Formal Methods & Testing

Mícheál Mac an Airchinnigh[1]

6th September 1995

[1]The author is a Senior Lecturer at the Department of Computer Science, University of Dublin, Trinity College, Dublin, Ireland (e-mail: mmaa@cs.tcd.ie; http://www.cs.tcd.ie/www/mmacanai/mmacanai.html) and Technical Director of K&M Technologies Limited, Florence House, 1 Florence Villas, Bray, Co., Wicklow, Ireland (e-mail: mmaa@kmtech.ie; http://kmtech.ipdial.ieunet.ie).

Abstract

The tutorial addresses one of the key technologies, formal methods, that is reaching maturity in the software engineering domain, even to the extent that developers of safety critical systems may be open to special litigation in the event of accident/failure, if it can be shown that they have not taken *all reasonable precautions* in building their system. The use of a formal method does not obviate the need for testing, but it can serve as an innovative framework for better testing.

Requirements/specification bugs are the most pernicious of all. The methods by which they may be detected, at the phase in which they occur, have not received the degree of attention they deserve. To redress this serious imbalance, evident from the lack of literature, we dedicate ourselves exclusively, in this tutorial, to the use of an applied constructive mathematics, strictly within the framework of an *aformal method*, the Irish School of the Vienna Development Method (VDM♣), to the problem of bug detection at the requirements/specification level.

key words: applied constructive mathematics, formal method, requirements, testing, Vienna Development Method.

Contents

Chapter 1

A Context

> "Euclid already knows this [i.e., that it is a mistake to group together all the hypotheses under the name of *axioms* of the geometry], since he divides them into 'requirements' and 'common notions'. The 'requirements'—also called *postulates*—are the properties of plane geometry, while the 'common notions'—which at the end of the classical era came to be called '*axioms*'—concern all kinds of 'magnitude'." (Dieudonné 1992, 37).

1.1 Prologue

The time is rapidly approaching, assuming that it is not yet already at hand, when a system failure traced to a software fault, and injurious to a third party, will lead to successful litigation against the developers of said system software. The litigation will be successful in spite of any of the usual software disclaimers currently in vogue and in spite of the intensive system and software testing undertaken by responsible organizations and developers. How could that be possible? What factor or factors might an attorney use in pressing the claim for compensation?

We will assume that the defendant, the development organization, has in place an appropriate software development process, that adherence to the process has been assured by a vigorous quality programme, and that the best in testing techniques have been applied. We will further assume that the developers have been 'certified', in some sense, to be software engineers. How then could the defendant possibly lose?

Unfortunately, the smart attorney for the claimant may politely inquire to what extent the defendant has made **use of formal methods** in the software development and testing processes, and could the defendant please supply the appropriate documentation to support her/his claim.

Attorney for the defence, inadequately briefed by the defendant, will, of course, plead that formal methods are not mature enough, that it is impossible to prove large system programs correct, that there are not enough instructors or courses in formal methods, and that the universities and other third level colleges do not include it in their curricula.

Attorney for the claimant will counter-attack with a charge of negligence, that there is sufficient material in the public domain, and has been for some time (since **circa** 1978), that the defendant ought to have, at least, made the effort to deploy formal methods in the organization, and that the defendant is negligent to the extent that no effort was made to engage formal methods in the development and testing processes. Furthermore, our attacking attorney will rhetorically ask in what respect the software engineers can lay claim to that title and be ignorant of the mathematics of their field, or being knowledgeable, how could they be so negligent as to ignore what they know.

The final line of defence, in all probability, will be recourse to a plea that the pressures imposed by the **deadline**, the date of system delivery/handover, prohibited any 'experimentation' in formal methods. To which the defence, well-versed in the origins of words, will reply with a smirk: Whose deadline? That of the developers or the claimant who suffered from the negligence?

Lest it should be thought that the scenario, just outlined above, is nothing more than an 'attention-getter', then perhaps the formal methods debate, which (once?) raged in the UK, may provide further food for thought (Tierney 1992). Of particular note is the statement that

> "As a result of 00–55 [*The Procurement of Safety Critical Software in Defence Equipment*, UK MoD Defence Standard], formal methods cannot now be ignored as an option in the safety engineering tool-kit, an option moreover whose exclusion from—rather than inclusion in—the design process, must now be justified."(Tierney 1992, 269).

1.2 Formal Methods

There is an extraordinary wide variety of formal methods available to the software industry today, each of which requires a commitment to a specific philosophy, is distinguished from another by notation, method and field of application, and which needs for its continuing survival, a School of adherents which in turn has many of the attributes of a living organism, the most salient of which is, perhaps, self-perpetuation with a necessary inbuilt evolutionary mechanism which maintains its vitality and vibrancy[1].

But what are we to understand by the term 'formal method'? The consensus is that it denotes both a (specification) language and a method that is *rooted intrinsically* in mathematics and logic (and there **is** a difference between these two).

[1]At the time that this was written (May 1993) I was completely unaware of the *The Selfish Gene* (Dawkins 1989). However, having had the good fortune to come into contact with Prof. Yuzuru Tanaka and learned of his *IntelligentPad* system (Tanaka 1995) and understood the manner in which he envisaged a pad to be a technological multimedia manifestation of a **meme** (which Dawkins obtains as a contraction of *mimeme* [from μίμημα (Liddell and Scott 1986, 513)] to rhyme with **gene**), a concept which originates with Dawkins (Dawkins 1989, 189–201), then I realized that my recent neo-Darwinian musings both here and elsewhere (Mac an Airchinnigh and Kugler 1994) would appear to have some substance and would seem to call for further serious mimetic (cf., μίμησις: *imitation* (Liddell and Scott 1986, 513)) reflection. Naturally, the origins of mimēsis (and multimedia) and hence **meme**s are to be found in Aristotle's *Poetics* (Aristotle 1987, 540–56).

The qualifier *formal* is a particular twentieth century word that calls for a commitment to a very specific philosophy, the roots of which may be discerned in Hilbert's programme to formalize mathematics and in Russell's attempt to reduce mathematics to a body of formal logic, both of which failed utterly, and were subsequently shown to be theoretically impossible.

The subsequent work of Gödel, Kleene, Church and Turing, *inter alia*, heralded in a new era of formality which is today completely dominated by the computer scientists, one of whose first great achievements was the dual theories of *Formal Languages* and *Automata*, theories upon which all of our programming language technology depends.

The word 'formal' is derived from *form*, which is, of course, in opposition to *content*. The formalists seek to present a system that is, of itself, inherently independent of and devoid of meaning. An accessible account of the rise of formality in mathematics, which pre-dates that in computer science, is given by Morris Kline in the last chapter of his comprehensive work *Mathematical Thought from Ancient to Modern Times* (Kline 1972, 1182–211).

The form/content duality is also to be found in the syntax/semantics duality. Having mastered the theory of syntax and having built the machinery to process it, the formalists turned their attention to its dual, the theory of semantics—the meaning of the formal language. Therein was born the formal method that we know today.

It seems to the author, whose opinion is undoubtedly biased, that **the** first formal method of computer science (Bjørner and Jones 1978), ready bedecked with mathematics and logic, was the Vienna Development Method (VDM[2]), the specification language of which, replete with a suitable mathematical symbolism, was punningly called Meta–IV. That it deserves the name formal method may be further justified by the industrial concerns of its originators, or, more precisely, its sponsors, and the continued industrial concerns of all who espouse it. That is to say, the term 'formal method' has now acquired a connotation of industrial relevancy.

There are essentially two distinct classes of formal method:

1. the model-oriented formal methods, often called model-theoretic methods, of which the VDM (Bjørner and Jones 1978; Bjørner and Jones 1982), and $\mathcal{Z}$ (Woodcock and Loomes 1988; Diller 1990), are, perhaps the most widely known exemplars, being also one must add, among the most widely used in (European) industry;

[2]It was apparently Prof. Cliff B. Jones who first proposed the name Vienna Development Method. I seem to recall that he admitted as much to me in Odense, Denmark, in April 1993. In particular he regretted that he had ever done so because it seemed that the name was being used as a battle cry for rallying one's supporters.

In a subsequent private conversation in Odense, Prof. Peter Lucas further added the snippet that such naming and other things took place at a party on the balcony of his apartment in Vienna.

For the initiate, it is important not to get confused between the VDM and its 'precursor' the Vienna Definition Language which emanated from the same laboratory (Bjørner and Jones 1978, ix–xii).

2. the rest, which includes those based on axiomatics[3] and the process algebras[4].

The author begs the tolerance of all the other authors whose works have not been cited, and of the researchers and developers whose methods, being of equal importance, in their own way, to both the VDM and $\mathcal{Z}$, have not even been mentioned by name. **They** will appreciate the quandry. The formal literature is vast indeed.

Due to its longevity, as measured relative to the computer science era, which we deem to be approximately 45 years (1950–)[5], it is not surprising that evolutionary mechanisms[6] have forced the development of VDM dialects, each of which has its School. For the interested reader, a brief overview of these VDM dialects is given in (Toetenel 1992, 12–5). That dialect which is currently known as the Irish School of the VDM, denoted VDM♣ and humourously referred to by Jim Woodcock as Club Class[7] VDM in Odense, Denmark, April 1993, is regarded as the direct philosophical descendant of the IBM lab, being faithful to the spirit of the founding fathers:

> "We stress here, as was stressed in the introduction to this volume, that the meta-language [which was called Meta–IV] is to be used, not for solving algorithmic problems (on a computer), but for specifying, in an implementation-independent way, the architecture (or models) of software. Instead of using informal English mixed with technical jargon, we offer you a very-high-level 'programming' language. We do not

[3]Our hostility to axiomatics is quite well-known. However, it *must* be emphasized that this hostility is purely for pedagogical reasons—(software) engineers are not and should not be interested, from a professional point of view of course, in the foundations of the mathematics which they use in daily life. In addition one must distinguish between an informal axiomatics, which is part and parcel of the working mathematician's toolkit, and a complete rigorous axiomatics, much beloved of logicians and their kind. For those who wish to investigate things axiomatic, Stoll's little work on the subject (Stoll 1974) is as good a starting point as any.

[4]Process algebra or the algebra of behaviour complements the algebra of structure. The work by Hennessy is a good introduction (Hennessy 1988).

[5]Note well that we date the genesis of the computer science era with the emergence of both (practical) machine and programming language. To look further back towards Babbage and Lovelace on the one hand and on the other towards Gödel, Kleene, Church, Turing might be to confuse computer science with algorithmics, computation or logic?

[6]This was one of those unforeseen felicitous 'throw-away' remarks that perhaps may now be completely and rigorously justified in terms of *self-replicators*, which in this specific context, might be certain, as yet unidentified, mathematical computing memes (see yet again (Dawkins 1989))?

[7]For those unfamiliar with the aviation nuance, it corresponds roughly to First Class. On an historical note, the symbol ♣ was chosen to represent the *shamrock*, a small three-leaved plant much like clover which is the national plant of Ireland and is said to have been used by St. Patrick, the patron Saint of Ireland—the national holiday in Ireland is 17th March, the feast day of St. Patrick—to explain the mystery of the Trinity, or triune God. The Irish School of the VDM saw the light of day in the University of Dublin, Trinity College, roughly in 1987.

Marks such as VDM♣ may, unfortunately be interpreted as 'sectarian'. I like to think of them in the same manner as airline names. In an obvious sense a plane is a plane; but when run by an airline it acquires a certain character.

The basic mathematics underlying VDM and $\mathcal{Z}$ is the same; but ...

> offer an interpreter or compiler for this meta-language. And we have absolutely no intention of ever wasting our time trying to mechanize this meta-language. We wish, as we have done in the past, and as we intend to continue doing in the future, to further develop the notation and to express notions in ways for which no mechanical interpreter system can ever be provided." (Bjørner and Jones 1978, 33)

In this tutorial[8], we shall present some essential aspects of the applied constructive mathematics, which relies on ordinary algebra as a basis for conducting proofs.

