of a non-terminating, infinite sequence. The value $\perp$ can never be observed after a finite time by definition, all the other values can. The values in Domains are all polyvalent and in particular, all values are $\perp'$ at index 0. Besides $\perp'$, they can be something 'better', eg. 'true', or 'false', a valency of a higher index. Because of this design, we can define $y_0 = \perp$ of $f(x) = y$ of any function f with any argument x. In the course of the computation, there are now two possibilities. The algorithm either stops and returns a polyvalent value which is better defined than $\perp'$. Or the algorithm does not stop. In which case the default-value of a function $\perp'$ coincides with the meta-value $\perp$, 'after' the infinite execution sequence is fictitiously completed. As $\perp'$ and $\perp$ coincide, we will not distinguish between them any longer. In such a way Scott solved the halting problem and by doing so, connected the world of computers and computability to the world of mathematics. What are the consequences?

As long as we use $\mathbb{N}$ as a domain for algorithms and programs, we must reason with the properties of *partial functions.* From the mathematical point of view, this is a disaster:

For *total functions* are subject to well understood algebraic laws, partial functions are not.[3] Scott mathematically *extended* the notion of 'computable function', as 'complex numbers' are an extension of 'reals'. In the universe of polyvalencies, all computable functions compute totally defined maps. This opens up a world of possibilities. For example, now we can make use of *algebraic laws* when arguing about properties of programs. The theory, in which the algebraic laws of functions are spelled out, is Category Theory. Category Theory also functions like a hub through which many other mathematical theories can be accessed. This brings with it a further advantage: When making use of extended computation, we can tap into a fast pool of mathematical knowledge.

So Domain Theory and extended computability is a big step forward. And the title of one of his presentations of Domain Theory, *Lectures on a Mathematical Theory of Computation*[**Sco81**] is very apt.

[3] An example is functional composition.

CHAPTER 3

Preliminary Notes on Algebra and Algebraic Specification

In mathematics, variables are used in two, stinkingly different ways. In $x \in \mathbb{N}$ for example, x stands for a *definite* number. But an equation like $x + y = y + x$ can be understood without referring to particular domains like $\mathbb{N}$. Obviously, this equation also holds true for ratios and real numbers, but also for objects which are no numbers at all, like sets or booleans, if "+" is interpreted accordingly. Indeed one can start out from $x + y = y + x$ and investigate mathematics or our daily life, what kind of objects and operations are 'covered' by these *laws*. In this case the *variables* stand for *indefinite objects* and the operators for operator-types. In such a way, equations provide *finite ways* for managing indefinite multitudes. This perspective is developed in algebra.

An algebra consists of a set of elements and operations on the set. But to drive algebra beyond single types of objects (sorts), algebraic operations are defined in terms of how *two or more of them interact*. Modern algebra studies groups, rings, fields and other structures, and by doing so, considers *classes of algebras* having certain properties in common, properties. The notion of *free algebra* focuses on the algebraic classification of items into *definite* objects (constants) and *indefinite* ones (variables). Given any algebra, "...we can write T for the set of all *definite terms*, constituting the definite language, and $T[V]$ for the larger indefinite language permitting variables drawn from a set V in place of some of the constant symbols. When V contains only a single variable "x", $T[\{"x"\}]$ is usually abbreviated to $T["x"]$ or just $T[x]$ which is usually unambiguous. This convention extends to the algebra Ω of terms of T together with its list of operation symbols view as operations for combining terms; we write $\Omega[V]$ and call it *the free algebra* on V. The *initial algebra* Ω is $\Omega[\emptyset]$, the case of no variables."[1]

An initial algebra can be derived from the notational presentation of a theory and has many special properties. For example, they are unique up to isomorphism. Or they are independent of any particular representation. An algebra contains a representationally independent and initial, minimal 'core'.[2]

The objective is to derive the initial algebra for Domain Theory under the extension of the Zero-Domain from a notational presentation. To do so, we use techniques developed in the field of *algebraic specification of Abstract Data-Types*.[3]. The theory of *Abstract*

[1] Quoted from the *Stanford Encyclopedia of Philosophy*, Algebra, [**Pra08**].

[2] "Another way of looking at this result is to think of the initial algebra(s) as the hub or center of a wheel with all the other algebras of the class that interpret a signature placed radially around the wheels circumference. Each of these perimeter algebras is connected to the hub by a single spoke which is the unique homomorphism from the focal initial algebra(s) to that perimeter algebra. Every algebra of the class can therefore be reached or derived from an initial algebra." [**TM93**] in chapter: "Initial Algebras"

[3] Maude-Primer, [**CDE+07**]

Data-Types looks upon a *mathematical theory* as a specification which describes the properties and behavior of data-types to be implemented.

3.1. Algebraic Specification of Abstract Data-Types

A *signature* Σ consists of a set of sorts and a family of sets of operators defined on the sorts. An Σ-*algebra* is a mathematical structure which satisfies or 'fits' a signature Σ. It assigns to each sort a set of values called *carrier-set*. The family of carrier-sets associated with the sorts is called the *carrier* of the algebra. For each operation on the sorts there is a corresponding function over the carrier-set. An algebra which satisfies a signature is called a *realization* of the signature. With the help of *equations*, an algebra can be partitioned (or quotiented) into equivalent classes. A signature together with a set of equations is called a *presentation*. The 'meaning' of a presentation is the set of its possible algebras, or *models*. If an algebra is a model, it satisfies each of the equations. If E is a set of such equations, we write M^* for the set of all models which satisfy E. The set of equations which are satisfied by every model M is called E^*. The set of all equations which hold in all the models of E is called the closure of E and written E^{**}. A *theory* is a presentation where E is closed.

Equations for which the right hand side is simpler than the left hand side are understood as *reduction rules*. Not all equations can be viewed as reduction rules. An equation defining *commutativity* for example, cannot. These kinds of equations are collected in a set called 'structural axioms' A (perhaps with some others). A *Membership Rewriting Logic* is a tuple $\langle \Sigma, E \cup A \rangle$ with Σ a signature, E a set of reducible equations and membership functions and A a set of structural axioms. Suppose that the *Membership Rewriting Logic* conforms to the *Church-Rosser* assumption and the reductions always *terminate*. Under these assumptions the final values of the reductions called *canonical forms* of all the expressions form an algebra called the *canonical term algebra* $Can_{T/E \cup A}$. By definition, the canonical term algebra modulo the structural axioms A coincides with the initial algebra $T_{T/E \cup A}$

$$Can_{T/E \cup A} \cong T_{T/E \cup A}.$$

For this case operational- and mathematical semantics are congruent. $Can_{T/E \cup A}$ is a logic of provable equality modulo A and each term $t \in E$ belongs to a equivalence class $[t]_A$ *modulo* the structural axioms A. The reduction process is called *equational simplification* or *rewriting* and gives rise to the binary relations $\rightarrow_{E/A}$ and $\overset{\star}{\rightarrow}_{E/A}$ over A-equivalence classes in $T_{T/E \cup A}$. The relation $\overset{\star}{\rightarrow}_{E/A}$ is the *reflexive* and *transitive closure* of $\rightarrow_{E/A}$. $\rightarrow_{E/A}$ is defined through the equations: if $[t]_A \rightarrow_{E/A} [t']_A$ then there is an equation $t = t' \in E$.
We say that $\rightarrow_{E/A}$ is *terminating* if there is no infinite sequence

$$[t_0]_A \rightarrow_{E/A} [t_1]_A \rightarrow_{E/A} \cdots [t_k]_A \rightarrow_{E/A} [t_{k+1}]_A \rightarrow_{E/A} \cdots .$$

$\rightarrow_{E/A}$ is called *confluent* if whenever $[t]_A \overset{\star}{\rightarrow}_{E/A} [t']_A$ and $[t]_A \overset{\star}{\rightarrow}_{E/A} [t'']_A$ then there is $[u]_A$ such that $[t']_A \overset{\star}{\rightarrow}_{E/A} [u]_A$ and $[t'']_A \overset{\star}{\rightarrow}_{E/A} [u]_A$. An equivalent class $[t]_A$ is called an E/A-canonical form if there is no $[t']_A$ such that $[t]_A \rightarrow_{E/A} [t']_A$.

If a theory $T = \langle \Sigma, E \cup A \rangle$ is such that the equations E are terminating and confluent modulo A, then each equivalent class $[t]_A$ can be reduced to a unique canonical form $can_{E/A}[t]$. Thus reduction gives an effective decision procedure for equational reasoning in T, namely

$$T \vdash t = t'' \text{ iff } can_{E/A}[t] = can_{E/A}[t'].$$

CHAPTER 4

The Derivation of the Initial Algebra

A complete catalogue of a presentation of Domain Theory is given in appendix I. Domain Theory is based on definitions and theorems of partial order 6.3.1, the definitions and theorems of the the meet-operator 6.3.8 and 6.3.7 and the definition of neighborhood system 6.4.1 and Domain elements 6.4.2. We assume finitary Domains so the Domain elements are all principal filters 6.4.4. Putting everything together we get table 1.

Table 1. Collection of Axioms for Domain Theory

(1)	$X \leq X = X$	Reflexivity
(2)	if $X \leq Y$ and $Y \leq Z$ then $X \leq Z$	Transitivity
(3)	if $X \leq Y$ and $Y \leq X$ then $X = Y$	Antisymmetry
(4)	$X \wedge X = X$	Idempotence
(5)	$X \wedge Y = Y \wedge X$	Commutativity
(6)	$((X \wedge Y) \wedge Z) = (X \wedge (Y \wedge Z))$	Associativity
(7)	$\Delta \in D$	NBS (i)
(8)	If $X \in D$ then $X \subseteq \Delta$	NBS (Text)
(9)	if $X, Y, Z \in D$ and $Z \subseteq X \cap Y$ then $X \cap Y \in D$	NBS (ii)
(10)	$\uparrow X := \{Y \in D \mid X \subseteq Y\}$	Domain elements

The objective is to turn the definitions cum theorems into equations for a theory

$$T_D = \langle \Sigma_D, E_D \cup A_D \rangle \tag{1}$$

such, that the equations E_D are terminating and confluent modulo A_D. In such a theory each equivalent class $[t]_A$ can be reduced to a unique canonical form $can_{E/A}[t]$. If a theory has a canonical form, operators can be classified into those which are ‘constructors’ and those which can be defined via the constructors. The latter are called *defined functions.* They ‘disappear’ after equational simplification. For an example let us look at the *Peano* axioms. If the constant operation “0” and the successor function “S(Nat)” are defined as constructors, then the “+” operator, which is defined via the equations

$$0 + N = N \text{ and}$$
$$S(M) + N = S(N + M)$$

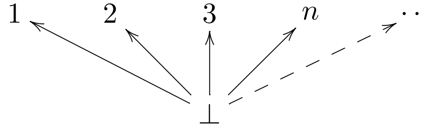
turns into a defined function. An expression with the “+”-operator, e.g. E.g. “0 + S(0)” ‘disappears’ after simplification (evaluation). Here the expression reduces to “S(0)”.

Intuitively a constructor ‘creates’ or ‘makes’ a data-object which stands for a*ground-term.* On the other hand, “+” in an expression like “0 + S(0)” indicates a *pending operation.* Because constructors by definition construct *only* ground terms, expressions which consist only of constructors cannot be simplified. They are in *canonical-* or

normal form. The initial objective can now be formulated more precisely: the goal is the definition of sorts, operations and defined functions via equations in such a way, that the set of equations provides the 'rules' by which any expressions involving the sorts and operations can be reduced to the constructors of an algebra.