With the passing of time, and with experience gained in applying the method, we have become accustomed to think of our work as an **a**formal method, where the Greek prefix 'α-' has the same meaning of negation that is found in common words such as **a**tom, **a**theist and **a**gnostic. That the method is within the formal methods community, is guaranteed by its philosophy of constructibility.

In addition, we strive to build a mathematics with which an engineer would feel comfortable. We have noted that formal logic is not used, in practice, by engineers, scientists, or mathematicians. We do not believe that computer scientists or software engineers are so distinguished by their field, that, of necessity, they must commit themselves to formal logic.

That said, we must confess, that even for those who are *au fait* with mathematics, such as engineers and scientists, the mathematics we use and, especially its notational accoutrement, comes something of a shock, initially. On the other hand, computer scientists, software engineers, and programmers [**not** coders!], i.e., whose very world and culture is, to some extent, determined by the intrinsic duality of machine and language, and by the absolute necessity of constructibility, will find their thoughts and concepts reflected by and abstracted within the applied constructive mathematics that characterizes methods such as the VDM♣.

We do not wish that our remarks on formal logic, scattered throughout this tutorial, should be misconstrued as being in some sense derogatory of the work being done by those who have committed themselves to it. Rather, we have taken a deliberate scientific standpoint, a point of view, and propose that one place mathematics, logic, and method, on a par with, say, the experimental tools—the equipment—of the chemists, physicists and biologists. In other words, the very fact that most formal methodists espouse formal logic as the vehicle for reasoning and proof, is in itself, an incentive to take an alternative approach, thereby placing the School in opposition to the others in a very definite way, for which we make no apology. Where there is such difference and diversity then one can expect strong growth among all—subject, of course, to the usual predator-prey mathematical model[9]. In this respect, 'standardization' of a formal method, whilst being in some sense necessary and essential for industrial engagement, especially for the purpose of tool building to assist in the

[8]The first *version* was presented at the International Software Quality Week in San Francisco in May 1993 (Mac an Airchinnigh 1993) and subsequently in 1994 and 1995. Further detailed information on the method may be found in an approximately one hundred page tutorial presented at VDM'91 in Noordwijkerhout, The Netherlands (Mac an Airchinnigh 1991). The genesis of the philosophical foundations are contained in the author's doctoral thesis (Mac an Airchinnigh 1990).

[9]Need a good reference.

management and control of the complexity of system specification, a complexity that is several orders of magnitude less than that of system code, is nevertheless a formidable obstacle to creativity, ingenuity, and evolutionary growth. One does not 'standardize' mathematics.

The tutorial text is **dead**, once created. It is there both for the benefit of those who participate personally in the tutorial, acting as an *aide mémoire* to the historical events and the dialogue of the tutorial itself, and for those who did not partake in the dialogue—the tutorial—for whom the event can not possibly have any direct influence. It is an historical imperative that the dead letter must be directed primarily at non-participants and it is, therefore, to them that this tutorial text must be primarily addressed.

Before presenting the specific mathematical material as applied to the chosen application, taken from the domains of medical information and network communication, it behoves us to say a few words on testing. After all, the word is included in the title of the tutorial.

1.3 Testing

It will be assumed that the reader is familiar with an infrastructure and culture such as might have emerged in conformance with the Federal Information Processing Standard on the validation, verification, and testing of (computer) software (National Bureau of Standards (U.S.) 1990), or any comparable national, international, or industrial set of guidelines for same.

It is surely self-evident from the title of the tutorial, in the placing of formal methods before testing, and in the very juxtaposition of the two terms, that our treatment of testing will differ considerably from current accepted practice or theory. We do not propose to do away with software testing techniques or to render a text such as that of Beizer (1990) redundant. If one is searching for further software testing techniques, then this is not the tutorial for them. There are contemporaneous sibling tutorials to address this aspect. What **is** to be found, though taking second place, is a sound theoretical and foundational framework for testing, a sort of skeleton, upon which and with respect to which, testing techniques have a place.

Due to its precise mathematical nature, a formal method permits the term *verification*—the building of the system rightly—to attain the fullness of its meaning. Within the VDM, there are planes of abstractions, both vertical and horizontal, expressed in terms of models and their operations. A *downward* movement, by which models on one plane are developed into models on a lower plane, i.e., that which is nearer to the ultimate realization of the system—the implementation, is usually known as a *reification* or downward refinement. A *lateral* movement is an *elaboration* on the same plane of abstraction. In such an elaboration we move through lateral planes by adding information, often in an exploratory fashion.

The method (M) in the VDM, posits a variety of choices for both reification and elaboration, and provides the mathematical techniques, among which the **retrieve** functions feature prominently, by which one may prove rigorously the validity of a development (D). In such a context does verification take on its fullest meaning.

Validation—the building of the right system—is to be construed as the pro-

cess by which it may be shown that the system to be built, and subsequently, the delivered system, satisfies the requirements and, therefore, ultimately the clients who need the system and who are bearing the costs. We have already exhibited, in considerable detail, the use of the VDM♣ for verification (Mac an Airchinnigh 1990; Mac an Airchinnigh 1991). In this tutorial, we take the opportunity to make public for the first time an extensive treatment of the method of the VDM♣ by which requirements modelling and requirements specification are to be validated, **within their phase** of the life-cycle.

In this context we are wont to refrain from using the term *testing*, since it has very strong connotations with software and system testing. In particular, in all aspects of telecommunications systems, it is almost impossible to speak about requirements and specification testing, so forcefully is the term bound to a 'testing culture' and the accompanying specific set of testing techniques used in practice. In consequence, we coined the phrase 'the determination of critical examination criteria for validity'[10] to replace the term testing when we wish to be precise and unambiguous about the method by which we determine bugs in system requirements and system specification in the pre-development phases of the life-cycle.

To illustrate what we mean by the *determination of critical examination criteria*, in a context other than that of the mathematical methods to be presented, we have found the following real-life example to be particularly illuminating. The case in question concerns the terrible disaster that befell the space shuttle Challenger on 28th January 1986 (Gleick 1992, 415), a failure that was subsequently determined to be due to (i) physical failure of the so-called O-rings, and (ii) managerial failure that permitted a launching deadline to overrule the results of the standard testing techniques (Gleick 1992, 426).

The test that led to exposure of the failure was as breathtakingly simple in its nature, as it was far reaching in its implications for **all** systems engineering (Gleick 1992, 423):

> Dr. Feynman: This is a comment for Mr. Mulloy [*project manager for the solid rockets*]. I took this stuff [*i.e., rubber*] that I got out of your seal [*i.e., the Viton O-ring*] and I put it in ice water, and I discovered that when you put some pressure on it for a while [*For the test, Feynman had earlier purchased a small C-clamp and pliers in a hardware store in Washington. The provision of a carafe of ice-cold water and glass completed the test equipment.*] and then undo it, it doesn't stretch back. It stays the same dimension. In other words, for a few seconds at least and more seconds than that, there is no resilience in this particular material when it is at a temperature of 32 degrees [*Fahrenheit*].
>
> I believe that has some significance for our problem.

One must place this test in context. Despite all of the resources available, and the testing techniques that must have been applied, why is it that the Feynman test, so cheap and so simple, should make (testing) history? The answer is as

[10] Naturally, one can not really take this too seriously. Indeed, even now, I am inclined to break out in a fit of laughter when I read this convoluted expression!

On the other hand it took so much time to invent the phrase that I am loathe to abandon it.

simple as the test itself. There are some very fundamental (physical) principles that can not be ignored, and consequent fundamental (engineering) principles that must not be violated without, as in this case, disasterous repercussions. Physical principles are well established and well known, with an astonishing variety of experimental data to support them, and a wealth of mathematics that brings coherence to the interpretation of the experimental results. Is that not also true of computing experiments—the programs that we write and the systems that we build? What then are the corresponding computing principles and where is the mathematics to support them?

It will be argued and demonstrated in this tutorial that formal methods supply at least some of the answers.

We shall conclude this prologue by addressing ourselves to systems requirements, and what better way than to give a definition of requirements verifiability from the ANSI/IEEE standard that sets the guidelines for Software Requirements (IEEE Computer Society 1990, 26):

Definition 1 *A requirement is verifiable if and only if there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.*

Guidelines are one thing, definitions are another, and the reality is a third. In reality, many requirements are strictly *delta* requirements[11], an expression of the need for an enhancement, i.e., added functionality, to an existing system. Rarely is one given the opportunity of capturing, analyzing, modelling, and specifying, the requirements for a system *ab initio*.

Those who have been responsible for such systems requirements, whether as deltas or for new systems, know that certain sentences, felicitous requirements statements, already trigger potentially realizable implementation strategies. This is all the more the case for the formal methodist. A brief statement of description is usually sufficient to trigger the model building activity, and having begun with the names of concrete domains drawn from that description, a flood of purely abstract models rushes in, each model proposing itself for consideration, and leading to a frenzy in reuse. For this indeed is in the very nature of mathematical modelling—many of the necessary abstractions are already available. Nor is this activity localized to requirements modelling. It is a phenomenon familiar to all (applied) mathematicians, whatever the field, a phenomenon that leads one to speak of mathematical *discovery* rather than mathematical *invention*.

Thus, although we have argued elsewhere (Mac an Airchinnigh 1991) against the inlining of formal methods in a deadlined development project, this was not due to some unstated inherent weakness in formal methods *per se*, but rather in the realistic appraisal of the amount of expertise available in the field.

This remark requires us to comment on what may realistically be expected of a formal methods course, or of what the likely impact will be on an organization,

[11]The delta requirements phenomenon is typified by the attempt to produce written requirements to satisfy a request for certain functionality, for example, with respect to an existing system for which there are either (i) no written requirements or (ii) any written requirements that do perchance exist, do not reflect the true nature of the running system.

if it should seriously embrace a formal methods culture, and integrate it into its software process.

Based on experience, we estimate that, if introduced into an organization in 1993, formal methods will bear fruit in a tangible manner, three to four years hence[12]. By that we mean to say that the formal methodists will by then have become integrated (and accepted) within the organization, and their mathematical techniques will have become an integral part of the software process. A comparable time span is to be found in small software start-up companies from initial team-building activities and product idea brainstorming until first successful product release.

To continue with the company start-up analogy, we assert that there is a critical-mass factor which is absolutely essential for success. That is to say, we proclaim that perhaps as few as five formal methodists are necessary. This is based on both psychological and sociological factors[13], and the **key** sociological factor for formal methods, as for mathematics in general, is that both are intrinsically social activities. Proof is a social activity. Automation of proof may be desirable in certain cases ... but there follows the corollary that the proof is not interesting/significant mathematically.

But from where are all these formal methodists to come? A two-week course, for instance, is certainly not sufficient to train/educate someone in the exploitation of formal methods and their application to real projects. The two-week course **is** necessary to prepare the organization, management, development team, etc., to be able to embrace the new technology, to alert the development teams to the potential richness and added-value, and to deflate potential perceived threats to 'the way things are done'!

In preparing for this tutorial, and having in mind the potential for litigation in the medical domain, and in view of the subject matter, the idea of focusing on a system that processed blood tests suggested itself. That such a system might exist, did not deter us from modelling one, and we considered the pun on test to be charmingly worthwhile.

Thus to set the scene, we hurriedly scribbled a few contextual statements and decided upon a requirement, the invalidation of which would prove not merely embarrassing to all concerned, but that would lead, almost certainly, to litigation in certain circumstances.

Example 1 (Blood Test System) *Reflections of the author as they were scribbled in November 1992.*

1) A hospital laboratory performs blood tests for the patients of a group of medical practitioners who are affiliated to/registered with the hospital.

2) In the interests of privacy, a general practitioner has access only to the blood tests of her/his own patients.

3) By law (of MMA), the hospital has access to all blood tests to monitor community health.

[12]These figures are, strictly speaking, taken out of the air.

[13]On re-reading what I wrote, I was inclined to say that these figures also were snatched out of the air. Then I reflected that perhaps the statement was a reasonable expression of my understanding of reality. To justify it, one might start with sayings such "Two's a company, three's a crowd".

4) The hospital does not have access to patient identifiers, only to the blood test identifiers (and results, of course), and to the corresponding general practitioner.

5) ...

Requirement 1 *Blood tests must not be mixed up.*

From this text a development is exhibited. It must be confessed that we thought to rewrite the description to reflect more accurately the mathematics that did eventually emerge. Such a re-polishing, both of description and mathematical formulation, is commonplace. However, we felt that an honest historical account would have a greater impact. In addition, it presents an interesting exercise to determine in what respect the mathematical development has diverged from the original description.

We begin in the following section (§ **2**) with a well-paced presentation of the notion of a model and its operations, and introduce some of the fundamentals of the notation we use. Embedded within this introduction is a very brief summary[14] of some of the salient aspects of the algebraic theory underlying and supporting the notation. It will be observed that the mathematical theory is at the level of a first year or second year undergraduate course in mathematics at College or University.

Whereas § **2** is a direct outcome of the initial requirements text, with its emphasis on doctors, patients, and blood tests, § **3** introduces the model for a centre for disease control, inspired by those which are established in the U.S.A. and in France.

Already by § **3**, the name blood test was being considered conceptually as a place holder for any test or other event that might be related to the notification of a communicable disease, including, for example, such events as the HIV serological prescription used in France (Garnerin et al. 1991). The focus has, however, now moved to the communication network, which permits us to demonstrate (cf., Q.E.D. ≡ **quod erat demonstrandum**) what we mean by a proof and to show how, in practice, theoretical results are arrived at and used. This material of § **4** is, by far, the most technical of the tutorial.

Having introduced the transport medium for blood test data transmission, and with a view to a major generalization of the application, we present some further aspects of network modelling in § **5**. It is, of course, completely independent of the medical application.

§ **6** completes the tutorial with a few succinct observations and concluding remarks. We had originally intended to present an outline of a multimedia services application which naturally extended the medical information system and the network system. The blood test place holder was conceptually extended to embrace image, audio, and data, which doctors and labs might wish to transmit to the disease centre, or to one another, and, by inference, to a full general multimedia service application via total abstraction. This work is still in progress.

[14]There are different books for different people. Since my own background is pure mathematics, then the books that I use and cite are really the ones that suit me very well. For example, for the algebra, I *now* like to mention Jacobson (1985). I do not know how it may seem to the initiate.

Chapter 2

Models

> "With a view to requirements modelling, the purpose of a model is to ask questions and demonstrate that answers can be given entirely in terms of the model. If such answers can not be found then the model is inadequate." (Mac an Airchinnigh 1991). *On the other hand, it is quite possible that the wrong questions are being asked!*

A good recommendation in writing is to begin at the beginning. We take these words to heart, keeping in mind that there may be those for whom this introduction is exceedingly elementary, and try within the allotted space to be as overtly simple as possible. When we think of blood test, we might naturally think of doctors and hospitals. Therefore, it seems quite reasonable that we begin by examining the relation that a doctor bears to her/his patient[1].

Example 2 (Doctor-Patient Relation) *Each doctor d has a set of patients p_1, p_2, ..., p_n. Such a doctor-patient relationship, viewed from the perspective of the doctor, will be designated a mapping, thus:*

$$[d \mapsto \{p_1, p_2, \ldots, p_n\}]$$

For some other doctor d', i.e., $d' \neq d$, we will have a similar mapping:

$$[d' \mapsto \{p'_1, p'_2, \ldots, p'_n\}]$$

The space of all such doctor-patient relationships is usually designated by

$$DOC \rightarrow \mathcal{P}PAT$$

where the symbol $\mathcal{P}$ denotes the power set functor[2]. A typical collection of doctors and their patients will be denoted by some variable name which is

[1] It seems to us that one ought to be familiar with the usual mathematics of sets such as might be found in the introduction to any good algebra book, such as (Jacobson 1985), for example.

[2] Given a set $A = \{a, b, c\}$, then the powerset functor applied to A, denoted $\mathcal{P}A$ will give us the set of all the subsets of A: $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, A\}$. For further details, consult for example, (Jacobson 1985, 3).

customarily a letter, say the Greek letter μ. For convenience, when specifying the model, we will usually attach such a variable name thus:

$$\mu \in DOC \rightarrow \mathcal{P}PAT$$

Once we have written down a model, we may immediately write down mathematical expressions and provide possible **interpretations**. For example, given a collection of doctors and their patients μ, and some doctor d which is recorded in μ, then the expression

$$\mu(d)$$

designates the patients of doctor d. Alternatively, we may view the expression as the response to the question

Q: Who are the patients of doctor d?

However, we consider the expression to be valid if and only if doctor d is actually recorded in the given collection μ. We specify this validity condition, in terms of the characteristic function[3], by the expression

$$\chi[\![d]\!]\mu$$

which is our preferred form[4] for the alternative $d \in \mathsf{dom}\,\mu$. In this way we ensure that all of our maps are *total* with respect to their own local domains. Moreover, construction of the maps may immediately be interpreted as map transformations.

In presenting our work to others, or for recording purposes in archival documents, it is customary to provide a heavy sprinkling of syntactic sugar for the simple mathematical expression $\mu(d)$. Thus, the query on the patients of some doctor may be considered to be a (table) lookup operation, which we might characterize in the form:

1 LookUp: $DOC \longrightarrow ((DOC \rightarrow \mathcal{P}PAT) \longrightarrow \mathcal{P}PAT)$
.1 LookUp$[\![d]\!]\mu \triangleq \mu(d)$

and the validity constraint, i.e., that d is in the domain of μ, is expressed as a pre-condition of the 'lookup' operation:

2 *pre*–LookUp: $DOC \longrightarrow ((DOC \rightarrow \mathcal{P}PAT) \longrightarrow \mathbb{B})$
.1 *pre*–LookUp$[\![d]\!]\mu \triangleq \chi[\![d]\!]\mu$

[3]Although, in practice, we use the characteristic function in the curried form $\chi[\![a]\!]S$ in place of the test for set membership $a \in S$, it is important to note that in classical mathematics the characteristic function is customarily written in the form $\chi_S(a)$ (See, for example, (Jacobson 1985, 5)).

[4]I believe that I chose these strange-looking brackets '$[\![$' and '$]\!]$' from the Oxford style of specification of the denotational semantics of programming languages. Thus, if mk-While(e, s) denotes a typical while loop construct in a programming language and if ρ, σ denote environment and store, respectively, then one might describe the denotational semantics of the while loop as the value of a '**meaning**' function $\mathcal{M}$ of the form $\mathcal{M}[\![$ mk-While$(e, s)]\!](\rho, \sigma)$.

But the expression $\chi[\![d]\!]\mu$ itself may also be wrapped up in syntactic sugar as an 'is-recorded' operation:

$$\begin{array}{ll} 3 & \text{isRecorded}: DOC \longrightarrow ((DOC \to \mathcal{P}PAT) \longrightarrow \mathbb{B}) \\ .1 & \text{isRecorded}[\![d]\!]\mu \triangleq \chi[\![d]\!]\mu \end{array}$$

Being a predicate, it does not have a pre-condition. Use of this predicate name allows one to express the pre-condition of the lookup operation in a wonderfully verbose form:

$$.1 \quad \textit{pre}\text{–LookUp}[\![d]\!]\mu \triangleq \text{isRecorded}[\![d]\!]\mu$$

With frequency of use it is customary to abbreviate operation names such as LookUp$[\![d]\!]$ by Lkp$[\![d]\!]$, or even the completely mathematical form $L[\![d]\!]$.

Syntactic sugaring may also be applied to parameter names, of course. Thus instead of the purely symbolic d, one might use *doc* or even *doctor*. However, such syntactic sugaring, whilst being 'user-friendly', is rarely if ever used in mathematics, tending, as it does, to obstruct the clarity of the expressions. Indeed, the very use of model names such as DOC and PAT obscures the remarkable amount of reuse available via the abstract $X \to \mathcal{P}Y$ form.

Let us now explore further our model of doctors and patients. It will give us an opportunity to present other fundamental mathematical operations and expressions. First, a typical collection of doctors and their patients may be represented by

$$\mu = [d^1 \mapsto \{p_1^1, p_2^1, \ldots, p_j^1\}, d^2 \mapsto \{p_1^2, p_2^2, \ldots, p_k^2\}, \ldots]$$

Note our decision to use both superscript and subscript notation for indexing. It is included to indicate some of the experimental work currently being investigated in the VDM$^{\clubsuit}$ to try to adopt/adapt tensor notation for our own purposes.

For convenience we may exhibit the same information in 'matrix' form:

$$\mu = \left[\begin{array}{lcl} d^1 & \mapsto & \{p_1^1, p_2^1, \ldots, p_j^1\} \\ d^2 & \mapsto & \{p_1^2, p_2^2, \ldots, p_k^2\} \\ \ldots & & \ldots \\ d^m & \mapsto & \{p_1^m, p_2^m, \ldots, p_l^m\} \end{array}\right]$$

The quotes around matrix are necessary to emphasize the lack of ordering by column in such a map. Ordering is an essential concept in the normal matrix. However, now that we have made the comment, we feel free to use the word matrix unquoted to describe such a display.

If we wish to inquire as to the set of all the doctors recorded in the collection, then the answer is immediately given by

$$\mathsf{dom}\,\mu = \{d^1, d^2, \ldots, d^m\},$$

which is the column of the matrix. If, on the other hand, we inquire as to the set of all patients in the matrix, then the operator rng gives the second column:

$$\mathsf{rng}\,\mu = \{\{p_1^1, p_2^1, \ldots, p_j^1\}, \{p_1^2, p_2^2, \ldots, p_k^2\}, \ldots, \{p_1^m, p_2^m, \ldots, p_l^m\}\}$$

But technically the question calls for the result

$$\{p_1^1, p_2^1, \ldots, p_j^1, p_1^2, p_2^2, \ldots, p_k^2, \ldots, p_1^m, p_2^m, \ldots, p_l^m\}$$

which may be obtained by flattening, i.e., applying a reduction with respect to set union, of the range result:

$$\begin{aligned}
{}^{\cup}/\,\mathsf{rng}\,\mu &= {}^{\cup}/\{\{p_1^1, p_2^1, \ldots, p_j^1\}, \{p_1^2, p_2^2, \ldots, p_k^2\}, \ldots, \\
&= \{p_1^m, p_2^m, \ldots, p_l^m\}\} \\
&= \{p_1^1, p_2^1, \ldots, p_j^1\} \cup \{p_1^2, p_2^2, \ldots, p_k^2\} \cup \ldots \\
&= \cup\{p_1^m, p_2^m, \ldots, p_l^m\} \\
&= \{p_1^1, p_2^1, \ldots, p_j^1, p_1^2, p_2^2, \ldots, p_k^2, \ldots, p_1^m, p_2^m, \ldots, p_l^m\}
\end{aligned}$$

Explorations: Even at this early stage in our presentation, it is clear that there are very interesting avenues to explore. One is based on the notion of *counting*, i.e., on the notion of number[5]. For example, if we let $|A|$ denote the number of elements in A, i.e., the cardinality of A, then it is clear that for every doctor-patient relation μ, the number of doctors must always be greater than or equal to the number of suites[6] of patients:

$$|\,\mathsf{dom}\,\mu| \geq |\,\mathsf{rng}\,\mu|$$

Furthermore, inequality is only possible in that exceptional situation where two different doctors have exactly the same suite.