4.1. The Zero-Domain

Instead of one symbol '$\perp$', we introduce a *whole undefined Domain*, where all elements $\perp_n$ indicate undefinedness. Here in an example.



The order of Zero-Domains is defined by

$$X \leq Y \equiv (X = \perp) \text{ or } (X = Y). \tag{2}$$

They are characterized by the fact, that if $X \leq Y \leq Z$ then $X = Y$ or $Y = Z$. We think of a Zero-Domain as introducing a set of objects for the later construction of data-objects. Relative to each other these 'data-atoms' are not discrete and disjunct but merrily *distinct*, $X \neq Y$. Domains are partial orders for which antisymmetry 1(3) must hold.

Definition 4.1.1. A Zero-Domain $Zero_D$ a Domain according to def. 6.4.2, in which all neighborhoods $X, Y \in D$ are defined as distinct: $X \leq Y \equiv (X = \Delta)$ or $(X = Y)$. The elements of the Zero-Domain are called distinctions.

We designate distinctions by expressions for natural numbers in a different font $\mathsf{0}, \mathsf{1}, \mathsf{2}, \ldots$. Δ is represented by $\mathsf{0}$. Distinctions are distinct and *atomic*. They must conform to the antisymmetry axiom in table 1(3). That is, if X, Y are distinctions then

$$\text{if } X \neq Y \text{ then not } (X \leq Y \text{ and } Y \leq X).$$

Initial algebras are characterized through the assumption that terms are distinct, until they are proven equivalent. An equivalent class $[t]_A$ is called an E/A-canonical form if there is no $[t']_A$ such that $[t]_A \rightarrow_{E/A} [t']_A$. All distinctions, and not just $\mathsf{0}$, are our intended ground terms. In order to accommodate them we must redefine the equivalence class of a term. We extend the canonical form, so that we may stop reduction if an element of the Zero-Domain is reached.

Definition 4.1.2. Let $Zero_D$ be a Zero Domain. An equivalent class $[t]_{A_D}$ is called an E_D/A_D zero-canonical form, if there is a $[t']_{A_D}$ such that $[t]_{A_D} \rightarrow_{E_D/A_D} [t']_{A_D}$ and $[t']_{A_D} \in Zero_D$.

4.2. Sorts and Terms

We augment commutativity(5) and associativity(6) with idempotence(4) to obtain the set of *structural axioms* A_D. We assign all objects which conform to these axioms the sort Term. Objects of this sort are our distinctions. In addition, there will be *compound terms* like 1.2 or 1.2.3.5, etc. "." here stands for the meet $\wedge$ and intersection $\cap$. There are no brackets. Intuitively the meaning of 1.2.3.5 is $\{1, 2, 3, 4, 5\}$

The other sort we will introduce is Order. Typical expressions of this sort are $0 \geq 1$ or $10 \geq 3$. Order *relations* $X \leq Y$ (respective $Y \geq X$) and $X \subseteq Y$ (respective $Y \supseteq X$) will belong to this sort. In order to accommodate the filter operator of axiom (10) we also need *sets*. We will assume that Set and their associated standard operators and predicates are predefined and available.

4.3. Virtual- and Actual- Membership

In an algebra, *membership-predicates* determine if a term is a member of a specific sort or not. In a many-sorted initial algebra, each term belongs to an equivalent class with a unique representative of a certain sort. In our case this causes problems. Question: is "$0 \geq 0$" a member of Order or not? The answer must be 'yes', otherwise the axiom of reflexivity (1) would not hold. But according to axiom (7), 0 (which stands for Δ) is also a member of Term. A problem arises, because according to the tenets of algebraic specification, each canonical term may only appear *exactly once*. Thus the answer to the question must be 'yes' and 'no'. How does this situation arise? By partitioning Domain Theory into the very sorts Term and Order *one and the same* value is distributed over *two* sorts. To handle this case of *cross-sort identity* we introduce the notions of *virtual-* and *actual-* membership. Actual membership predicates returns membership for a particular sort only. Thus the predicate

```
function IsActualMemberOfOrder(X,Y:Term):Boolean
```

on the argument

```
IsActualMemberOfOrder(0,0)
```

returns *false*.

But there is also a predicate

```
IsMemberOfOrder(X,Y:Term):Boolean
```

which returns the *virtual*-membership. It is defined recursively by

```
  If X=Y then Return ActualMemberOfTerm(X)
  else Return ActualMemberOfOrder(X,Y)
```

This Membership predicate returns *true* on argument (0,0), because 0 is an actual member of Term.

Virtual membership as it is introduced here, is an important concept for Domain Theory. Now the theorem 6.3.6

$$X \leq Y \text{ always implies } inf(X, Y) = X.$$

can be read as a *double implication* regulating *cross-sort identity* between Order and Term. Thus not only `IsMemberOfOrder` must be defined recursively, but also `IsMemberOfTerm` must check among the actual list of Order for particular meets. Thus Term and Order are sorts which are *recursively codefined*. The full definition of `IsMemberOfOrder` and `IsMemberOfTerm` will be provided later.

4.4. Constructors and Defined Operators

The sort Order keeps a record of the order relationship $\geq$ between elements $\mathsf{t_n}, \mathsf{t_m} \in \mathsf{Term}$. They correspond to the subset/superset relations between elements of D. Δ is represented in our algebra by $\mathsf{0}$, a *constant* and *constructor* of sort Term. Constructors must be strictly unique and we introduce $\geq$ as operator and constructor which the signature

$$\geq : \mathsf{Term\ Term} \rightarrow \mathsf{Order}.$$

With the help of these constructors the axiom of transitivity can be introduced as a *defined function*.

$$\mathsf{t_0} \geq \mathsf{t_1} \circ \mathsf{t_1} \geq \mathsf{t_2} = \mathsf{t_0} \geq \mathsf{t_2}.$$

We can also define Filter as a function with a Term as argument and a set (of terms) as return value.

We now come to the most difficult part. The role of the meet-operator in conjunction with axiom (9). First we note that (9) is equivalent to the condition

$$\text{whenever } X, Y, Z \in D \text{ and } Z \subseteq X, Z \subseteq Y \text{ then } X \cap Y \in D.$$

If we restrict our Domains to trees, in (9) with $X = Y$, the meet required by (9) 'virtually' exists because of theorem 6.3.6. We can define a virtual membership predicate which checks for the existence of the meet recursively.

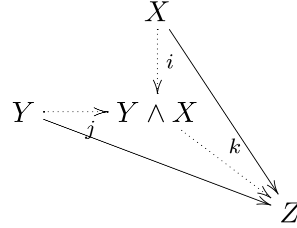
```
function IsMemberOfTerm(X,Y:Term):Boolean
   Return IsMemberOfOrder(X,Y) or IsMemberOfOrder(Y,X);
```

But for the case $X \ngeq Y$ and $Y \ngeq X$ this membership predicate is still undefined. Scott argues, that if Z is approximated by X as well as Y, then it is intuitively necessary that Z is approximated *by the combined information*, $X \wedge Y$. [1] As X and Y are given, we know the information content of the term $X \wedge Y$: it is *exactly the combined information of their components*, here X and Y.[2] To express this information, we construct the already mentioned *compound terms*. Compound-terms are different from simple terms: *they are secondary objects derived from and completely reducible to simple terms*. They

[1] [**Sco81**] p.149.

[2] 'The intersection is exactly what is common to all the given elements. The union on the other hand, is just what the (increasing sequence of the) x_n approximates; the union combines contributions from all the x_n into a "better" element – but no more than that.' [**Sco73**] p.158

contain the *exact amount* of information of all the simple terms they are made up of. Compound-terms are recursive defined objects which are *identified* through their associated filter.



We look upon $X \wedge Y$ as a constructor with exactly this meaning. We introduce the meet-operator with signature

$$\wedge : \mathsf{Term\ Term} \rightarrow \mathsf{Term},$$

and now we we can express (9) as a *defined function* and *double constructor*.

$$\mathsf{t}_1 \geq \mathsf{t}_0 \diamond \mathsf{t}_2 \geq \mathsf{t}_0 = \mathsf{t}_1 \wedge \mathsf{t}_2 \geq \mathsf{t}_0$$

It constructs a defined compound-term $\mathsf{t}_1 \wedge \mathsf{t}_2$ in such a way, that i, j are established. Between this new compound term and Z an order k may need to be constructed also.

4.5. The Signature

We can now define the signature and set of equations for neighborhood systems.

Signature	DynamicDomain				
Sorts	Term Order				
Operations					
	$0, 1, \cdots, \mathsf{n}$:		$\rightarrow$	Term	ctor
	$\wedge$:	Term Term	$\rightarrow$	Term	ctor
	$\geq$:	Term Term	$\rightarrow$	Order	ctor
	$\circ, \diamond$:	Order Order	$\rightarrow$	Order	
	Filter:	Term	$\rightarrow$	Set	
Variables					
	$\mathsf{t}_0, \mathsf{t}_1, \mathsf{t}_2$:	Term			
Equations					
	$\mathsf{t}_0 \wedge \mathsf{t}_0 = \mathsf{t}_0$	Idempotence			A_D
	$\mathsf{t}_0 \wedge \mathsf{t}_1 = \mathsf{t}_1 \wedge \mathsf{t}_0$	Commutativity			A_D
	$\mathsf{t}_0 \wedge (\mathsf{t}_1 \wedge \mathsf{t}_2) = (\mathsf{t}_0 \wedge \mathsf{t}_1) \wedge \mathsf{t}_2$	Associativity			A_D
	$\mathsf{t}_0 \wedge 0 = \mathsf{t}_0$	NBS (Text)			
	$\mathsf{t}_0 \geq \mathsf{t}_0 = \mathsf{t}_0$	Reflexivity			A_D
	$\mathsf{t}_0 \geq \mathsf{t}_1 \circ \mathsf{t}_1 \geq \mathsf{t}_2 = \mathsf{t}_0 \geq \mathsf{t}_2$	Transitivity			def
	$\mathsf{t}_1 \geq \mathsf{t}_0 \diamond \mathsf{t}_2 \geq \mathsf{t}_0 = \mathsf{t}_1 \wedge \mathsf{t}_2 \geq \mathsf{t}_0$	NBS (ii)			def
	$\mathsf{Filter}(\mathsf{t}_0) = \{t_n \mid t_n \geq t_0, t_n \in Zero_D\}$	Domain element			def

'Filter' is defined as a bookkeeping structure for the proper mathematical filter.

4.6. The Implementation of the Initial Algebra

The computer language which the implementation is written in, is fictional and halfway between a proper programming language and mathematical notation. Mathematical notation abstracts too much from important implementational issues. And a concrete programming language burdens one with details which are completely unnecessary for the understanding of an algorithm. I assume, the reader is accustomed to the `O.P` notation, which indicates that `P` is a property of object `O`. A `Term` is an object which has the properties `Name` and `Filter`. Both are of type `Set` which is a primitive type of the programming language. An `Order` is an object which has two properties, `X` any `Y` which are of type `Term`. `Terms` and `Orders` are the container classes of all the objects `Term` and `Order` respectively.