Such an exceptional case prompts us to consider the possibility where two suites are *almost identical*, for example $S_1 = \{a, b, c, p\}$ and $S_2 = \{a, b, c, q\}$. On the other hand we might consider the possibility where two suites are *almost distinct*, say $S_3 = \{a, b, c, p\}$ and $S_4 = \{r, s, t, p\}$. Although the suites are distinct in both cases, it is clear that the numbers of patients involved are different. Specifically, it is always the case that for two suites S and T,

$$|S \cup T| \leq |S| + |T|$$

with equality precisely in that exceptional case where the suites are *disjoint*, i.e., have no members in common. We may extend this result to all the suites in our doctor-patient relation. If $\mathsf{rng}\,\mu = \{S_1, S_2, \ldots, S_k\}$ denotes the collection of all suites then we have

$$|S_1 \cup S_2 \cup \ldots \cup S_k| \leq |S_1| + |S_2| + \ldots + |S_k|$$

Equality pertains in exactly that unique case where each pair of suites is disjoint, i.e., have no members in common. In such a case we say that the suites of patients form a **partition**[7].

This particular exploration permits us to exhibit an important distinction between the way we do mathematics and the way we **might** do mathematics in

[5]It is perhaps a truism to say that there is no mathematics without number. Indeed, even in formal methods work, if we can not use number even implicitly, then we can not be said to be doing mathematics. Hence, even when engaged in seemingly mundane specification, there will always be a question lurking in the back of our minds: *Where does number fit in?*

[6]For convenience, we consider the set of patients asociated with a doctor to be a *suite*. So, just as we have a pride of lions, a gaggle of geese and a school of fish, then we might say that a doctor has a suite of patients.

[7]Although we are not usually interested in the numbers of patients, we *are* interested in whether we have a partition or not. The concept of **partition** is foundational to

a completely formal setting. Specifically, the **naming** of the different suites, S_1, S_2, ..., S_k is a matter of convenience in establishing the inequality given. But, suppose that we must achieve the same result without the use of such auxillary names[8]?

From our earlier work above, it is clear that the left-hand side of the inequality, $|S_1 \cup S_2 \cup \ldots \cup S_k|$, may be replaced by $|^{\cup}/\,\mathsf{rng}\,\mu|$. To eliminate the variables on the right-hand side is not so easy. First, we introduce a function α which maps a set S to a maplet[9] of the form $[S \mapsto |S|]$, thus associating with each suite S, the number of members in S. Then iterating with α over the range of μ will give a collection of such maplets. In other words, $\mathcal{P}\alpha\,\mathsf{rng}\,\mu$, is just a set of unique counters. We can combine these to form a single table, $^{\oplus}/\mathcal{P}\alpha\,\mathsf{rng}\,\mu$. Such a table has the structure of a bag[10]. Finally, The number of elements in the bag is given by $|^{\oplus}/\mathcal{P}\alpha\,\mathsf{rng}\,\mu|$ and this is just the right-hand expression which we have been seeking. The inequality with all auxillary names removed now has the form:

$$|^{\cup}/\,\mathsf{rng}\,\mu| \leq |^{\oplus}/\mathcal{P}\alpha\,\mathsf{rng}\,\mu|$$

We have not been completely successful in the elimination of names. Our result includes the name of a function, α, which plays a crucial role[11].

Let us continue to examine our model by proposing other obvious questions.

Q: What happens if we wish to register a new doctor d?

In other words, we have an existing collection μ and some new doctor d is to be registered, i.e., we have the pre-condition $\neg\chi[\![d]\!]\mu$. In this case we extend the collection using a special extend operator, $\sqcup$, which is expressly provided for this purpose:

$$\mu \sqcup [d \mapsto \emptyset]$$

In particular, $\mu \sqcup [d \mapsto \emptyset]$ is valid, i.e., mathematically defined, only if d does not occur in the domain of the map, $\neg\chi[\![d]\!]\mu$.

> **Commentary:** The map extend operator, $\sqcup$, is used wherever we wish to merge two disjoint maps. It is intuitively simple—essentially a glueing operator.

---

computer science.

In our search for 'meaningful descriptive language' we have stumbled upon the "asparagus spears" analogy (Goldblatt 1984, 90). A (fully) partitioned doctor-patient relation is like a bundle of asparagus spears, where each doctor is the root and her/his suite is the spear head. Certain mathematicians might call such a relation a *fibre bundle*.

[8] In computer science such auxillary names are essentially temporary variables. To eliminate such variables, we must construct an appropriate expression using well-defined operators and functions. There must of course be some sort of starting point which in this case will be μ.

[9] A maplet is a total function with one domain element and one range element.

[10] A bag is like a purse of money. Two bags β_1 and β_2 may be combined to form a single bag $\beta_1 \oplus \beta_2$. The definition is simple. Take an empty bag and pour the contents of each of the two bags into it.

[11] Even this name may be removed, I suppose, by resorting to λ-notation.

Formally, for all μ_j, μ_k in $X \to Y$, $\mu_j \sqcup \mu_k$ is defined only if $\mathsf{dom}\,\mu_j \cap \mathsf{dom}\,\mu_k = \emptyset$, in which case,

$$\mu_j \sqcup \mu_k = \{x \mapsto y \mid (y = \mu_j(x)) \vee (y = \mu_k(x))\}$$

We may extend the definition of this glueing operator to non-disjoint maps μ_j and μ_k only if they are identical over the common domain $\mathsf{dom}\,\mu_j \cap \mathsf{dom}\,\mu_k$ (cf., the glueing lemma in (Armstrong 1983, 69).) The above formal definition still holds. However, to signify the distinction between strict map extension of a pair of disjoint maps and our extended definition, we will denote the latter by $\mu_j \cup \mu_k$. This glueing operator may be expressed in terms of the original map extend operator:

$$\mu_j \cup \mu_k = \lhd\!\!\lhd [\![\mathsf{dom}\,\mu_k]\!]\mu_j \sqcup \mu_k$$

We use the map extend operator for the specification of the declaration of new variables within a *block* in a programming language.

Note that patients associated with the doctor are not registered at this time. For this purpose, we have another special operator, the override operator †. Specifically, let us suppose that the patient p of doctor d is to be recorded. We may specify this by

$$\mu \dagger [d \mapsto \mu(d) \cup \{p\}]$$

One will note that the occurrence of $\mu(d)$ in the above expression implies the necessary[12] precondition $\chi[\![d]\!]\mu$.

Commentary: The override operator, †, may be conveniently defined in terms of the extend operator. Suppose that we have two maps μ_j and μ_k and we wish to form $\mu_j \dagger \mu_k$. It is a simple matter of surgery. First we remove from μ_j all those maplets whose domain elements occur in μ_k and then glue on μ_k back onto the rest of μ_j.

For all μ_j, μ_k in $X \to Y$,

$$\mu_j \dagger \mu_k = \lhd\!\!\lhd [\![\mathsf{dom}\,\mu_k]\!]\mu_j \sqcup \mu_k$$

Of course, in the special case that the two maps are disjoint, then the override operator reduces to the extend operator. Hence we say that the override operator *covers* the extend operator. The same holds true, of course, for the override of two maps which agree on the intersection of their domains.

The override operator is exactly what we need to specify the semantics of the *assignment statement* in a programming language.

Since the override operator may be rigorously defined in terms of the deletion operator, $\lhd\!\!\lhd [\![d]\!]$, which removes all information associated with d from the map, and the extend operator, then the registration of new patients may be written in the form:

$$\mu \dagger [d \mapsto \mu(d) \cup \{p\}] = \lhd\!\!\lhd [\![d]\!]\mu \sqcup [d \mapsto \mu(d) \cup \{p\}]$$

[12] In the VDM$^{\clubsuit}$ all of our maps are total with respect to their domain. Hence $\mu(d)$ is only valid if indeed d is in the domain of μ. The override operator itself is always defined.

Such a rewriting turns out to be invaluable in carrying out certain types of proofs. Lest we forget, $\lhd [\![d]\!]$, is a convenient form for the more precise $\lhd [\![\{d\}]\!]$. In other words, we tend to simplify notation wherever possible. Other convenient forms are $\lhd_d$ and $\underset{d}{\lhd}$.

> **Commentary:** We consider the removal operator to be useful in specifying the semantics of local variables. Once control passes out of a block, any temporary variables created and the associated storage, must be removed. This is the case, for example, in *procedures*. However, it is important to note that this is not the usual way in which the denotational semantics of procedures is given.
>
> The strange form of the removal operator is due to $\mathcal{Z}$. That it is customarily expressed as a functional operator in the VDM♣ ought to become clear in the context of more elaborate specifications, to be presented later. We like to think that it resembles the head of an axe.

One will wonder whether a doctor and her/his patient could be registered at the same time. Well, of course, if that is the requirement, then the expression needed is

$$\mu \sqcup [d \mapsto \{p\}]$$

But we may also write this expression in the form

$$\mu \dagger [d \mapsto \{p\}],$$

a fact which may be checked by applying the definition given above for the override operator. Nevertheless, in system specification, clarity is essential and one ought to say what one means. Allowing the override operator to be a sort of 'catchall' may be dangerous.

To the initiate, all of the mathematical material presented thus far, will have seemed to be bewildering. Let us summarize. For convenience, we will use another simple example. Suppose that we have tables of information, τ_j, τ_k, in $X \to Y$ which associates information y with index x. A typical table might be that shown below:

θ	the empty table.
$\mathsf{dom}\,\tau$	the set of indices in the table.
$\mathsf{rng}\,\tau$	the set of information values in the table.
$\tau_j \sqcup \tau_k$	the merging of two disjoint tables.
$\tau_j \dagger \tau_k$	the update of one table by another.
$\lhd [\![A]\!]\tau$	a table with all information associated with the indices in A removed.

In the example given, the index is a particular VDM♣ expression and the associated information is the interpretation which we wish to place on the expression.

One basic question surely springs to mind: Why choose these particular operators? Are there any others of significance? Before proceeding with a further analysis of our simple model, let us turn to the mathematical algebraic theory that provides the setting for all of the work.

2.1 Monoids

Everyone is familiar with the natural[13] numbers

$$\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$$

and that if one adds any two natural numbers m and n, then the result is a natural number. This basic fact is called the **law of closure:** for all m, n in $\mathbb{N}$, then

$$m + n \in \mathbb{N}$$

The zero, 0, is a *unique* natural number (with respect to addition) in the sense that it is the only number which when added to any other natural number n, gives the same natural number n. We say that zero is the (unique) **identity element** for addition:

$$n + 0 = n = 0 + n$$

We may, for convenience, speak of this fact as the **law of the identity**. Finally, we are familiar with the simple fact that the order in which we do additions of more than two natural numbers does not matter to the overall result. More formally, we say that for all m, n, and p in $\mathbb{N}$,

$$(m + n) + p = m + (n + p)$$

This is the **associative law**. Any set M, furnished with a binary operator, $\oplus$, and which satisfies the three laws given above is called a monoid, denoted $(M, \oplus, u)$, where $u \in M$ is the (unique) identity element for the operator $\oplus$. Thus the set of natural numbers under addition, $(\mathbb{N}, +, 0)$, is a monoid.

If, in addition to the three laws cited, we have the **commutative law**, i.e., for all m, n in M,

$$m \oplus n = n \oplus m$$

then we have a commutative monoid. The set of natural numbers under addition, $(\mathbb{N}, +, 0)$, is a commutative monoid. It just so happens that the monoid is fundamental to computer science. It turns up everywhere. Before we return to consider the doctor-patient model, it is important to note that the 'same' set may give rise to different monoids with respect to different operators. For example, if we remove the zero from the natural numbers, $\lhd [\![0]\!]\mathbb{N}$, we obtain the set of counting numbers

$$\{1, 2, 3, 4, 5, \dots\}$$

which we may denote, for convenience, by $\mathbb{N}'$, or $\mathbb{N}_1$. The set of counting numbers forms a commutative monoid, $(\mathbb{N}', \times, 1)$, under multiplication.

Words over some alphabet, such as the English alphabet, also form a monoid[14] under word catenation. For us a word is just a sequence of letters. Thus, not

[13] Of course, one might ask why they are called natural? Indeed, one might wonder whether or not there are unnatural numbers.

[14] For technical reasons, this monoid is known as the *free monoid*. To the casual observer, it seems just like another member of the family of monoids—a harmless everyday familiar type of monoid. What a surprise then is it for one to learn/discover that all of computer science depends upon it!

only are *dog* and *god* words, but so are *apg* and *zwt*. The catenation of *ma* and *dame* may be represented by $ma \frown dame$ or just simply by juxtaposition *madame*. The identity element for catenation is the empty word, which we denote by Λ.