We start with the constructors. The Term-constructor has the arguments `AName` and `AFilter` by which the Term is initialized. If the `Name` is a Member of the Zero-Domain (`Zero_D`) then the Filter is initialized by the `Name`. In such a way a term can be constructed from a distinction (a singleton) and later a meet.

```
ctor MakeTerm(AName,AFilter:Set);
begin
  Name:=AName;
  if IsActualMember(Self) then Self.Destroy
  else begin
    if Name in Zero_D then Filter:={0,Name}
      else Filter:=AFilter;
  end;
end;
```

The constructor `MakeOrder` does the main work. It creates an ordered tuple $X \geq Y$ and adds the information of the lesser defined term to that of the better defined term Y. This – now better defined Term Y – may be a part of a compound term, in which case the filter of this compound term must be adjusted also. All this happens recursively. The recursion is the implementation of the ◦- operator. At the end, after the information for all the terms of the zero-domain has been recomputed, new compound terms may have to be created. This is done in the procedure `CloseMeet`.

```
ctor MakeOrder(AX,AY:Term);

  procedure AdjustCompoundTerms(X,Y:Term);
  begin
    for all C in Terms with C not in Zero_D,
      if Y in C.Filter then C.Filter:=Union(C.Filter,X.Filter);
  end;

  begin
    if IsOrderMember(AX,AY) then exit;
    X:=AX;Y:=AY;
```

```
    Y.Filter:=Union(Y.Filter,X.Filter);
    AdjustCompoundTerms(X,Y);
    for all X',Y' in Terms,
      if  IsOrderMember(X',X) and IsOrderMember(Y,Y')
       then MakeOrder(X',Y');
    end;
    CloseMeet;
  end;
```

In `CloseMeet` the $\diamond$- operator and double constructor is implemented.

```
procedure CloseMeet;
var
  TermCount,
  OrderCount: integer;
  Begin
    Repeat
      TermCount:=Card(Terms);
      OrderCount:=Card(Orders);
      For all X,Y,Z in Terms,
        if IsOrderMember(X,Z) and IsOrderMember(Y,Z)
         then MakeOrder(GetMeet(X,Y),Z));
    Until TermCount=Card(Terms) and OrderCount=Card(Order);
  End;
```

We now define the various membership functions. To each `IsMember` function there is a corresponding `Fetch` function which returns, instead of a boolean, the corresponding sort member – if it exists. As they are structured exactly like the corresponding `IsMember` function they are not presented here.

```
function IsActualMember(X:Term):Boolean;
Begin
  If T in Terms with T.Name=X.Name then return true
  else return False;
End;
```

```
function IsMeetActualMember(X,Y:Term):Boolean;
var
  U:Filter;
Begin
 U:=Union(X.Filter,Y.Filter);
 If T in Terms with T.Filter=U Then return True else return False;
end;
```

```
function IsMeetMember(X,Y:Term):Boolean;
Begin
  If IsOrderMember(X,Y) then return true
```

```
  else if IsOrderMember(Y,X) then return true
  else return IsMeetActualMember(X,Y);
End
```

```
function FetchMeet(X,Y:Term):Term;
var
  U:Set;
begin
  If IsOrderMember(X,Y) then return Y
  else  if IsOrderMember(S,P) then return X
  else  begin
    U:=Union(X.Filter,Y.Filter);
    for all T in Terms
      if T.Filter=U then return T;
    end;
  end;
end;
```

`GetMeet` is needed for the $\diamond$-operator. It returns the meet of two terms of present in `Terms` or `Orders` otherwise it creates a new compound term.

```
function GetMeet(X,Y:Term):Term;
begin
  If IsMeetMember(X,Y) then FetchMeet(X,Y)
  else return MakeTerm(Union(X.Name,Y.Name),Union(X.Filter,Y.Filter));
end;
```

```
function IsActualOrderMember(X,Y:Term):Boolean;
Begin
  If O in Orders with O.X.Name=X.Name and O.Y.Name=Y.Name
  then return true else return false;
End
```

The Reflexive order $X \leq X$ (1) is only a virtual member of `Orders`. And `0` is the least defined term for all others. This least defined relation is also implemented virtually only (2). Otherwise if X,Y are distinctions, then they must be in `Orders` (3). If one of the terms is a compound term (4) then the criteria is the subset relation whereby the direction is reversed because the compound term is constructed as the union of its components (4).

```
function IsOrderMember(X,Y:Term):Boolean;
Begin
 (1) If X.Name=Y.Name then return IsTermMember(X.Name)
 (2) else if X.Name={0} then return true
 (3) else if X in Zero_D and Y in Zero_D then return IsActualMember(X,Y)
 (4) else return IsSubset(X.Filter, Y.Filter);
End
```

These are all the functions needed. We have to check, if all the equations of table 4.5 are represented and implemented. Terms are implemented as sets and the idempotence, associativity and commutativity conditions are met. By `IsOrderMember` (1) the order is reflexive. By `IsOrderMember` (2) `0` is the least defined term. Transitivity is covered by `MakeOrder` as is NBS (ii). The Filter-function which generates a set is a basic structure for `IsOrderMember`. With this function the mathematical filter can be extracted. Antisymmetry: the Zero-Domain is antisymmetric. As long as a symmetry is not introduced by a set of premisses directly or indirectly, antisymmetry holds.

Now we prove that the equations of table 4.5 always terminate.

Theorem 4.6.1. *The equations E_D in T_D are terminating modulo A_D.*

PROOF. There are two sets of ground-terms, simple and compound ones. And there are two defined operations Transitivity and NBS(ii). For these termination must be shown. First we note that for the case NBS(ii) for $X \geq Z$ and $Y \geq Z$ with $X = Y$ then the transitivity reduction satisfies $X \wedge Y$. In this case the conditions of `CloseMeet` in `MakeOrder` are never met. `MakeOrder` calculates the transitive closure which, if the number of simple ground terms is finite, is always finite also. In this case `MakeOrder` always terminates (1).
In the case $X \neq Y$ the conditions in `CloseMeet` hold and new compound terms are generated. The information of each new compound term is derived from the transitive closure and from other compound terms. The compound terms are thus created in increasing cardinality. If the number of simple ground terms is finite, so is this increasing chain of generated compound terms. Thus `CloseMeet` always terminates (2). With (1) and (2) the reduction always terminates. □

Theorem 4.6.2. *The equations E_D in T_D are confluent modulo A_D.*

'Confluence' informally means, that the closure relation computed is independent of the order, in which the computation is performed on the arguments. What is computed are the elements (filters). The elements are defined by the set of all their sub-elements $\uparrow X$ as in remark 6.4.5

$$x = \bigcup \{\uparrow X | X \in x\}.$$

The elements are sets of sets. Therefore, we can proof the theorem 4.6.2 by checking that the construction of elements is implemented correctly.

PROOF. Again we look at the case NBS(ii) for $X \geq Z$ and $Y \geq Z$ with $X = Y$ then the transitivity reduction satisfies $X \wedge Y$. In this case the conditions of `CloseMeet` in `MakeOrder` are never met. The elements in this case are defined through the union operation on `Filter` in the `MakeOrder` constructor only: `Y.Filter:=Union(Y.Filter,X.Filter)`. There the filter is defined recursively for all sub-elements. `AdjustCompoundTerms` is never called and the implementation for case (1) is correct. In the case $X \neq Y$ the conditions in `CloseMeet` the Filter of a compound term is the union of the filter of all its component terms. The compound terms are created in an increasing chain and may contain component terms at a later stage. `AdjustCompoundTerms` guarantees that the Filter of a component term is always the union of its component filters under the addition of new information. Thus (2) component filters are implemented correctly. With (1) and (2) filters are implemented correctly and confluence is proven. □

Summary: The equations E_D in T_D are terminating and confluent modulo A_D. Each equivalent class $[t']_{A_D}$ is in unique zero canonical form $can_{E_D/A_D}[t']$ according to def. 4.1.2 and we have a decision procedure for equational reasoning.

CHAPTER 5

Dynamic Domains

Dynamic Domains are the Domains resulting from the Small Calculus. Instead of neighborhoods we have Terms^* which are dynamically created by constructors over the distinctions $\mathsf{0..n}$ of a Zero-Domain Zero and an initial set of order-constructors Orders. Both are 'closed up' under the equations of the signature of the canonical term algebra 4.5 besides$\mathsf{Terms}^* \supseteq \mathsf{Terms}$ we obtain the set $\mathsf{Orders}^* \supseteq \mathsf{Orders}$.

Definition 5.0.3. (Dynamic Domain) A *Dynamic Domain* D is a triple $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ with Zero a Zero Domain according to def. 4.1.1, Terms, a set of Term-constructors over Zero and $\mathsf{Orders} \subseteq \mathsf{Terms} \times \mathsf{Terms}$ a set of order-constructors of the canonical term algebra 4.5. Let $\mathsf{X}, \mathsf{Y} \in \mathsf{Terms}^*$. Then the (ideal) elements of the Dynamic Domain are those subsets $\mathsf{x} \subseteq \mathsf{Terms}^*$ where:

(a) if $\mathsf{X} \in \mathsf{Terms}^*$ then $\mathsf{X} \in \mathsf{x}$;
(b) if $\mathsf{X} \in \mathsf{x}$ and $\mathsf{X} \leq \mathsf{Y} \in \mathsf{Orders}^*$ then $\mathsf{Y} \in \mathsf{x}$.

The domain of all such term elements is written as $|\mathsf{D}|$.

5.1. Examples

In the following examples, of Dynamic Domains, terms/elements are represented by nodes in a graph. The information contained in the closure $\stackrel{\star}{\to}$ is displayed via the edges; redundant (transitive) edges are not drawn. We shall look upon sequences of Domains. There is a one-one correspondence between elements of the Zero-Domain and elements proper (filters) of the Domain. This allows us to view these distinctions as 'movable' and the associated elements as storing the history of the 'moves'. The moves themselves are caused by the assertion of information-order between distinctions. An example of a sequence of 'moves' is shown on figure 1.

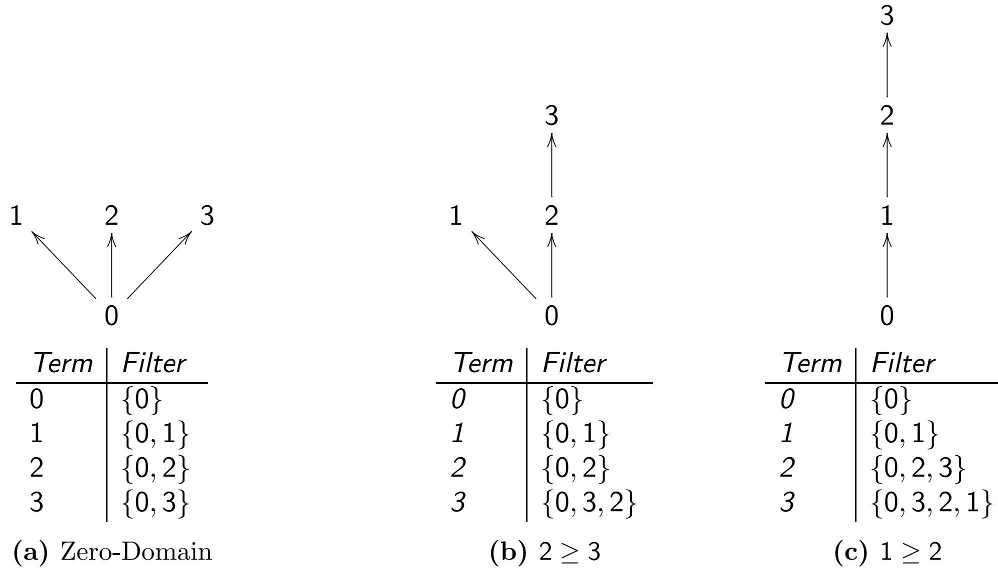
Example 5.1.1. *Let D be a dynamic domain with* $\mathsf{Term} = \{0, 1, 2, 3\}$. *Fig.1(b) results from* $\mathsf{Order} = \{2 \geq 3\}$, *figure 1(c) from* $\mathsf{Order} = \{2 \geq 3, 1 \geq 2\}$. *The closures* $\stackrel{\star}{\to}$ *are represented by their associated graphs.*

Further 'moves' beyond (c) are excluded by the antisymmetry condition 1(3).

Figure fig.2 in conjunction with the figures (1) show the *confluence* of the $\geq$-operation.

These examples of moves can also be interpreted to model the flow of information by Mill's syllogisms in the introduction. ($Socrates/Wellington = 3$, $human = 2$ and $animal = 1$.)

Figure 1. Sequence of Dynamic Domains



Term	*Filter*
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2\}$
3	$\{0,3\}$

(a) Zero-Domain

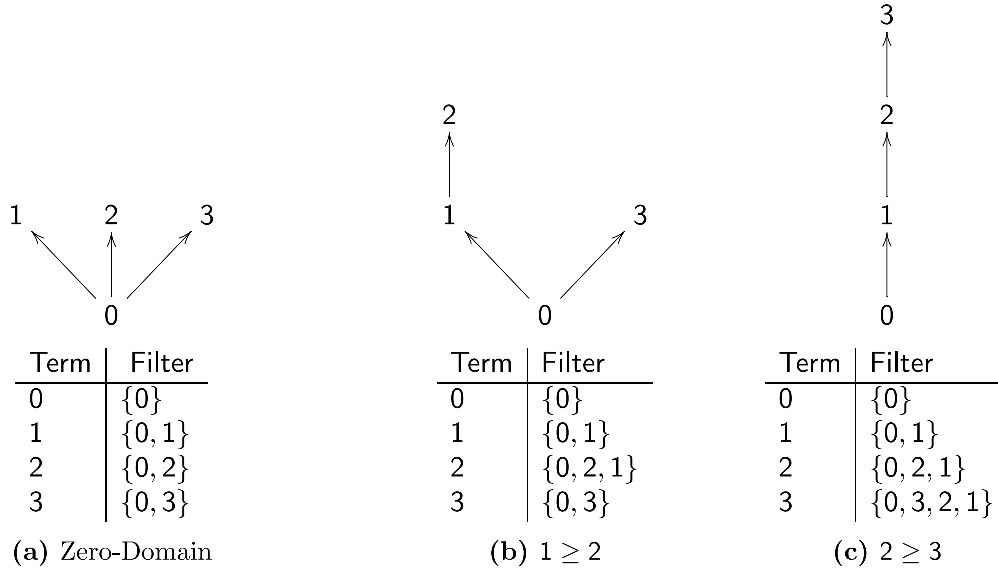
Term	*Filter*
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2\}$
3	$\{0,3,2\}$

(b) $2 \geq 3$

Term	*Filter*
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2,3\}$
3	$\{0,3,2,1\}$

(c) $1 \geq 2$

Figure 2. Confluence of Dynamic Domains



Term	Filter
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2\}$
3	$\{0,3\}$

(a) Zero-Domain

Term	Filter
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2,1\}$
3	$\{0,3\}$

(b) $1 \geq 2$

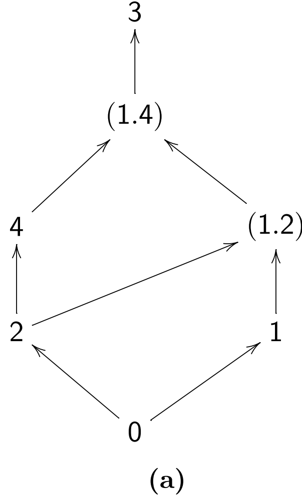
Term	Filter
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2,1\}$
3	$\{0,3,2,1\}$

(c) $2 \geq 3$

A more complex example which leads to compound terms is the following one.

Example 5.1.2. *Let* D *be a dynamic domain with* $\mathsf{Term} = \{0,1,2,3,4\}$ *and* $\mathsf{Order} = \{1 \geq 3, 4 \geq 3, 2 \geq 4\}$. *The closure* $\stackrel{\star}{\rightarrow}$ *is represented by its associated graph fig.3:*

Figure 3. A Dynamic Domain with Compound Terms



(a)

Term	*Filter*
0	$\{0\}$
1	$\{0,1\}$
2	$\{0,2\}$
3	$\{0,3,1,2,4\}$
4	$\{0,4,2\}$
(1.4)	$\{0,1,2,4\}$
(1.2)	$\{0,1,2\}$

(b)

5.2. Elements and their Inverses

Domain elements are filters (see def. 6.4.2). Instead of neighborhoods we have terms and we can query the order relation between terms with `IsMemberOrder`. The order relationship between terms (neighborhoods) and principal filters is reversed

$$X \subseteq Y \text{ iff } \uparrow Y \subseteq \uparrow X.$$

Further more, we know that our distinctions are ground-terms. Because of the inclusion reversal, each distinction X corresponds to an element $\uparrow\mathsf{X}$ with $\mathsf{X} \in \uparrow\mathsf{X}$. $\uparrow\mathsf{X}$ is defined by $\uparrow\mathsf{X} := \{\mathsf{Y} \in \mathsf{Terms}^* | \mathsf{X} \leq \mathsf{Y} \in \mathsf{Orders}^*\}$. Therefore X is the 'seed' from which we can calculate all the *lesser determined* terms belonging to the element. This is the reason behind the element definition (a) and (b). Because of (a) terms and elements are in one-one correspondence in Dynamic Domains.

The filter definition is neutral towards the 'shape' of the filter elements. $'0'$ so far played the role of the empty set or empty sequence. This 'empty-shape' of $'0'$ is not necessary. In Scott's definition of neighborhood systems, all neighborhoods are subsets of a set Δ which functions as $'0'$. The initial algebras being finite, we can also derive filters in that 'opposite' shape. This gives us the next definition.

Definition 5.2.1. (Inverse Elements) The *inverse elements* of a Dynamic Domain $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ are those subsets $[\mathsf{X}] \subseteq \mathsf{Terms}^*$ where for all $\mathsf{X}, \mathsf{Y} \in \mathsf{Terms}^*$:

$$\text{if } \mathsf{X} \geq \mathsf{Y} \in \mathsf{Orders}^* \text{ then } \mathsf{Y} \in [\mathsf{X}].$$

The domain of all such inverse elements is written as $|\mathsf{D}|^{-1}$.

In [**Sco81**] a $[X]$-notation can be found in many places. What is the relation between the inverse elements $[\mathsf{X}]$ and $[X]$? Scott introduces the notation in his theorem 1.10.

Theorem 5.2.2. *Given any neighborhood system D define for $X \in D$*

$$[X] = \{x \in |D| | X \in x\}$$

The subsets $[X] \subseteq |D|$ for $X \in D$ form a neighborhood system over $|D|$ which determines a domain isomorphic to $|D|$.

PROOF. We note first that (1) $[\Delta] = |D|$.
Next, note that
(2) X, Y are consistent in D iff $[X] \cap [Y] \neq \emptyset$;
and that for $X, Y \in D$
(3) $[X] \cap [Y] = [X \cap Y]$ if $X \cap Y \in D$.
Inasmuch as
(4) $\uparrow X \in [X]$ for all $X \in D]$,
it easily follows that D and the family
$\{[X] | X \in D\}$
are in a one-one, inclusion preserving correspondence. □

The definition of the inverse elements 5.2.1 is modelled on the definition of $[X]$ and theorem 5.2.2. In definition 5.2.1 the elements in 5.2.2 are substituted by the unique tokens. Now we can state the relationship between the $[X]$ of theorem 1.10 and the inverse elements of $|\mathsf{D}|^{-1}$

Theorem 5.2.3. *Let $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ be a Dynamic Domain. Then the domain with elements $[X]$ derived according to theorem 5.2.2 is isomorphic to the inverse domain $|\mathsf{D}|^{-1}$ of def.5.2.1 .*

PROOF. To prove the isomorphism of two domains we must show that (1) there is a bijection between the elements and (2) the inclusion relationship between elements is maintained (see definition of domain isomorphism 6.4.11).
(1) In theorem 1.10 there is one-one correspondence between neighborhoods $X \in D$ and elements $x \in |D|$. Neighborhoods correspond to $\mathsf{X} \in \mathsf{Terms}^*$. According to def.5.2.1(a) there is a one-one correspondence between these terms and the inverse elements $[\mathsf{X}]$. Therefore there is a one-one correspondence between elements of theorem 1.10 $[X]$ and inverse elements $[\mathsf{X}]$.
(2) $[X]$ elements are defined in terms of elements x which are defined in terms of neighborhoods X. Each change of level leads to an inclusion reversal. $X \subseteq Y$ iff $y \subset x$ iff $[X] \subseteq [Y]$. The two inclusions reversals cancel each other out and $X \subseteq Y$ iff $[X] \subseteq [Y]$. With inverse elements $[\mathsf{X}]$ we start from inverted elements 5.2.1(b) and $\mathsf{X} \leq \mathsf{Y}$ iff $[\mathsf{X}] \subseteq [\mathsf{Y}]$ and $[\mathsf{X}], [\mathsf{Y}] \neq [0]$. With $[\mathsf{X}] \subseteq [\mathsf{Y}]$ and for $[\mathsf{X}], [\mathsf{Y}] \neq [0]$ iff $[\mathsf{X}] \subseteq [\mathsf{Y}]$ the isomorphism of the two domains is proven. □

The domain of inverse elements is isomorphic to the Domain of $[X]$. This Domain is, in turn, isomorphic to the initial domain if the inclusion relation between inverse elements is reversed. We put this down in a remark.

Remark 5.2.4. Given any Dynamic Domain $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$, the domain of elements $|\mathsf{D}|$ and the domain of inverse elements $|\mathsf{D}|^{-1}$ are isomorphic to each other.

Any Dynamic Domain $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ generates *two* sets of elements. What distinguishes them is their inclusion relationship. Also, the inclusion-relationship between terms (neighborhoods) and elements is reversed. When Domains are graphically

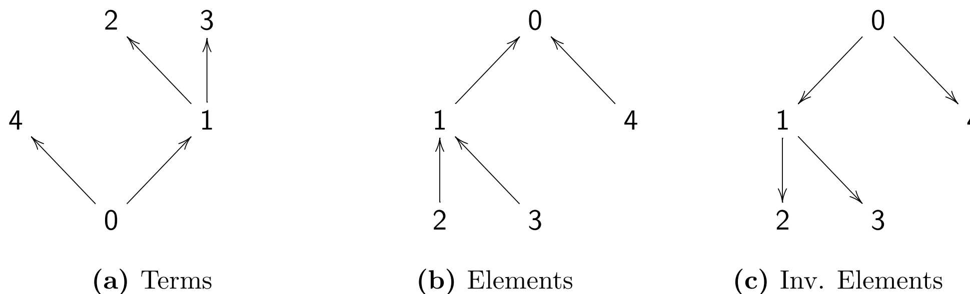
represented in literature, the inclusion relation is usually represented by an upward arrow from the least element, bottom. We have represented term-graphs in this way. To take account of the element-relations, we will flip the term-graph *downward* at 0, and keep the upward-direction of the term-graph representation. This gives a graph for the elements. The least informative element is now at the *top*. The further *down* an element is in the graph, the *more information* it contains and the *more determined* it is. To display the inverse elements, [X], we only need to reverse the direction of the arrows in the graph for the elements. Thus we can identify the elements with the *position of a node* in the a graph. The information an element contains can be learned from following the arrows *upwards*. The elements which are contained in an inverse element can be learned by following the arrows *downward*. This is shown in the example 5.2.5.