If we denote the alphabet by Σ, then words over the alphabet are said to belong to a set which we denote by $\Sigma^\star$. The free monoid is then $(\Sigma^\star, \frown, \Lambda)$ and if the alphabet contains more than one letter, which is normally the case, then the monoid is not commutative.

Now let us consider our doctor-patient model in the abstract:

$$X \rightarrow \mathcal{P}Y$$

where X denotes *DOC* and Y denotes *PAT*. Then all the elements μ in $X \rightarrow \mathcal{P}Y$ forms a non-commutative monoid, $(X \rightarrow \mathcal{P}Y, \dagger, \theta)$, under the override operator, where θ denotes the null map.

Just as in the case of natural numbers, we may provide another operator for the underlying set of maps that gives us a commutative monoid. Specifically, let us consider

$$X \rightarrow \mathcal{P}'Y$$

where $\mathcal{P}'Y = \lhd [\![\emptyset]\!] \mathcal{P}Y$. Then let us define a special $\textcircled{\cup}$ operator which combines the extend operator and the override operator as follows:

$$\mu \textcircled{\cup} [d \mapsto \{p\}] \triangleq \begin{cases} \mu \sqcup [d \mapsto \{p\}], & \text{if } \neg\chi[\![d]\!]\mu \\ \mu \dagger [d \mapsto \mu(d) \cup \{p\}], & \text{otherwise} \end{cases}$$

Then $(X \rightarrow \mathcal{P}'Y, \textcircled{\cup}, \theta)$ is a commutative monoid where the operator $\textcircled{\cup}$ **inherits** its commutativity from that of set union $\cup$. Thus the registration of a doctor and her/his patient might be succinctly captured by

$$\mu \textcircled{\cup} [d \mapsto \{p\}]$$

This abstract monoid will prove to be of especially great significance in some of our later specifications.

2.2 Morphisms

Every engineer is surely familiar with the remarkable property of logarithms, embodied in the physical realization of the slide rule, that the log of a product is the sum of the logs, and that the log of 1 is 0.

$$\begin{aligned} \log(xy) &= \log(x) + \log(y) \\ \log(1) &= 0 \end{aligned}$$

Formally, we say that the log function is an **isomorphism** from the multiplicative group[15] of strictly positive real numbers, $(\mathbb{R}^+ \backslash \{0\}, \times, 1)$, to the additive group of real numbers, $(\mathbb{R}^+, +, 0)$. The morphism is called 'iso-' precisely because the two groups have exactly the same structure but with different operators. In addition to exhibiting the mathematical elegance and æsthetics of the

[15] A group, $(G, \oplus, u)$, is a monoid with the additional law that every element $g \in G$

isomorphism of the two groups, the example also demonstrates unequivocally how eminently practical[16] such isomorphic structures can be. Moreover, it is the function which is most important.

One question, therefore, immediately springs to mind. What are the isomorphic monoids? Can we find some simple examples? Let us start with natural numbers under addition, $(\mathbb{N}, +, 0)$, and natural numbers (without the zero) under multiplication, $(\mathbb{N}', \times, 1)$. Are they isomorphic? How can we show whether or not they are? The question seems very reasonable since we might suppose that this problem is just a simpler version of the isomorphism of the reals illustrated above.

Let us suppose that there is indeed a sort of log function which we shall call f. Then, for all m, n in $\mathbb{N}'$,

$$\begin{aligned} f(m \times n) &= f(m) + f(n) \\ f(1) &= 0 \end{aligned}$$

Have we not solved the problem?

Unfortunately, although the pattern looks right, we have not supplied the mechanism[17]. We must say exactly what f is. This is a good example of where experimental mathematics (Epstein and Levy 1995) might fit in. At any rate the problem is simple enough to be tackled by pencil and paper. Our starting point is already determined for us:

$$f(1) = 0$$

Furthermore, it seems very reasonable to make the assignment

$$f(2) = 1$$

From this we construct the table of powers of two:

$$\begin{aligned} f(2^2) &= f(2 \times 2) = f(2) + f(2) = 2f(2) = 2 \\ f(2^3) &= f(2 \times 2 \times 2) = f(2) + f(2) + f(2) = 3f(2) = 3 \\ &\ldots \end{aligned}$$

---

has a unique inverse $\bar{g} \in G$ such that

$$g \oplus \bar{g} = u = \bar{g} \oplus g$$

Monoids can be converted into groups by supplying appropriate inverse elements and making some adjustments to the basic operator where necessary. Of special interest is the free goup, obtained from the free monoid by adding in an alphabet of inverse letters, denoted $\bar{\Sigma}$. When a word ω and its inverse $\bar{\omega}$ are catenated, annihilation results, giving the null word, Λ.

We hypothesize that the free group underlies all communication.

[16]Naturally, one consequently suspects that there might be isomorphisms of no practical use whatsoever. But, then, how can one tell in advance whether a given isomorphic pair will be of practical use/benefit or not?

[17]It is said that any one can have a theory. To be acceptable there must also be an accompanying mechanism. For example, I might have the theory that the moon is made of green cheese. However, unless I can show how it fits in with our current theories of cosmology, I am unlikely to be believed.

Clearly our mechanism, i.e., the given f, will not work. Nevertheless the result is very interesting[18].

Consider the free monoid over the alphabet with one letter $\Sigma = \{|\}$. Such a letter is sometimes called a tally mark. Then catenation of words is simple and obvious: $|||| \frown ||| = |||||||$, for example. Clearly, such a free monoid is isomorphic the monoid of natural numbers under additon, denoted

$$(\Sigma^\star, \frown, \Lambda) \cong (\mathbb{N}, +, 0)$$

and the required function is a length function *len* which gives us the length of the tally string. For example, $len\colon ||| \mapsto 3$. Formally,

$$\begin{aligned} len(\sigma \frown \tau) &= len\,\sigma + len\,\tau \\ len(\Lambda) &= 0 \end{aligned}$$

Further experimentation with free monoids and the length function quickly leads us to conclude that it is only the free monoid over a one-letter alphabet that gives us an isomorphism under length. For example, for the two-letter alphabet $\{a, b\}$, we have

$$len(aaba) = len(abaa) = 4$$

We may view this as the result of a sort of *glueing* process where *aaba* and *abaa* get glued together and become labelled as 4. Indeed, all words of length 4 get glued together. In this case, and consequently in general, we say that *len* is a **homomorphism**[19] from the free monoid to the monoid of natural numbers under addition.

Our next example we take from the application in hand. Let μ_j, μ_k in $DOC \to \mathcal{P}PAT$ be two collections of doctors and their patients such that μ_k is to be interpreted as a 'batch update' of μ_j, denoted $\mu_j \dagger \mu_k$. Then the operator dom which returns the set of doctors in the collection is a homomorphism from the monoid of maps, $(DOC \to \mathcal{P}PAT, \dagger, \theta)$, to the monoid of sets, $(\mathcal{P}DOC, \cup, \emptyset)$.

$$\begin{aligned} \mathsf{dom}(\mu_j \dagger \mu_k) &= \mathsf{dom}(\mu_j) \cup \mathsf{dom}(\mu_k) \\ \mathsf{dom}(\theta) &= \emptyset \end{aligned}$$

The domain operator is **not** an isomorphism. We may demonstrate this fact by showing that two distinct maps *glue* to the same set. A simple example is

$$\begin{aligned} \mathsf{dom}([d \mapsto S] \dagger [d \mapsto T]) &= \mathsf{dom}([d \mapsto S]) \cup \mathsf{dom}([d \mapsto T]) \\ &= \{d\} \cup \{d\} \\ &= \{d\} \end{aligned}$$

Both $\mathsf{dom}([d \mapsto S])$ and $\mathsf{dom}([d \mapsto T])$ map to the same result, $\{d\}$.

[18] Why is the result interesting? What exactly is the result anyway?

[19] For further enlightening remarks see Jacobson (1985, 58 et seq.). The terminology of morphisms can be quite intimidating to the non-mathematician. If the homomorphism is from the monoid to itself then we call it an **endomorphism** and if it is an isomorphism then we call it an **automorphism**. Endomorphisms play a very significant role in computer science.

Although map extension, $\sqcup$, does not give a monoid, it is a specialization of map override, $\dagger$, and, therefore, we also have

$$\mathsf{dom}(\mu_j \sqcup \mu_k) = \mathsf{dom}(\mu_j) \cup \mathsf{dom}(\mu_k)$$

for **disjoint** maps, μ_j and μ_k, i.e., where $\mathsf{dom}\,\mu_j \cap \mu_k = \emptyset$. This is an extremely important result. Although we do not have a monoid homomorphism in this particular case, the effect is the same.

Unfortunately, the range operator on maps, rng, is not a monoid homomorphism:

$$\begin{array}{rcl} \mathsf{rng}(\mu_j \dagger \mu_k) & \subseteq & \mathsf{rng}(\mu_j) \cup \mathsf{rng}(\mu_k) \\ \mathsf{rng}(\theta) & = & \emptyset \end{array}$$

We may demonstrate[20] inequality with the same example we used above to illustrate that the domain operator was not an isomorphism.

$$\begin{array}{rcl} \mathsf{rng}([d \mapsto S] \dagger [d \mapsto T]) & = & \mathsf{rng}([d \mapsto T]) \\ & = & \{T\} \\ & \neq & \{S, T\} \\ & = & \{S\} \cup \{T\} \\ & = & \mathsf{rng}([d \mapsto S]) \cup \mathsf{rng}([d \mapsto T]) \end{array}$$

However, we do have the important result for disjoint maps:

$$\mathsf{rng}(\mu_j \sqcup \mu_k) = \mathsf{rng}(\mu_j) \cup \mathsf{rng}(\mu_k)$$

Therefore, from the definition of the override operator in terms of the extend operator, we also have the equality

$$\begin{array}{rcl} \mathsf{rng}(\mu_j \dagger \mu_j) & = & \mathsf{rng}(\lhd\!\!\!\lhd [\![\, \mathsf{dom}\, \mu_k]\!] \mu_j \sqcup \mu_k) \\ & = & \mathsf{rng}(\lhd\!\!\!\lhd [\![\, \mathsf{dom}\, \mu_k]\!] \mu_j) \cup \mathsf{rng}(\mu_k) \end{array}$$

This result is essential for the execution/construction of proofs. Before rejoining our model development there is yet another important theoretical aspect that needs to be covered, one that addresses the meaning of the removal or deletion operator.