Example 5.2.5. *Let* $D = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ *be a Dynamic Domain with* $\mathsf{Terms} = \{0, \ldots, 4\}$*,* $\mathsf{Order} = \{2 \geq 1, 3 \geq 1\}$ *and the accompanying Domain* Zero*. The correspondence between terms and elements is displayed in the table and the figures below.*

Table 1. Terms and Elements

Terms X	*Elements x*	*Inv. Elem. [X]*
0	{0}	{0, 1, 2, 3, 4}
1	{0, 1}	{1, 2, 3}
2	{0, 1, 2}	{2}
3	{0, 1, 3}	{3}
4	{0, 4}	{4}

Figure 4. Isomorphic Graphs



(a) Terms **(b)** Elements **(c)** Inv. Elements

If we represent the elements by nodes in a graph and do not indicate the direction, then one graph can represent the elements as well as the inverse elements. Elements and inverse elements are in a one to one correspondence with the Terms^*. Thus the elements of Dynamic Domains are uniquely characterized by 'their' term! And the converse is true for each kind of element and versus element - inverse element. To take account of the various correspondences we introduce some new notation.

Notation 5.2.6 (Characteristic term, unique element)**.** *Let* $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ *be a Dynamic Domain. Then* $\mathsf{X} \in \mathsf{Terms}^*$*,* $\mathsf{x} \in |\mathsf{D}|$*, and* $[\mathsf{X}] \in |D|^{-1}$ *are in one-one*

correspondence and we write:

$\mathsf{X.x}$	*:* X *is the* <u>*characteristic term*</u> *of the element* x*,*
$\mathsf{X.[X]}$	*:* X *is the* <u>*characteristic term*</u> *of the inverse element* $\mathsf{[X]}$;
$\mathsf{x.X}$	*:* x *is the* <u>*unique element*</u> *determined by the term* X*;*
$\mathsf{[X].X}$	*:* $\mathsf{[X]}$ *is the* <u>*unique inverse element*</u> *determined by the term* X*;*
$\mathsf{x.[X]}$	*:* x *is* <u>*the element*</u> *with the inverse element* $\mathsf{[X]}$
$\mathsf{[X].x}$	*: the* <u>*inverse element*</u> *of the element* x*.*

After having defined Dynamic Domains, we can define mappings between Dynamic Domains in accordance with definition 6.4.7.

Definition 5.2.7. (<u>Approximable mapping</u>) Let $\mathsf{D_0}, \mathsf{D_1}$ be two Dynamic Domains. An approximable mapping $\mathsf{f} : \mathsf{D_0} \to \mathsf{D_1}$ between Dynamic Domains is a binary relation between terms $\mathsf{X} \in \mathsf{Terms}_0^*$ and terms $\mathsf{Y} \in \mathsf{Terms}_1^*$ such that

(i) $\mathsf{0_0 f\ 0_1}$;

(ii) XfY and $\mathsf{XfY'}$ always implies $\mathsf{Xf(Y \wedge Y')}$ and

(iii) $\mathsf{XfY, X \leq X \in Orders_0^*}$, and $\mathsf{Y \leq Y' \in Orders_1^*}$ always implies $\mathsf{X'fY'}$.

Finally we define the identity mapping and composition between Dynamic Domains $\mathsf{D_0}, \mathsf{D_1}$ in accordance with theorem 6.4.9.

Definition 5.2.8 (Identity Mapping, Composition)**.** The identity mapping $1_D : D \to D$ on a Dynamic Domain $D = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ relates $\mathsf{X}, \mathsf{Y} \in \mathsf{Terms}^*$

(i) $\mathsf{X1_DY}$ iff $\mathsf{X \leq Y \in Orders^*}$.

If $\mathsf{f} : \mathsf{D_0} \to \mathsf{D_1}$ and $\mathsf{g} : \mathsf{D_1} \to \mathsf{D_2}$ are given, then the composition $\mathsf{g \circ f} : \mathsf{D_0} \to \mathsf{D_2}$ relates $\mathsf{X} \in \mathsf{Terms}_0^*$ and $Z \in \mathsf{Terms}_2^*$ as follows:

(ii) $\mathsf{Xg \circ fZ}$ iff $\exists \mathsf{Y} \in \mathsf{Terms}_1^*.\ \mathsf{XfY}$ and YgZ.

5.3. Deformation in Dynamic Domains

Homeomorphic transformations are intuitively understood as elastic deformations of geometric objects in topological spaces. A homeomorphism is defined as follows:

Definition 5.3.1 (Homeomorphism)**.** Two topological spaces X and Y are called <u>homeomorphic</u> or <u>topologically equivalent</u> if there exists a one-one and onto- (surjective) function $h : X \to Y$ such that h and h^{-1} are continuous. The function h is called a <u>homeomorphism</u>.

We now combine the 'geometric' view of properties with the qualitative notion of information. That is, we view Dynamic Domains as systems which stay identical in the deformation inflicted by information- increase/decrease. Domains are identical if they are isomorphic; all isomorphism in domains comes from approximate mappings. Therefore, if two Dynamic Domains D_0 and D_1 are to be considered the same, there must be approximable mappings $h : D_0 \to D_1$ and $h^{-1} : D_1 \to D_0$.

$$h^{-1} \circ h = 1_{D_0} \text{ and}$$
$$h \circ h^{-1} = 1_{D_1}.$$

To these mappings there are corresponding functions h and h^{-1} which, as functions, must be surjective or 'onto'. Therefore *at least* the distinctions of the Zero-Domain must be the same in both domains. But in some Dynamic Domains there will be additional compound terms. We remember: compound terms are *reducible* to distinctions. So we can define the notion of *more/less* information *purely* in terms of *inclusion* of D_0 and D_1. To this purpose we define the notion of *subsystem*:

Definition 5.3.2 (Subsystem)**.** A Dynamic Domain $\mathsf{D}_0 = \langle \mathsf{Terms}_0, \mathsf{Orders}_0, \mathsf{Zero}_0 \rangle$ is a *subsystem* of $\mathsf{D}_1 = \langle \mathsf{Terms}_1, \mathsf{Orders}_1, \mathsf{Zero}_1 \rangle$ written

$$\mathsf{D}_0 \lhd \mathsf{D}_1$$

iff $\mathsf{Zero}_0 = \mathsf{Zero}_1$, $\mathsf{Terms}_0^* \subseteq \mathsf{Terms}_1^*$ and $\mathsf{Orders}_0^* \subseteq \mathsf{Orders}_1^*$.

Remark 5.3.3. If $\mathsf{D}_0 \lhd \mathsf{D}_1$ then $\mathsf{Zero} = \mathsf{Zero}_0 = \mathsf{Zero}_1$ is the smallest Domain included in both Dynamic Domains.

This definition of subsystem is the same as the definition of subdomain in def. 6.10 by Scott *except* that it is *not* required that D_0 and D_1 *share the same consistency.*[1]

We now can set out to define h for Dynamic Domains. It is based on the following idea. If $\mathsf{D}_0 \lhd \mathsf{D}_1$ then to each $\mathsf{X} \in \mathsf{Terms}_0^*$ there uniquely corresponds a term $\mathsf{Y}' \in \mathsf{Terms}_1^*$. Y' in turn corresponds to a unique element $\mathsf{y}'.\mathsf{Y}' \in |\mathsf{D}_1|$ We define h as the approximate mapping between a X and all $\mathsf{Y} \in \mathsf{y}'.\mathsf{Y}'$.

To define h^{-1} we utilize the inverse Domain $|\mathsf{D}_0|^{-1}$ and do h in reverse. $\mathsf{Terms}_1^* \supseteq \mathsf{Terms}_0^*$, but to each term $\mathsf{Y} \in \mathsf{Terms}_0^*$ there is a uniquely corresponding term $\mathsf{X}' \in \mathsf{Terms}_1^*$. X' in turn corresponds to a unique inverse element $[\mathsf{X}]'.\mathsf{X}' \in |\mathsf{D}_1|^{-1}$. We define h^{-1} as the approximate mapping between all $\mathsf{X} \in [\mathsf{X}]'.\mathsf{X}'$ and a Y.

Theorem 5.3.4. *If two Dynamic Domains* $\mathsf{D}_0 \lhd \mathsf{D}_1$ *are given, then there are two approximable mappings* $\mathsf{h} : \mathsf{D}_0 \to \mathsf{D}_1$ *and* $\mathsf{h}^{-1} : \mathsf{D}_0 \to \mathsf{D}_1$ *defined by*

$$\mathsf{h} = \{\mathsf{X} \to \mathsf{Y} | \mathsf{Y} \in \mathsf{y} \textit{ and } \mathsf{y}.\mathsf{Y}' \in \mathsf{Terms}_1^*\}$$

$\forall \mathsf{X} \in \mathsf{Terms}_0^*, \forall \mathsf{Y} \in \mathsf{Terms}_1^*, \exists \mathsf{Y}' \in \mathsf{Terms}_1^*$.

$$\mathsf{h}^{-1} = \{\mathsf{X} \to \mathsf{Y} | \mathsf{X} \in [\mathsf{X}]' \textit{ and } [\mathsf{X}]'.\mathsf{X}' \in \mathsf{Terms}_0^*\}$$

$\forall \mathsf{X} \in \mathsf{Terms}_1^*, \forall \mathsf{Y} \in \mathsf{Terms}_0^*, \exists \mathsf{X}' \in \mathsf{Terms}_1^*$.

Proof. We have to check that h is an approximable mapping according to def. 6.4.7(i)-(iii).
(i) is satisfied because $\mathsf{0} \in Zero$.
(ii): If $\mathsf{X} \to \mathsf{Y}$ and $\mathsf{X} \to \mathsf{Z}$ then $\mathsf{X} \to \mathsf{Y} \wedge \mathsf{Z}$ because $\mathsf{Y} \wedge \mathsf{Z} \in \mathsf{y}.\mathsf{Y}'$ (filter condition).
(iii) is satisfied because h is defined over all $\forall \mathsf{X} \in \mathsf{Terms}_0^*, \forall \mathsf{Y} \in \mathsf{Terms}_1^*$.
With (i)-(iii) h is an approximable mapping.

[1] In [**Sco81**] p.247. A neighborhood systems D_0 is subdomain of D_1 iff they are defined over the same set of tokens, $D_0 \subseteq D_1$ and if $X, Y \in D_0$ and $X \cap Y \in D_1$ then $X \cap Y \in D_0$

Proof that h^{-1} is an approximable mapping: dual to proof of h.

□

It is now time for an example.

Example 5.3.5. *Let* Zero *be a Zero-Domain with* $\mathsf{Terms} = \{0, \ldots, 4\}$*, and* $\mathsf{D}_0 \lhd \mathsf{D}_1$ *two Dynamic Domains over* Zero *in a subsystem relation with* $\mathsf{Orders}_0 = \{1 \geq 2,\ 1 \geq 3\}$ *and* $\mathsf{Orders}_1 = \{1 \geq 2,\ 1 \geq 3,\ 3 \geq 4\}$*. The correspondence between terms and elements of the three Domains is displayed in the following tables.*

Figure 5. Sequence of Homeomorphic Domains

X	x	[X]
0	{0}	{0, 1, 2, 3, 4}
1	{0, 1}	{1}
2	{0, 2}	{2}
3	{0, 3}	{3}
4	{0, 4}	{4}

(a) Zero

x	[X]
{0}	{0, 1, 2, 3, 4}
{0, 1}	{1, 2, 3}
{0, 1, 2}	{2}
{0, 1, 3}	{3}
{0, 4}	{4}

(b) D_0

x	[X]
{0}	{0, 1, 2, 3, 4}
{0, 1}	{1, 2, 3, 4}
{0, 1, 2}	{2}
{0, 1, 3}	{3, 4}
{0, 1, 3, 4}	{4}

(c) D_1

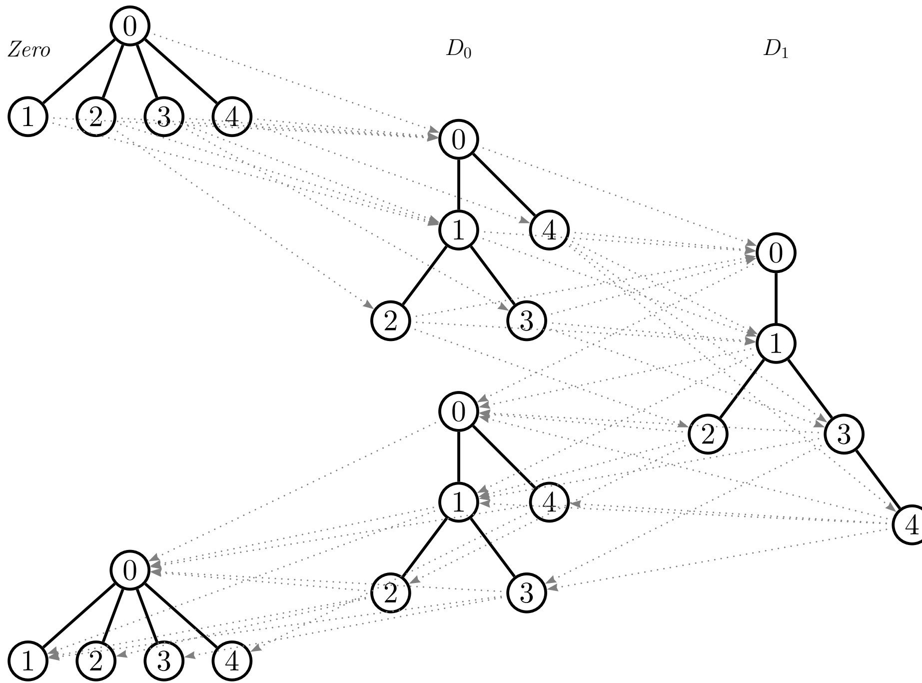


Table 2. First Part of Homeomorphism (h, h')

$\mathsf{h} : \mathsf{Zero} \to \mathsf{Terms}_0^*$	$\mathsf{h}' : \mathsf{Terms}_0^* \to \mathsf{Terms}_1^*$
$\{0 \mapsto 0,$	$\{0 \mapsto 0,$
$1 \mapsto 0, 1 \mapsto 1,$	$1 \mapsto 0, 1 \mapsto 1,$
$2 \mapsto 0, 2 \mapsto 1, 2 \mapsto 2,$	$2 \mapsto 0, 2 \mapsto 1, 2 \mapsto 2,$
$3 \mapsto 0, 3 \mapsto 1, 3 \mapsto 3,$	$3 \mapsto 0, 3 \mapsto 1, 3 \mapsto 3,$
$4 \mapsto 0, 4 \mapsto 4\}$	$4 \mapsto 0, 4 \mapsto 1, 4 \mapsto 3, 4 \mapsto 4\}$

Table 3. Second Part of Homeomorphism (h'^{-1}, h^{-1})

$\mathsf{h}'^{-1} : \mathsf{Terms}_1^* \to \mathsf{Terms}_0^*$	$\mathsf{h}^{-1} : \mathsf{Terms}_0^* \to \mathsf{Zero}$
$\{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0,$	$\{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0,$
$1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1, 4 \mapsto 1,$	$1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1,$
$2 \mapsto 2,$	$2 \mapsto 2,$
$3 \mapsto 3, 4 \mapsto 3,$	$3 \mapsto 3,$
$4 \mapsto 4\}$	$4 \mapsto 4\}$

CHAPTER 6

Logic

In the first part, the Small Calculus will be turned into a formal deductive system. In the second part this system is related to Aristotelian logic.

6.1. The Small Calculus as a Deductive System

A formal system consists of a finite set of tokens. Formation rules describe how complex expressions are constructed from tokens and which expressions are allowed and well-formed. For any finite expression composed of tokens it must be *decidable* if it is well-formed or not. The set of well-formed-formulas (wff's) is called the language L of the formal system. Transformation rules describe how from a set of premisses $P \subseteq L$ assumed as given, other wff's can be inferred. If a set of wff's is given initially or unconditionally, it is a called a set of axioms. The transformation-rules together with the axioms are called the deductive base of a calculus. The set of wff's which can be inferred from a deductive base is called the set of theorems.

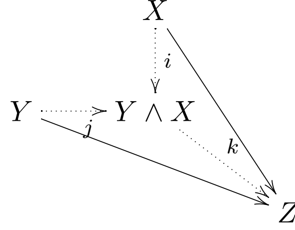
For the Small Calculus, the various pieces of a formal system are easy to 'identify'. The orders defined through the Zero-Domain makes up the set of 'axioms'. The 'transition rules' consist of transitivity and NBS(ii). In Orders are the premisses P and Orders* consists of the set of 'theorems' (together with the axioms and premisses). 'Formation rules' describe how compound terms are built up from distinctions.

The Small Calculus 'deduces' all the finite information contained in a finite set of premisses. In the course of deduction, additional (compound) terms may be introduced due to the recursive nature of the objects. Thus order relations may be deduced between terms which may be formed later and are not there from the beginning! This makes the notion of "well-formed-formula" not as clear-cut as one would wish. In one way we know how well-formed expressions of the Small Calculus must look. But: terms as well as theorems depend on a set of premisses – which is conditional and arbitrary – and, let us now assume, may be extended indefinitely. Therefore, the question whether a certain expression is a wff is *undecidable until* it is deduced by a finite set of premisses. The feedback loop between premisses and terms makes for a stark difference to formal systems. Thus, we know what it means that an expression is well-formed *relative* to a given set of premisses, but this knowledge does not help us to answer the question if an expression is decidable in an *absolute* way, independent of a given set of premisses.

We shall refer to this situation by saying, that in the Small Calculus expressions are created *dynamically.* In formal systems, this production process is split into two: *possible* theorems are all defined *statically* in advance. In such static systems the question as to whether an expression is well-formed can always be decided because it does not depend on the deductive properties of the formal system. In the Small Calculus it does. A static

deneb@logike.info – 2010-09-02 – Revision 8

deductive system can therefore model a dynamic one only *up to a point*. To identify this point, we fix the set of terms in advance and close up under reflexivity, transitivity and meet. Let us look at figure 4.4 again:



A compound term is generated if $X \geq Z, Y \geq Z$ and $X <> Y$. In the static level of the compound-term recursion, there is a one-one correspondence between a Z and $Y \wedge X$. Thus, for the first level of the recursion,

(3) $$Y \wedge X \in Terms^* \text{ iff } X, Y, Z \in Terms \text{ and } X \geq Z \text{ and } Y \geq Z.$$

In the algebra, the meets are represented by terms and collected in the set Terms^*. For the *static closure* we replace Terms^* with the set Meets^* and define $Y \wedge X \in Meets^*$ iff $X \geq Z$ and $Y \geq Z$ with $X, Y, Z \in Terms$.

Definition 6.1.1. Let $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ be a Small Calculus. Then we call $\langle \mathsf{Terms}^*, \mathsf{Orders}^* \rangle$ the dynamic closure of D. The static closure of D is $\langle \mathsf{Meets}^*_{\mathsf{static}}, \mathsf{Orders}^*_{\mathsf{static}} \rangle$ with

$$Orders^*_{static} = \{X \geq Y | X \geq Y \in Order^* \text{ and } X, Y \in Terms\};$$
$$Meets^*_{static} = \{X \wedge Y | X, Y, Z \in Terms \text{ and } X \geq Z, Y \geq Z \in Orders^*_{static}\}.$$

The static closure of a Small Calculus over a finite set of premisses can be computed independently of the dynamic closure. We need:

- the set $T = \{0, 1, 2, \ldots\}$ of terms;
- the set $M = \{A, I\}$ indicating binary relations $\geq$ and $\wedge$;
- the set $E = \{Uxy | U \in M, x, y \in T\}$ of expressions;
- the set $\Sigma = 2^E$.

Σ is the fundamental set of expressions for static closures of a Small Calculus. From the operations of transitivity and NBS(ii) we derive three deduction rules:

(1) $Axy, Ayz \rightarrow Axz$;
(2) $Ixy, Azy \rightarrow Izx$;
(3) $Axy, Ayz \rightarrow Ixz$

These rules are abbreviations for a binary relation $r \in \Sigma \times \Sigma$.

Definition 6.1.2. Let r denote a relation on $\Sigma : r \in \Sigma \times \Sigma$. If for some $\alpha, \beta \in \Sigma$, $(\alpha, \beta) \in r$ we write $\alpha \xrightarrow{r} \beta$.

The first rule is an abbreviation for a relation r_1

$$\alpha = \{\ldots, Axy, \ldots, Ayz, \ldots\} \rightarrow \alpha' = \{\ldots, Axy, Ayz, Axz \ldots\}.$$

An application of a rule must lead to an definite enlargement of α, otherwise the rule is thought not to be applicable. The other two rules (2) and (3) abbreviate relations r_2, r_3 accordingly. The relation r is a binary relation and we define a closure operator

$$\Phi : \Sigma \times \Sigma$$

which is to compute the reflexive transitive closure of r. The closure operator maps the set of premisses α to β, the set of the premisses together with *all* the logical consequences of the three rules.

Definition 6.1.3. Let T be a set of terms with $0 \in T$. the relation be defined as $r_1 \cup r_2 \cup r_3$. The set α is a possible empty set of premisses. $\Phi_r^n(\alpha)$ is called closure of level n if

$$\Phi_r^n(\alpha) = \begin{cases} \{Axx, A0x| \text{ for all } x \in T\} & ifn = 0; \\ \alpha & ifn = 1; \\ \Phi_r^{n-1}(\alpha) \cup \Phi_r^{n-2}(\alpha) & otherwise. \end{cases}$$

The closure is defined by

$$\Phi_r(\alpha) \equiv \Phi_r^*(\alpha) \equiv \bigcup_{n=0}^{\infty} \Phi_r^n(\alpha).$$

Proposition 6.1.4. *Let* $\mathsf{D} = \langle \mathsf{Terms}, \mathsf{Orders}, \mathsf{Zero} \rangle$ *be a Small Calculus and* $\langle \mathsf{Meets}^*_{\mathsf{static}}, \mathsf{Orders}^*_{\mathsf{static}} \rangle$ *its static closure.* $\Phi_r^n(\alpha)$ *computes the static closure under the mapping*

$$\mathsf{Terms} \rightleftarrows T$$
$$X \geq Y \in \mathsf{Orders} \rightleftarrows Axy \in \alpha$$

Proof. We have to prove (1) that there is a $1:1$-correspondence between $Orders^*_{static}$ and $Axy \in \Phi_r^*(\alpha)$. And (2), that $X \wedge Y \in Meets^*_{static}$ and $Ixy \in \Phi_r^*(\alpha)$ are in $1:1$-correspondence.

(1) $Orders^*_{static}$ as well as $\Phi_r^*(\alpha)$ contain the reflexive transitive closure over the same set of premisses and over the same set of terms. But $Orders^*_{static} \subseteq Orders^*$ and there may be chains in $Orders^*$ which generate $X \geq Y \in Orders^*$ with $X, Y \in Terms$ but the corresponding $Axy \notin \Phi_r^*(\alpha)$. Let $X \geq Y$ be such a generated approximation. If it was generated from a chain with compound terms C, then there must have been a chain $X \geq C, C \geq Y$. But for each such chain which generates $X \geq Y$ there is at least one chain where C is replaced by $Z \in Term$, because of the condition compound terms generation. Therefore the reflexive and transitive closure $Orders^*_{static}$ and $\Phi_r^*(\alpha)$ are equivalent.