2.3 Monoids with Operators

In considering the natural numbers, we identified two well-known monoids: the additive monoid, $(\mathbb{N}, +, 0)$, and the multiplicative monoid, $(\mathbb{N}', \times, 1)$. We wish now to focus on the monoid $(\mathbb{N}, +, 0)$ and ask in what manner the operation of multiplication may be 'integrated' into it. It comes as no surprise to learn that we are here speaking about **the distributive law**, the distribution of

[20]The manner in which the demonstration is *presented* uses a technical device whereby one combines two developments into one. This is not normally the order in which the demonstration would be arrived at initially.

multiplication over addition. Specifically, for all m, n in $\mathbb{N}$ and for some k in $\mathbb{N}'$, we usually write the law in the form

$$k(m+n) = km + kn$$

Let us now expressly exhibit the multiplication operator $\times$ which was denoted by juxtaposition above.

$$k \times (m+n) = k \times m + k \times n$$

Finally, being good[21] computer scientists, we will replace the infix $\times$ with a prefix functional name, say M, and wrap up the multiplier k in brackets:

$$\begin{aligned} M[\![k]\!](m+n) &= M[\![k]\!](m) + M[\![k]\!](n) \\ M[\![k]\!](0) &= 0 \end{aligned}$$

Well, what a pleasant surprise! The expression, $M[\![k]\!]$, which we will call an operator, behaves exactly like a monoid morphism. Specifically, it maps the monoid $(\mathbb{N}, +, 0)$ to itself. For this reason, we call it an **endomorphism** of $(\mathbb{N}, +, 0)$. Now we might ask what sort of structure operators of the form $M[\![k]\!]$ might have. Specifically, is there a binary operator that turns the set of all such operators into a monoid? The answer is affirmative. Denoting the binary operator by $\circ$, which usually denotes functional composition, we immediately have

$$\begin{aligned} (M[\![j]\!] \circ M[\![k]\!])n &= M[\![j]\!](M[\![k]\!]n) \\ &= M[\![j]\!](k \times n) \\ &= j \times (k \times n) \\ &= (j \times k) \times n \\ &= M[\![j \times k]\!]n \end{aligned}$$

Consequently, we may define the law of composition of the functional operators $M[\![j]\!]$ and $M[\![k]\!]$, for all j, k in $\mathbb{N}'$ as

$$M[\![j]\!] \circ M[\![k]\!] = M[\![j \times k]\!]$$

Therefore, the set of all functional operators is closed under this functional composition. It is rather easy to verify that the associative law holds. Finally, $M[\![1]\!]$ is the unique identity. We have just established that the set of functional operators is a monoid under functional composition as defined above. Moreover, the association

$$M[\![k]\!] \leftrightarrow k$$

[21]This particular evocative phrase, *good computer scientist*, is intended to convey the notion that there are certain ways of thinking and looking at things that one must acquire before one might be considered a computer scientist. One such way is exhibited here—the interchangeability of infix operators and curried functions.

Further, one might be a good computer scientist without knowing it! In other words, one might have acquired the above way of looking at operators but never actually have stumbled across the term 'currying'.

demonstrates unequivocally that this monoid is isomorphic[22] to $(\mathbb{N}', \times, 1)$.

With this background information, we are ready to show that map removal with respect to a set is a monoid endomorphism. Recall that

$$\lhd [\![d]\!]\mu$$

denotes the collection, $\mu \in DOC \rightarrow \mathcal{P}PAT$, of doctors and their patients with information on doctor d removed/deleted. Let $S \in \mathcal{P}DOC$ denote a set of doctors. Then we may show that

$$\begin{aligned} \lhd [\![S]\!](\mu_j \dagger \mu_k) &= (\lhd [\![S]\!]\mu_j) \dagger (\lhd [\![S]\!]\mu_k) \\ \lhd [\![S]\!]\theta &= \theta \end{aligned}$$

and, of course, that this endomorphism applies to the specialized extend of two disjoint maps

$$\lhd [\![S]\!](\mu_j \sqcup \mu_k) = (\lhd [\![S]\!]\mu_j) \sqcup (\lhd [\![S]\!]\mu_k)$$

The collection of such removal operators, $\lhd [\![S]\!]$, $S \in \mathcal{P}DOC$, forms a monoid under functional composition defined by

$$\lhd [\![S]\!] \circ \lhd [\![T]\!] = \lhd [\![S \cup T]\!]$$

The functional operator, $\lhd [\![\emptyset]\!]$, is the unique identity element and the associate law holds for the composition. Setting up the formal association

$$\lhd [\![S]\!] \leftrightarrow S$$

demonstrates that the monoid of removal operators is isomorphic to the monoid of sets of doctors $(\mathcal{P}DOC, \cup, \emptyset)$.

This concludes our very brief exposition of some of the algebraic theory underlying the formal method that we are using in this tutorial. Other theoretical results necessary in the context of the application being developed will be presented in their proper place.

Space prohibits inclusion of other significant foundational material here. Further details may be found in the author's doctoral thesis (Mac an Airchinnigh 1990) and in a comprehensive tutorial presented in the Netherlands (Mac an Airchinnigh 1991).

2.4 Complementary Models

Let us look at the relationship between our model and reality. We ask ourselves what interpretation is to be given to the model, in what way can it be said to be an abstraction of some reality. As computer scientists we will also naturally ask ourselves how we will implement the model. Does it involve hardware or software? Where will we draw the system boundary that delineates the implementable thing from the purely conceptual thing. The model

$$\mu \in DOC \rightarrow \mathcal{P}PAT$$

[22]Such a monoid of operators is also known as a monoid of transformations. This result may be generalized to *Cayley's Theorem*: "*Any monoid is isomorphic to a monoid of transformations*" (Jacobson 1985, 38).

says something about a collection of doctors and their patients. Now it is quite clear that we can not put these doctors and patients into a computer. What goes into a computer is basically encoded information. What information might we associate with our doctors and patients?

Example 3 (Doctors and Names) *Every doctor d has a name. Thus, we introduce the space of doctors and their names.*

$$\nu \in DOC \rightarrow DOC\_nm$$

A typical doctor-name map will have the form

$$[d_1 \mapsto n_1, d_2 \mapsto n_2, \ldots, d_j \mapsto n_k]$$

In the real world, we know that there is some non-zero probability that two different doctors will have the same name. For instance, it is possible that in some ν we might find entries of the form

$$d_j \mapsto \text{Beizer}, \quad \text{and} \quad d_k \mapsto \text{Beizer}$$

For this reason, names are considered to consist of surname, firstname and middle initials. But even here, chance may have it, that two distinct doctors will have the same (extended) names. Computer scientists overcome this particular problem by insisting on giving people **identifiers**. Therefore, let us introduce this new model

$$\alpha \in DOC \rightarrow DOC\_id$$

Technically, this association is an exact analogue of the notion of coordinates in geometry. Formally, we make a distinction between points and their coordinates. If we let E^3 denote the 3-dimensional euclidean space of points, then we may choose to assign an orthonormal cartesian coordinate frame and associate with each point a triple of real numbers taken from $\mathbb{R}^3$. Such an assignment of cartesian coordinates may be modelled by the map

$$\varrho \in E^3 \rightarrow \mathbb{R}^3$$

A particular point p in ϱ will have coordinates (x_p^1, x_p^2, x_p^3). Moreover, distinct points will have distinct coordinates, i.e., the map ϱ will be 1–1. We may express this fact formally by stating that the number of domain elements, points, is exactly the same as the number of range elements, cartesian coordinates:

$$|\,\mathsf{dom}\,\varrho| = |\,\mathsf{rng}\,\varrho|$$

On the other hand, to the same point we may assign, say, spherical coordinates, triples of numbers, (r_p, ϕ_p, θ_p), where r_p, ϕ_p, θ_p in $\mathbb{R}$, and $0 \leq r_p$, $0 \leq \phi_p < 2\pi$, $0 \leq \theta_p \leq \pi$. Such an assignment may also be modelled by

$$\sigma \in E^3 \rightarrow \mathbb{R}^3$$

Unlike the cartesian coordinate map, the spherical coordinate map is not 1–1. The problem arises at the origin point of the coordinate space. Specifically, when $r_p = 0$, we can not distinguish between $(r_p, 0, 0)$ and $(r_p, \pi/2, \pi)$, say.

Is it not surprising, therefore, that coordinate functions should be complex in computer science?

We can certainly put identifiers into a computer. But what do we do about the doctor part of the association? Doctors will not go into computers. Consider our two models

$$\begin{aligned} \nu &\in DOC \rightarrow DOC\_nm \\ \alpha &\in DOC \rightarrow DOC\_id \end{aligned}$$

Since the latter is 1–1, we may **invert** it to obtain

$$\alpha^{-1} \in (DOC \rightarrow DOC\_id)^{-1} = DOC\_id \rightarrow DOC$$

Composition of the inverted model and the doctor-name model gives us the basic information model

$$\begin{aligned} \nu \circ \alpha^{-1} = \delta &\in DOC\_id \rightarrow DOC\_nm \\ &= (DOC \rightarrow DOC\_nm) \circ (DOC\_id \rightarrow DOC) \end{aligned}$$

where the composition $\nu \circ \alpha^{-1}$ is defined over the common intersection of the domains of ν and α^{-1}. Now we have arrived at a model that may be used in an implementation. The unique identifier represents the doctor and the name represents an attribute of the doctor, her/his name.

Technical Aside: The notion of inverse images is a key one in mathematics, and consequently, in the VDM♣. It plays a critical rôle in classical mathematics. It is not surprising, therefore, that in a model-theoretic formal method, such as the VDM♣, it should also prove to be of critical importance. For example, it was by the use of the inverse image that the annihilator form of the invariant for the bill of materials problem was discovered (Mac an Airchinnigh 1991, 201), a major result that led to the tractability of proving that operations on a bill of material satisfied the invariant. A second major result was the discovery that the inverse image is just the appropriate construct to identify *aliases* in a system specification (Mac an Airchinnigh 1991, 182). Again, in determining the *standard development* steps of the VDM♣, the inverse image proved essential in the parametrization step (Mac an Airchinnigh 1991, 189). That the inverse image should prove to be of considerable use is sufficient reason for us to study its properties at length.

Since the inverse image will re-occur frequently in this tutorial, it seem appropriate to mention briefly some of its uses in classical mathematics. We begin with a definition from classical algebra.

Every map μ from a set X to a set Y,

$$\mu \in X \rightarrow Y$$

determines an equivalence relation E_μ in X, (Jacobson 1985, 12), by specifying

$$x_j E_\mu x_k \quad \text{if and only if} \quad \mu(x_j) = \mu(x_k)$$

The inverse image $\mu^{-1}(y)$ of the element $y \in Y$, is the set of all elements $x \in X$ which map under μ to y, formally

$$\mu^{-1}(y) = \{x \in X \mid \mu(x) = y\}$$

If $y = \mu(x)$ then the set $\mu^{-1}(y)$ is just the equivalence class $\bar{x}_{E_\mu} = \mu^{-1}(\mu(x))$ in X determined by the element x, which subset of X is sometimes referred to as the "fibre over the element" y (Jacobson 1985, 12).

The set of all such fibres constitutes a partition of X determined by E_μ, (Jacobson 1985, 12), i.e., they are elements of the quotient set X/E_μ, which in turn is a subset of $\mathcal{P}(X)$.

If we let ν denote the natural map from X to X/E_μ, then there is a map $\bar{\mu}$ from X/E_μ to Y which is induced by μ, such that we have the factorization

$$\mu = \bar{\mu} \circ \nu$$

It is, of course, quite natural to generalize the inverse image to apply to a subset S of Y rather than a simple element. Specifically,

$$\mu^{-1}(S) = \{x \in X \mid \mu(x) \in S\}$$

In analysis, the concept of inverse image plays a fundamental definitional rôle. For example, a function f from a measurable space X to a topological space Y,

$$f \in X \to Y$$

is defined to be measurable, (Rudin 1970, 8), provided that $f^{-1}(S)$ is a measurable set for every open set S in Y.

Such examples of the use of the inverse image in mathematics are intended solely to indicate its relevance therein. Due to the underlying philosophy of the VDM♣, (Mac an Airchinnigh 1990), it behoves us to explore the range of its use in the specification of systems and any properties that inverse images might have □

For those *au fait* with (relational) databases, it will be recognized that the uniqueness of the identifier is spelt out more precisely by adding extra information such as social security number, address, phone number, etc.

$$\delta' \in \mathit{DOC\_id} \to (\mathit{DOC\_nm} \times \text{SOC\_SEC\_NUM} \times \mathit{ADDRESS} \times \text{PHONE\_NUM} \times \dots)$$

We may, if we so wish, extract the original identifier-name structure from this model by iterating over δ' with a pair of functions, the identity function, I, and a projection function, π_1, that selects the first component of the attribute tuple. Note that the projection function is not to be confused with the number that denotes the area of a unit disc, π! We denote this iteration by

$$(\mathrm{I} \to \pi_1)\delta' = \delta$$

All of these models may be manipulated by map operators that were introduced in the context of the doctor-patient collection: domain and range operators, map extend and map override, and removal with respect to a set. For each operator expression we can find a meaningful interpretation pertinent to the problem at hand and **most importantly** conditions of validity—pre-conditions—guarding each operator. *Is it really necessary to state that the pre-conditions are precisely those things which need to be tested in an implementation?* There are other aspects of models which give rise to the determination of critical examination criteria for validity.