(2) $Meets^*_{static} = \{X \wedge Y | X, Y, Z \in Terms \text{ and } X \geq Z, Y \geq Z \in Orders^*_{static}\}$. According to (1) $Orders^*_{static}$ and $\Phi_r^*(\alpha)$ are equivalent. We only have to show that for each $Axy \in \Phi^*$, $Ixy, Iyx \in \Phi^*$. Let Axy be such an element. By def., $Axx, Ayy \in \Phi_r^0$. From rule (3) $Axx, Axy \rightarrow Ixy \in \Phi^*$. From rule (2) $Ixy, Axy \rightarrow Iyx \in \Phi^*$. As Axy was chosen arbitrarily, for each $Axy \in \Phi^*$ there is a tuple $Ixy, Iyx \in \Phi^*$. Therefore $Meets^* \subseteq \Phi^*$ and $Meets^*$ is closed. With (1) and (2) the proposition is proven. □

The three rules by which the *static closure* of a Small Calculus can be computed belong to one of the oldest known formal logics: *Aristotelian syllogistic.*

Table 1. A Deductive Base for Aristotelian Logic

Barbara	Axy, Ayz	$\to Axz$;
Baroco	Oyz, Ayx	$\to Oxz$;
Bocardo	Azy, Oxy	$\to Oxz$;
Barbari	Axy, Ayz	$\to Ixz$;
Cesare	Eyx, Ayz	$\to Exz$;
Dimatis	Ixy, Azy	$\to Izx$;
Ferison	Exy, Izy	$\to Oxz$.

6.2. The Small Calculus and Aristotelian Logic

Among the transmitted texts of Aristotle there is a formal system called "Aristotelian syllogistic" or "categorical logic". It is constituted by clearly defined syntactic rules. The seminal paper, in which Aristotelian logic is investigated from the point of formal systems appeared in 2005. It is called *"Aristotelian Syntax From a Computational-Combinatorial Point of View"* by Klaus Glashoff.[**Gla05**]. Glashoff systematically enquires into all the possible deductive bases and relates them to modern reconstructions of Aristotelian logic found in literature. Here is one of these bases.[1]
We note that table 1 contains a subbase which *coincides* with our three rules.

Definition 6.2.1. The rules

$$\text{Barbara } Axy, Ayz \to Axz;$$
$$\text{Dimatis } Ixy, Azy \to Izx;$$
$$\text{Barbari } Axy, Ayz \to Ixz;$$

form a *subbase* of Aristotelian logic.

The subbase is closed. That is, Barbara,Dimatis and Barbari are the *only* rules by which Axy, Ixy-forms can be deduced. We put this observation down in a remark.

Remark 6.2.2. The deductive subbase consisting of Barbara,Dimatis and Barbari is closed.

In the definition 6.1.3 of the closure operator, the axioms consisted of expressions of form Axx,$A0x$. Expressions of this form do not appear in Aristotle's formal logic.[2] Historically Axx (and Ixx which can be derived through Barbari), were added later. They were called "identities" (tautologies) and were considered superfluous, as they were true of everything. In Domains $A0x$ is true of every x and relates to identity.[3] Thus they belong to tautologies too.

Dynamic Domains rest on the properties of confluence and normal form. With Aristotelian logic we have a *larger system* which is *confluent* and reduces to *normal forms*, as Glashoff proved in lemma 2.3 and 2.4. This suggests, that Dynamic Domains can be substantially extended.

[1] According to theorem 4.3 of [**Gla05**]
[2] [**Gla05**] chapter 4.3
[3] See theorem 6.4.9

Appendix I: Domain Theory and its Foundation

The notational presentation of Domain Theory is from *Lectures on a Mathematical Theory of Computation* [**Sco81**]. But for the purpose of derivation, this presentation is incomplete. It is required that *all* the implied conditions of Domain Theory must be made *explicit*. Domain Theory is based on *order theory*. We begin therefore with an essential catalogue of order-theoretic definitions before the Domain Theory proper is presented.

6.3. Order Theory

Definition 6.3.1. A partial order is a pair $(O, \leq)$ where O is a set and $\leq$ a binary relation $\leq O \times O$ which is reflexive, transitive and antisymmetric. That is for any elements $X, Y, Z \in O$

(R)	$X \leq X$;	(Reflexivity)
(T)	if $X \leq Y$ and $Y \leq Z$ then $X \leq Z$;	(Transitivity)
(A)	if $X \leq Y$ and $Y \leq X$ then $X = Y$.	(Antisymmetry)

In partial orders, $X \leq Y$ has the same meaning as $Y \geq X$. $X < Y$ is short for $X \leq Y$ and $X \neq Y$.

Definition 6.3.2. Let $(O, \leq)$ be a partial order. A subset $M \subseteq O$ is called directed iff
(a) $M \neq \emptyset$;
(b) For all $X, Y \in M$ there is a $Z \in M$ with $X \leq Z$ and $Y \leq Z$.
An ω-chain is a sequence $(X_i)_{i \in \omega}$ such that $X_i \leq X_{i+1}$ for all i.

Remark 6.3.3. If $(X_i)_{i \in \omega}$ is a chain then $\{X_i | i \in \omega\}$ is directed.

Definition 6.3.4. Let X, Y be elements of a partial order $(O, \leq)$. $S \in O$ is called a least upper bound or supremum iff the the following conditions hold:

(a)	$X, Y \leq S$;	(Upper Bound)
(b)	For all $Z \in O$, if $X, Y \leq Z$ then $S \leq Z$.	(Least Upper Bound)

$T \in O$ is called greatest lower bound or infimum iff:

(c)	$T \leq X, Y$;	(Lower Bound)
(d)	For all $Z \in O$, if $Z \leq X, Y$ then $Z \leq T$	(Greatest Lower Bound)

hold.

If suprema and infima exist in a partial order, they are uniquely determined because of antisymmetry (A) Therefore we are allowed to write:

$$S = sup(X, Y) \text{ and } T = inf(X, Y).$$

Theorem 6.3.5. *Let $X \leq Y$. Then the supremum of X and Y is precisely Y, that is,*

$$X \leq Y \text{ always implies } sup(X, Y) = Y.$$

Theorem 6.3.6. *Let $X \leq Y$. Then the infimum of X and Y is precisely X, that is,*

$$X \leq Y \text{ always implies } inf(X, Y) = X.$$

Remark 6.3.7. Let X, Y be elements of a partial order $(O, \leq)$ with $X, Y, Z \in O$. Supremum and infimum are

commutative: $sup(X, Y) = sup(Y, X)$, $inf(X, Y) = inf(Y, X)$,
associative: $sup(sup(X, Y), Z) = sup(X, sup(Y, Z))$, $inf(inf(X, Y), Z) = inf(X, inf(Y, Z))$
idempotent: $sup(X, X) = X$, $inf(X, X) = X$.

Their value depends only on the set $\{X, Y, Z\}$.

Notation 6.3.8. *If X and Y are elements of O, we write $X \wedge Y$ for $\bigcap\{X, Y\}$ and $X \vee Y$ for $\bigcup\{X, Y\}$. These dyadic operators are called meet and join respectively.*

We now come to the concept of a *complete partial order* or *cpo*. 'Completeness' here refers to the underlaying class of increasing sequences. This class can vary. Here we use the most basic class of increasing sequences, the class of ω-chains.

Definition 6.3.9. Let $(O, \leq)$ be a partial order and $M \subseteq O$. An element $X \in O$ is called supremum of M written $\bigsqcup M$ if $X \leq Z$ for all upper bounds $Z \in M$. A (complete partial order (cpo) is a triple $(O, \leq, \perp)$ with $\perp \in O$ and

(a) $(O \leq)$ is a partial order;
(b) If $X \in O$ then $\perp \leq X$;
(c) $\bigsqcup M$ exists for every ω-chain $M \subseteq O$.

6.4. Domain Theory

In *Lectures on a Mathematical Theory of Computation*[**Sco81**] Scott introduces domains, that is cpo's, *indirectly* via neighborhood systems. A neighborhood system is defined as follows:

Definition 6.4.1. (Neighborhood System) A family D of subsets of a given set Δ is called neighborhood system (D, Δ) iff it is a non-empty family closed under the intersection of finite consistent sequences of neighborhoods. That is to say D must fulfill the two conditions:

(i) $\Delta \in D$;
(ii) whenever $X, Y, Z \in D$ and $Z \subseteq X \cap Y$ then $X \cap Y \in D$.

By convention Δ corresponds to the intersection of the empty sequence of neighborhoods. In particular,

$$\begin{aligned} \bigcap_{i<n} X_i =& \Delta, \text{ if } n = 0; \\ =& (\bigcap_{i<n-1} X_i) \cap X_{n-1}, \text{ if } n > 0. \end{aligned}$$

By (ii) this intersection property is extended to any finite sequence. Therefore $X_0, \ldots, X_{n-1}$ is consistent iff

$$\bigcap_{i<n} X_i \in D.$$

The ideal elements of a domain or their 'points', are defined in terms of the neighborhoods:

Definition 6.4.2. (Domain) The elements of a neighborhood system (D, Δ) are those subsets $x \subseteq D$ where:

(i) $\Delta \in x$;
(ii) $X, Y \in x$ always implies $X \cap Y \in x$; and
(iii) whenever $X \in x$ and $X \subseteq Y$ then $Y \in x$.

The domain of all such elements (filters) is written as $|D|$.

Domains possess important closure properties.[4]

Theorem 6.4.3. *If D is a neighborhood system and $x_n \in |D|$ for $n = 0, 1, 2, \ldots,$ then*

$$(i) \bigcap_{n=0}^{\infty} x_n \in |D|;$$

$$(ii) \bigcup_{n=0}^{\infty} x_n \in |D| \text{ provided}$$

$$x_0 \subseteq x_1 \subseteq x_2 \subseteq \ldots \subseteq x_n \subseteq x_{n+1} \subseteq \ldots.$$

Points correspond to *total elements* or maximal filters (maximal elements). There are also elements which are not total (maximal), the *partial elements* which correspond to principal filters and are *finitary.*

Definition 6.4.4. Let (D, Δ) be a neighborhood system according to Def. 6.4.1. For $X \in D$ the principal filter determined by X is defined by

$$\uparrow X := \{Y \in D | X \subseteq Y\}.$$

The correspondence between neighborhoods and finitary elements is *one-one.* The subset relation, infima and suprema are reversed:

$$X \subseteq Y \text{ iff } \uparrow Y \subseteq \uparrow X.$$

This, together with the congruence between *sets* and *elements of a partial order* provides the reason why neighborhood systems can be used to present cpo's.

Besides finitary elements there are total elements in a domain which are defined as 'limits' of a set of 'finitary' elements which are 'dense' in $|D|$. For each $x \in |D|$

Remark 6.4.5.

$$x = \bigcup \{\uparrow X | X \in x\}.$$

[4] See theorem 1.11 in [**Sco81**].

Points are total and maximal and *perfect* elements (ultra-filters). Principal filters are not total and maximal. They are *partial elements* and are as such *parts* of other elements, which can either be partial or ideal. The part-of or inclusion relationship between the elements is called *approximation-relation*.

Definition 6.4.6. For $x, y \in |D|$, we say that y approximates x if and only if $x \subseteq y$. The elements which approximates all others Δ is $\perp$ which is the 'least defined' and the 'most partial' element. Elements maximal with respect to the approximation relation are called total elements.

Instead of *approximation* the term *information order* is often used. If $x \subseteq y$ then y contains *at least* all the information of x and *maybe more*.