Before delving deeper into the application at hand, we would now like to introduce another model which is complementary to the basic doctor-patient model.

Example 4 (Patient-Doctor Relation) *From the perspective of the patient p, we may wish to record all the doctors that the patient consults, the eye doctor, the ear doctor, the cardiologist, etc. Such a patient-doctor relation may be denoted by*

$$[p \mapsto \{d_1, d_2, \ \ldots \ , d_m\}]$$

In other words we wish to look at the space

$$\mu^{-1} \in PAT \ \rightarrow \ \mathcal{P}DOC$$

This model is the inverse of the doctor-patient relation and, consequently, we use μ^{-1} to denote a typical instance. It is not the sort of model that one would normally wish to implement directly (via the coordination technique presented) because the map is conceptually accessible only to a **homunculus**, an external (global) observer. Such a situation occurs frequently in formal modelling and is of great usefulness in system specification. Some questions that we might ask of μ may be answered directly by reference to this model. Specifically, if we ask who are the doctors that some patient p consults, then the result is immediate

$$\mu^{-1}(p)$$

where we have, of course, the necessary pre-condition $\chi[\![p]\!]\mu^{-1}$. However, many software engineers may feel uncomfortable about introducing such a complementary model, that somehow we are cheating. Therefore, let us provide an explicit inversion algorithm that converts a given μ into the corresponding μ^{-1}. This will give us an opportunity to introduce the style of algorithm that we normally use in the VDM♣, which in this case is tail-recursive. Those familiar with programming in a logic programming language such as PROLOG or, especially, a functional programming language such as MIRANDA, will have no difficulty in following the reasoning behind the algorithm. Space prohibits us from giving a comprehensive treatment of the theory of tail-recursive functions, or of the more general theory of curried functions. Supplementary material may be found in (Mac an Airchinnigh 1990) and (Mac an Airchinnigh 1991). Suffice it to remark that tail-recursive functions have exact while-loop equivalents in imperative programming languages and the strategies for the transformation of a tail-recursive function into the corresponding while-loop are so well-developed that complete automation is often possible.

The inversion algorithm follows:

4 Invert: $(DOC \ \rightarrow \ \mathcal{P}'PAT) \longrightarrow (PAT \ \rightarrow \ \mathcal{P}'DOC)$

.1 $\text{Invert}(\mu) \triangleq \text{Invert}[\![\mu]\!]\theta$

5 Invert: $(DOC \ \rightarrow \ \mathcal{P}'PAT) \longrightarrow$
$((PAT \ \rightarrow \ \mathcal{P}'DOC) \longrightarrow (PAT \ \rightarrow \ \mathcal{P}'DOC))$

.1 $\text{Invert}[\![\theta]\!]\nu \triangleq \nu$

.2 $\text{Invert}[\![[d \mapsto S] \sqcup \mu]\!]\nu \triangleq \text{Invert}[\![\mu]\!] \circ \text{Invert}[\![d, S]\!]\nu$

6 Invert: $DOC \times \mathcal{P}PAT \longrightarrow$
$((PAT \rightarrow \mathcal{P}'DOC) \longrightarrow (PAT \rightarrow \mathcal{P}'DOC))$

.1 $\text{Invert}[\![d, \emptyset]\!]\nu \triangleq \nu$

.2 $\text{Invert}[\![d, \{p\} \uplus S]\!]\nu \triangleq \text{Invert}[\![d, S]\!](\nu \Cup [p \mapsto \{d\}])$

Annotations

4 Note that the null set is excluded from the maps. This is a critical point. It is not possible to invert an entry such as $d \mapsto \emptyset$. A similar remark applies to the signatures in **5** and **6**.

4.1 The computation of the inversion function is handed over to another function of the same name. This latter function is the tail-recursive function whose signature (**5**) differs from the function to be called. The argument outside the parenthesis (which appears as ν elsewhere) is often known as an accumulator variable and is initialized to the null map. Entry to the tail-recursive function is equivalent to entry to a while-loop.

Note the overloading of the function name. The two distinct functions may be determined by signature, i.e., by the number of parameters and their type. This facility is availed of by PROLOG and ADA, for example.

This inversion function is a monoid homomorphism:

$$\begin{aligned} \text{Invert}(\mu_j \Cup \mu_k) &= \text{Invert}(\mu_j) \Cup \text{Invert}(\mu_k) \\ \text{Invert}(\theta) &= \theta \end{aligned}$$

5.1 The tail-recursive function is a functional operator. In this, the base case, we have termination (exit from the while-loop) and $\text{Invert}[\![\theta]\!]$ is clearly the identity function.

5.2 We take advantage of the definition of the extend operator on maps to exhibit a non-null map explicitly as the extend of two disjoint maps. In this, the recursive case, we introduce a second tail-recursive function that will iterate over the set of patients.

The second invert function gives us a monoid of functional operators:

$$\begin{aligned} \text{Invert}[\![\mu_j \Cup \mu_k]\!]\nu &= \text{Invert}[\![\mu_k]\!] \circ \text{Invert}[\![\mu_j]\!]\nu \\ \text{Invert}[\![\theta]\!]\nu &= \nu \end{aligned}$$

6.1 Clearly, $\text{Invert}[\![d, \emptyset]\!]$ is the identity function. However, its arguments are formally equivalent to an entry of the form $d \mapsto \emptyset$ which we have ruled out. There is, of course, no inconsistency. We have taken care of that problem by arranging the signature (**6**) appropriately.

6.2 Since ordinary set union, $\cup$, does not allow us to express the disjoint union of two sets and thus to exhibit a non-empty set in two disjoint parts, we have recourse to a special union operator, $\uplus$, which functions as such a disjoint union. This recursive case is where some actual modification of the accumulator variable take place. Note that the result is elegantly

expressed in terms of the operator that gives us a monoid of patient-doctor relations.

Yet again do we have a monoid of functional operators. For convenience, we will use a slightly modified notational form:

$$\begin{aligned} \text{Invert}_d[\![S \cup T]\!]\nu &= \text{Invert}_d[\![T]\!] \circ \text{Invert}_d[\![S]\!]\nu \\ \text{Invert}_d[\![\emptyset]\!]\nu &= \nu \end{aligned}$$

The algorithm is proven correct by construction! That is to say, in the VDM♣, proofs are constructive. This philosophical position is poorly understood by the practitioners of other formal methods. In a sense it is the ideal that every programmer strives for. However, just as in programming, it is quite possible that the strong claim made about correctness, is in fact false. How is one to know? Therein lies the social aspect of mathematical proof. It is certainly not primarily an issue of **testing** the specification. Rather we emphasize that the specification is presented before others and the arguments for correctness are presented.

However, every expression in the VDM♣ is constructive and, consequently, executable in practice. Hence, a prototype of the specifications written in, say MIRANDA, is readily feasible. System specifications grow in complexity but at a much lower rate than system code. There are other formal methods with tool support which will happily accommodate the VDM♣. In this manner at least the syntactic complexity may be kept under control.

This completes our introductory treatment of models and their operators.

Chapter 3

The Lab and the Disease Centre

There is a well-know aphorism that "*familiarity breeds contempt*". Such aphoristic utterances convey kernels of wisdom that need to be elaborated upon to ascertain their precise meaning. We all know the counter-aphorism that "*absence makes the heart grow fonder*", a saying the meaning of which is usually immediately clear to those who return to the home hearth after prolonged visits to conferences and symposia. But what of the aphorism originally quoted? By what mechanism does familiarity breed contempt?

One reasonable theory that explains the saying has to do with *enlightenment*. In other words, the more familiar you are with the other, the more you get to know the other, warts and all. It is the very same with formal methods. Therefore, to reach the ultimate stage of perfect contempt, we first must become familiar and strive for enlightenment. In this tutorial the path to enlightenment is set out clearly within the framework of doctors, patients, hospital laboratories, blood tests and a centre for disease control. To this fascinating world we now return.

However, having elaborated in depth in Chapter 2 on the style of development by which we arrive at the mathematical expressions which are the answers to our questions, we feel free hereafter to be more brief in our exposition in order to focus more on the application at hand. We trust that one has already sufficient information and insight that she/he will be able to expand on a given expression, where necessary. Again, for further assistance, one may consult the material in (Mac an Airchinnigh 1990) and (Mac an Airchinnigh 1991).

A hospital laboratory carries out blood tests for the patients of doctors who are registered with the hospital. The recorded results of the blood tests may be modelled by the space

$$\varrho \in PAT \rightarrow B\_TEST$$

The hospital register which lists affiliated doctors and their patients is modelled by

$$\mu \in DOC \rightarrow \mathcal{P}PAT$$

This model we have already seen before. Now it may be the case that we wish to ensure that the patients mentioned in ϱ are precisely the same patients listed in the register μ. To establish such a constraint we use an **invariant**.

$$\begin{array}{ll} 7 & inv : (PAT \rightarrow B\_TEST) \times (DOC \rightarrow \mathcal{P}PAT) \longrightarrow \mathbb{B} \\ .1 & inv(\mu, \varrho) \triangleq \mathsf{dom}\, \varrho = {}^{\cup}/\, \mathsf{rng}\, \mu \end{array}$$

Instead of modelling the recorded information in this manner, we might prefer an alternative model:

$$\delta \in DOC \rightarrow (PAT \rightarrow B\_TEST)$$

One good reason for this alternative is simply that we have one model instead of two and the necessity for having an invariant of the form given above has been eliminated completely.

There is, as one would expect, a connection between the two approaches, a connection which can be captured formally. From the single model δ we may extract the model of patient-blood test information. If we **assume** that all of the patient-blood test maps in δ are mutually disjoint then we may use the following expression

$$\varrho = {}^{\sqcup}/\, \mathsf{rng}\, \delta$$

Clearly the very validity of our solution depends completely on the validity of the application of reduction with respect to map extend, $\sqcup$, and this operator may validly be used iff the patient-blood test maps in δ are mutually disjoint. What a very strong assumption! Earlier in the tutorial we were very liberal (and realistic) in allowing a person to be the patient of more than one doctor. If we wish to be true to that liberalism (and realism) then we certainly can not use the extend operator. Let us explore the problem in some detail, i.e., debug the specification models.

Suppose that we have a δ of the form

$$\delta = [d \mapsto [p \mapsto t_1], d' \mapsto [p \mapsto t_2]]$$

which records the fact that two different doctors have the same patient. Application of the range operator gives

$$\mathsf{rng}\, \delta = \{[p \mapsto t_1], [p \mapsto t_2]\}$$

The inappropriateness of the extend operator is all too apparent. But if we *were* able to find a function that computes the corresponding μ, what would the result look like? Clearly, the only possible solution ought to be of the form

$$[p \mapsto \{t_1, t_2\}]$$

which is an element of the space

$$PAT \rightarrow \mathcal{P}B\_TEST$$

We seem to have got off to a very bad start. However, taking this seemingly negative result and turning it around into a positive one, allows us to illustrate most clearly how we employ such functions to expose certain types of inadequacies

in the formal models. Such functions are customarily called **retrieve** functions and they feature prominently in the verification process of formal modelling. How happy we must feel to have caught such a pernicious bug at this early modelling stage—the requirements modelling/specification stage—long before we even entered into considerations of design and ultimate implementation.

We will have other reasons to discard our patient-blood test model later. Apart from the detected bug, there is another potential subtle one embedded in the revised model of patients and blood tests:

$$PAT \rightarrow \mathcal{P}B\_TEST$$

Its discovery will be left as an exercise. For the present we adopt the irrealistic position that everything is in order and abide by the decision to insist on the assumption of mutually disjoint patient-blood test maps. Now we will turn to the other retrieve function by which ϱ may be extracted from δ, keeping our fingers crossed lest we uncover another difficulty. Happily, everything is in order:

$$\varrho = (\mathrm{I} \rightarrow \mathsf{dom})\delta$$

Having noted the uncovered problem for later resolution, we turn our attention to the centre for communicable diseases.