Functions are introduced into Domain Theory indirectly, like the elements. They are defined via a *binary relation* between neighborhoods which is called *approximable mapping*:

Definition 6.4.7. (Approximable mapping) An approximable mapping $f : D_0 \to D_1$ between domains is a binary relation between neighborhoods such that

$$
\begin{aligned}
(i) &\quad \Delta_0 f \Delta_1; \\
(ii) &\quad XfY \text{ and } XfY' \text{ always implies } Xf(Y \cap Y') \text{ and} \\
(iii) &\quad XfY, X' \subseteq X, \text{ and } Y \subseteq Y' \text{ always implies } X'fY'.
\end{aligned}
$$

Approximable mappings associate with functions between domain elements in a unique way. This relationship is subject of a proposition:[5]

Proposition 6.4.8. *(Approximable Function) An approximable mapping $f : D_0 \to D_1$ between two neighborhood systems D_0 and D_1 always determines a function $f : |D_0| \to |D_1|$ between Domains by virtue of the formula:*

$$(i) \quad f(x) = \{Y \in D_1 | \exists X \in x. \quad XfY\}$$

for all $x \in |D_0|$. Conversely, this function uniquely determines the original relation by the equivalence:

$$(ii) \quad XfY \textit{ iff } Y \in (\uparrow X)$$

for all $X \in D_0$ and $Y \in D_1$. Approximable functions are always monotone *in the following sense:*

$$(iii) \quad x \subseteq y \textit{ always implies } f(x) \subseteq f(y),$$

for $x, y \in |D_0|$; moreover two approximable functions $f : D_0 \to D_1$ and $g : D_0 \to D_1$ are identical as relations iff

$$(iv) \quad f(x) = g(x), \textit{ for all } x \in |D_0|.$$

There is an important theorem [6] which is given here without proof:

[5] Proposition 2.2 in [**Sco73**] p.166-167.
[6] It is theorem 2.5 in [**Sco81**] p.169.

Theorem 6.4.9 (Identity mapping, composition)**.** *The class of neighborhood systems and approximable mappings forms a category, where the identity mapping $1_D : D \to D$ relates $X, Y \in D$ as follows:*

$$(i) \quad X 1_D Y \text{ iff } X \subseteq Y.$$

If $f : D_0 \to D_1$ and $g : D_1 \to D_2$ are given, then the composition $g \circ f : D_0 \to D_2$ relates $X \in D_0$ and $Z \in D_2$ as follows:

$$(ii) \quad X g \circ f Z \text{ iff } \exists Y \in D_1.\ X f Y \text{ and } Y g Z.$$

'Monotony' in Domain Theory is the first principle of the qualitative notion of information. It states that information is *cumulative* only. In terms of situations: passing from one situation to the other, information is always retained, never forgotten. We may also say that information is *isotone* or *positive*. In contrast, 'ignorance' (lack of information) is not necessarily isotone. It may – with luck – be reduced, as we proceed. Ignorance may be *antitone*. Abramsky sees in monotonicity of Domain Theory a *principle of stability of observable information.* If we are in some information state y then it may be better described by a state x, $x \subseteq y$. However, among the information available in a given situation, *we may use only the information which is stable under any change whatsoever.*[7] To see that, let us assume the following: any situation *will* or *will not* provide grounds for asserting $x \subseteq y$. And we can always recognize whether it does; that is, a situation is assumed to be *decidable.* In this sense we assume the *tertium non datur.* However: If a situation does *not* provide grounds for asserting $x \subseteq y$, it does not mean that $x \subseteq y$ is false. The situation may change and more evidence comes to hand. Thus $x \subseteq y$, which hitherto *failed to be true* now becomes true. What is prohibited though, is the negative case: if $x \subseteq y$ was found to be true in a situation, the situation changes in such a way that $x \subseteq y$ now turns out to be false. Once we have grounds for asserting $x \subseteq y$ then they are supposed to be *stable* or *conclusive* grounds, so that no possible further evidence can upset them.

The second principle of qualitative information is 'continuity'. Again, a concrete definition of continuity depends on the underlaying class of increasing sets, which for our purpose are ω-chains.

Definition 6.4.10. A function $f : D_0 \to D_1$ is continuous, iff it is monotonic and for all ω-chains $(x_n)_{n\in\omega}$ in D_0:

$$f = (\bigsqcup_{n\in\omega} x_n) = \bigsqcup_{n\in\omega} f(x_n).$$

This equation is interpreted by Abramsky as the condition, that a computational process in each *finite state n* has only access to a *finite* amount of information. And, if we are provided with an infinite stream of input-data, the output-stream at any finite

[7] "...the conceptional basis for for monotony is that *the information in Domain Theory is positive; negative information is not regarded as stable observable information.* That is, if we are at same information state s, then for all we know, s may still increase to t, where $s \sqsubseteq t$. Thus we can only use information which is stable under every possible information increase in a computable process." [**Abr08**, p.495]

state is directly related to some finite input.[8]

The antisymmetry of the the underlying partial order is interpreted by Abramsky as a *principle of extensionality*[9] : if two states convey the same information $x \subseteq y, y \subseteq x$, then they are considered the same, $x = y$. But when are two domains 'the same'? 'Sameness' of domains is defined in terms of isomorphism between two domains which is in turn defined in terms of approximation-preserving correspondence.

Definition 6.4.11. Two neighborhood systems D_0 and D_1 determine isomorphic domains iff there is a one-one correspondence between D_0 and D_1 which preserves inclusion between the elements of the domain. In this case we write $D_0 \cong D_1$.

It can be proven that an isomorphism between domains always results from an approximable mapping which maps the finite elements of one domain to the finite elements of the other.[10] Let $f : D_0 \to D_1$ be such a bijection, $x, y \in D_0$, ($x \subseteq y$ iff $f(x) \subseteq f(y)$. Then f and its inverse f^{-1} are continuous.[11]

[8] "The continuity condition . . . reflects the fact that a computational process will only have access to a finite amount of information at each finite stage of the computation. If we are provided with an infinite input, then any information we produce as output at any finite stage can only depend on some finite observation we have made of the input." [**Abr08**, p.495]

[9] [**Abr08**, p.493]

[10] [**Sco81**] theorem 2.7

[11] [**Wei87**] p.407

Bibliography

[Abr08] Samson Abramsky, *Information, Processes and Games*, Philosophy of Information, Elsevier, Amsterdam [u.a.], 1. ed., 2008, pp. 483–550.

[Ari84a] Aristotle, *Metaphysics*, The Complete Works of Aristotle (J. Barnes, ed.), Bollingen Series LXXI * 2, vol. II, Princeton University Press, 1. ed., 1984, The Revised Oxford Translation, pp. 1552–1728.

[Ari84b] ______, *Prior Analytics*, The Complete Works of Aristotle (J. Barnes, ed.), Bollingen Series LXXI * 2, vol. I, Princeton University Press, 1. ed., 1984, The Revised Oxford Translation, pp. 39–113.

[CDE+07] Manuel Clavel, Francisco Durn, Steven Eker, Patrick Lincoln, Narciso Mart-Oliet, Jos Meseguer, and Carolyn Talcott (eds.), *All about Maude - a high-performance logical framework*, Lecture notes in computer science, no. 4350, Springer, Berlin [u.a.], 2007.

[Emp90] Sextus Empiricus, *Outlines of Pyrrhonism*, Prometheus Books, 1990, Translation R.G. Burry.

[Fre77] Gottlob Frege, *Begriffsschrift und andere Aufsätze*, 3. aufl ed., Wiss. Buchges., Darmstadt, 1977.

[Fri00] Michael Friedman, *A Parting of the Ways: Carnap, Cassirer and Heidegger*, Chicago, ILL [u.a.]: Open Court, 2000.

[Gla05] Klaus Glashoff, *Aristotelian Syntax from a Computational-Combinatorial Point of View*, Journal of Logic and Computation **15** (2005), 949–973.

[Gla06] Klaus Glashoff, *Zur Übersetzung der Aristotelischen Logik in die Prädikatenlogik*, Wanderschaft in der Mathematik (Augsburg, Germany) (Magdalena Hyksov und Ulrich Reich, ed.), E. Rauner Verlag, 2006, p. 258.

[Hei39] Martin Heidegger, *Vom Wesen und Begriff der PHYSIS*, Wegmarken (Friedrich-Wilhelm von Herrmann, ed.), Gesamtausgabe, vol. 9, Vittorio Klostermann, 1939, pp. 237–300.

[Hei67] ______, *What is a Thing?*, Gateway Editions Ltd.,, Chicago, 1967.

[Hei89] ______, *Die Grundprobleme der Phänomenologie*, 2. ed., Gesamtauflage, vol. 24, Vittorio Klostermann, 1989, Ed. Friedrich-Wilhelm von Herrmann.

[Hei92] ______, *Die Grundbegriffe der Metaphysik (II. Abteilung: Vorlesungen 1923-1944)*, 2. ed., Gesamtausgabe, vol. 29/30, Vittorio Klostermann, 1992, Ed. Friedrich-Wilhelm von Herrmann.

[Hei95] ______, *Logik. Die Frage nach der Wahrheit (Wintersemester 1925/26)*, 2. ed., Gesamtauflage, vol. 21, Vittorio Klostermann, 1995, Ed. Walter Biemel.

[Irv10] A. D. Irvine, *Principia Mathematica*, The Stanford Encyclopedia of Philosophy (Edward N. Zalta, ed.), summer 2010 ed., 2010.

[Jos16] Horace William Joseph, *An Introduction to Logic*, 2. ed., rev ed., Clarendon Press, Oxford [u.a.], 1916.

[KRS99] Geoffrey Stephen Kirk, John Earle Raven, and Malcolm Schofield, *The Presocratic Philosophers*, 2. ed., reprint ed., Cambridge Univ. Press, New York, NY [u.a.], 1999.

[Lem83] E. J. Lemmon, *Beginning Logic*, Van Nostrand Reinhold, Wokingham, 1983.

[Mil84] John Stuart Mill, *A System of Logic Ratiocinative and Inductive*, Longmans, Green, London, 1884.

[Plo83] Gordon Plotkin, *Domains*, Department of Computer Science, University of Edinburgh, 1983.

[Pra08] Vaughan Pratt, *Algebra*, The Stanford Encyclopedia of Philosophy (Edward N. Zalta, ed.), fall 2008 ed., 2008.

[Sco73] Dana S. Scott, *Models for Various Type-Free Calculi*, Methodology and Philosophy of Science, vol. IV, North Holland, 1973, pp. 157–187.

[Sco81] ______, *Lectures on a Mathematical Theory of Computation*, Theoretical Foundations of Programming Methodology, vol. 20, D. Reidel Pub., 1981, pp. 145–292.

[Sco82] ______, *Domains for Denotational Semantics*, Springer Verlag, 1982.

[Sto77] Joseph E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.

[Tar86] Alfred Tarski, *The Semantic Conception of Truth and the Foundations of Semantics (1944)*, Collected Papers I-IV, vol. II, Birkhäuser Verlag, Basel Bosten Berlin, First published in: Philosophy and Phenomenological Reaserach 4 (1944) 1986, quoted by section number., pp. 661–699.

[TM93] J.L. Turner and T.L. McCluskey, *The Construction of Formal Specifications: An Introduction to the Model-Based and Algebraic Approaches*, Mcgraw Hill Book Co Ltd, 1993.

[Vic06] John Vickers, *The Problem of Induction*, The Stanford Encyclopedia of Philosophy (Edward N. Zalta, ed.), Winter 2006.

[Wei87] Klaus Weihrauch, *Computability*, EATCS Monographs on Theoretical Computer Science, vol. 9, Springer-Verlag, 1987.