By law, each hospital laboratory is required to notify the national centre for disease control of all blood test results. However, for the purposes of confidentiality, the centre is prohibited from obtaining the names of the corresponding patients. On the other hand, the centre must be able to notify any registered doctor on the basis of a blood test result of one of his or her patients. We shall begin by supposing that the centre's information is modelled by

$$\begin{array}{lcl} \varsigma & \in & LAB \rightarrow \mathcal{P}DOC \\ \kappa & \in & DOC \rightarrow (B\_TEST \rightarrow \mathbb{N}'), \end{array}$$

and propose to investigate the operations: (i) lab notifies centre, (ii) centre notifies lab and doctor.

Again we have two models and, therefore, we look carefully to see if an invariant must be specified that constrains the models appropriately.

It is quite clear that the set of doctors occurring in both models must be the same and we may record this observation formally:

$$\cup/\,\mathsf{rng}\,\varsigma = \mathsf{dom}\,\kappa$$

The submodel

$$\beta \in B\_TEST \rightarrow \mathbb{N}'$$

represents a bag which is used to count the number of occurrences of a given test result. It is a monoid $(B\_TEST \rightarrow \mathbb{N}', \oplus, \theta)$ where the binary operator is given by

$$\beta \oplus [t \mapsto n] = \begin{cases} \beta \sqcup [t \mapsto n], & \text{if } \neg\chi[\![t]\!]\beta \\ \beta \dagger [t \mapsto \beta(t) + n], & \text{otherwise} \end{cases}$$

In other words, the operator $\oplus$ inherits its properties from the addition of natural numbers. We recognise the similarity of this construction with the definition of the $\textcircled{\cup}$ operator defined earlier. This leads us to formulate a general theorem:

Theorem 1 (Indexed Monoids) *Let $(M, *, u)$ denote an arbitrary monoid, which we shall call the* base *monoid, with unit u, and $(M', *)$ the corresponding semigroup, i.e., with $M' = \lhd [\![u]\!] M$. Then for a domain X, the structure $(X \to M', \circledast, \theta)$ is an indexed monoid which inherits its operator properties from $(M, *, u)$, where for μ in $X \to M'$,*

$$\mu \circledast [x \mapsto m] = \begin{cases} \mu \sqcup [x \mapsto m], & \textit{if } \neg \chi [\![x]\!] \mu \\ \mu \dagger [x \mapsto \mu(x) * m], & \textit{otherwise} \end{cases} .$$

For convenience, let us denote by

$$X \circlearrowleft (M, *, u)$$

the construction of the indexed monoid from the base monoid. With this notation, we may express the additive monoid of bags $(X \to \mathbb{N}', \oplus, \theta)$ as $X \circlearrowleft (\mathbb{N}, +, 0)$ and the monoid $(X \to \mathcal{P}'Y, \textcircled{\cup}, \theta)$ as $X \circlearrowleft (\mathcal{P}Y, \cup, \emptyset)$.

We might be tempted to break off the treatment of the disease centre model at this point and devote ourselves to a further elaboration of the theory that has been opened up by this new theorem. However, there is yet much to be done and we must forgo such a pleasure and return to the grindstone.

As in the case of the lab we may consider a single model for the disease centre information:

$$\psi \in LAB \to (DOC \to (B\_TEST \to \mathbb{N}'))$$

To obtain the first of our two models, ς, we write

$$\varsigma = (\mathrm{I} \to \mathsf{dom})\psi$$

To obtain the second model, κ, we proceed as in the case of the single model for the hospital lab and try a reduction with respect to map extend over the range of ψ:

$$\kappa = {}^{\sqcup}/\,\mathsf{rng}\,\psi$$

Once more this reduction is valid iff the doctor-blood test maps are mutually disjoint. We ask ourselves under what conditions such maps might not be disjoint? Clearly, in analogy to the hospital lab case, the specified retrieve function is invalid if we permit the same doctor to be registered with at least two distinct labs. Once more we have uncovered a pernicious bug of the **same class** as before. It may very well be the case that a consultant is registered with more than one hospital lab in the real world where the system is supposed to work. Whether this bug infects the two separate models ς and κ, is left as an exercise.

But, of course, one may argue that an experienced systems analyst would have uncovered such bugs in the requirements negotiations with the clients. Still it is comforting to know that they can be detected easily with formal mathematical procedures.

Let us proceed then with the models as given and use the assumption of one doctor–one lab. Our next assignment is the consideration of the two notify operations.

3.1 Lab notifies centre

We will use the pairs of models for the lab and the centre, each of which is repeated here for convenience.

$$\begin{aligned}\varrho &\in PAT \to B\_TEST\\ \mu &\in DOC \to \mathcal{P}PAT\\ \varsigma &\in LAB \to \mathcal{P}DOC\\ \kappa &\in DOC \to (B\_TEST \to \mathbb{N}')\end{aligned}$$

Upon determination of the results of a blood test t for the patient p of doctor d, the laboratory l notifies the centre. We expect the operation to have the form:

$$\text{Notify}[\![l, d, t]\!](\varrho, \mu, \varsigma, \kappa) \triangleq \ldots$$

We will certainly expect that the notify operation has a pre-condition which we shall now determine. First we must ensure that the blood test t was recorded at the lab l:

$$\chi[\![t]\!]\mathsf{rng}\ \varrho$$

But the curious will note that we did not make use of the lab's identifier l. It is certainly required by the centre but the domain LAB is not mentioned anywhere in models ϱ, μ. This anomaly must be rectified. It is precisely to remove such anomalies that one introduces a homunculus model for the hospital laboratory, such as

$$\mu' \in LAB \to (DOC \to \mathcal{P}PAT)$$

But now that we have identified the problem and its solution, we will proceed with the original models.

Continuing with the pre-condition analysis, we require that the doctor in question be registered with the lab

$$\chi[\![d]\!]\mu$$

and that the patient p in question is a patient of doctor d:

$$\varrho^{-1}(t) \cap \mu(d) \neq \emptyset$$

Our pre-condition thus far was concerned with the validity of data at the specific hospital lab. This involved checking the first pair of models. What about validity with respect to the models of the centre?

We immediately identify distinct cases. Suppose that the lab l is not registered at the centre and neither is the doctor. Then we would have

$$\text{Notify}[\![l, d, t]\!](\varrho, \mu, \varsigma, \kappa) \triangleq (\varrho, \mu, \varsigma \sqcup [l \mapsto \{d\}], \kappa \sqcup [d \mapsto [t \mapsto 1]])$$

where the use of the extend operator indicates that we are adding the conditions $\neg\chi[\![l]\!]\varsigma$ and $\neg\chi[\![d]\!]\kappa$.

Now suppose that the lab is not registered, but that the doctor is registered with some other lab. The very nature of the model pair allows us to capture the specification accurately:

$$\text{Notify}[\![l, d, t]\!](\varrho, \mu, \varsigma, \kappa) \triangleq (\varrho, \mu, \varsigma \sqcup [l \mapsto \{d\}], \kappa \dagger [d \mapsto \kappa(d) \oplus [t \mapsto 1]])$$

where the extra conditions are given by

$$\neg\chi[\![l]\!]\varsigma \wedge \chi[\![d]\!]\kappa$$

Or it may be the case that the lab is registered and the doctor is not ...

After an exhaustive analysis, it becomes quite clear that we need to bring some order to the variants of the notify operation. One obvious solution, which usually occurs to programmers, using conventional languages, is to distinguish the various cases by writing nested if-then-else statements. From experience we know that this route is fraught with difficulties and provides a golden opportunity for the independent testers.

For those who are encouraged to use modular programming lanuages or object-oriented languages, it will be apparent that the preferred route is to introduce a separate register operation. Thus, if a lab is not registered with the centre, it **must** do so before transmitting data. Such a solution would occur to programmers who are familiar with the model of cellular telephony, or with those engaged in advanced broadband communications services (whether multimedia or not). The register operation for the lab is simply

$$\text{Register}[\![l]\!](\varrho, \mu, \varsigma, \kappa) \triangleq (\varrho, \mu, \varsigma \sqcup [l \mapsto \emptyset], \kappa)$$

The occurrence of the empty set in the expression causes us to reflect whether or not one ought to register the lab's doctors D at the same time that one registers the lab. If we decide to go this route, then ς becomes modified to

$$\varsigma \sqcup [l \mapsto D]$$

and subsequent new affiliates of the lab may be registered as and when the need arises. But registering the doctors has an impact on the κ model and we must take into account whether or not the doctor to be registered for lab l is not, in fact, already registered with another lab l'. The resolution of this particular complication is left as an exercise, the solution of which will depend heavily on the degree of committment to the pairs of model specification or whether a different model is to be chosen.

3.2 Centre notifies lab and doctor

Having determined that the results of a particular blood test t are significant, the centre notifies all the labs and affiliated doctors of the result. The specification will have the form:

$$\text{Notify}[\![t]\!](\varsigma, \kappa) \triangleq \ldots$$

For this specification we will be content to identify the doctors concerned, and then from this information to identify the corresponding labs. One may check

the validity of the given significant blood test in a manner similar to the method used in the previous notify operation. Here we will concentrate on extracting the relevant information.

First, since the input is simply a blood test and no numerical information is supplied then we will construct a derived model:

$$\kappa' = (\mathrm{I} \rightarrow \mathsf{dom})\kappa \in DOC \rightarrow \mathcal{P}B\_TEST$$

Using the inversion algorithm given earlier, we may construct another derived model:

$$\kappa'' = \mathrm{Invert}(\kappa') \in B\_TEST \rightarrow \mathcal{P}DOC$$

recalling that the validity of the inversion depends on prohibiting entries of the form $d \mapsto \emptyset$, a situation which can only arise if a doctor is registered with the centre but there is no record of a blood test for any of her/his patients. Hence, we must make sure that registration of doctors does not leave the centre's database in a state which would cause the inversion to fail!

The map κ'' may now be used to give the set of all doctors who have to be notified:

$$((\mathrm{I} \rightarrow \mathsf{dom})\kappa)^{-1}(t)$$

where we have chosen to eliminate the auxillary variables κ', κ'' and replace the word invert with the appropriate symbol. If one finds this expression to be too abstract then the recommendations on verbosity given in § 2 of the tutorial may be applied.

From the lab-doctors model, we apply the inversion once again to obtain the derived model

$$\varsigma' = \mathrm{Invert}(\varsigma) \in DOC \rightarrow \mathcal{P}LAB$$

Once again, validity depends on the non-occurrence of an entry of the form $[l \mapsto \emptyset]$ in ς. In other words, registration of a lab, without further information, will put the centre's database into a state which will cause the inversion to fail.

We may now use the set of doctors to be notified to restrict this derived model to give both the set of doctors and the corresponding labs that both need to be notified. Resorting to κ'', this may be expressed in the form:

$$\lhd [\![\kappa''(t)]\!]\varsigma'$$

For the stout of heart, we express the same result in terms of the original variables and eliminate the invert operation name to give

$$\lhd [\![((\mathrm{I} \rightarrow \mathsf{dom})\kappa)^{-1}(t)]\!]\varsigma^{-1}$$

Finally, it must be remarked that the restriction operator presented in this solution is another monoid endomorphism which is the dual of the removal monoid endomorphism. Rather than give the details here, the reader is referred to (Mac an Airchinnigh 1990).

In specifying the notify operations we have tacitly assumed that communication is carried out directly from lab to centre and vice-versa in a simple abstract network that might be assumed to have a star configuration. We certainly did not enter into any discussion as to what the actual communications network might look like. To develop the application further we now look at a simple abstract model of such a network and use it as a vehicle to illustrate what proofs look like in the VDM♣.

Chapter 4

Proof

Earlier we made the statement that proofs are constructions in the VDM♣. In most other formal models proofs are based on the use of formal logic. To illustrate the distinction we choose a simple model of a communications network with switching nodes, give a simple invariant in the formal logic style and remark upon its properties. Then an alternative constructive form of the invariant is supplied. Finally, the power of the method is exhibited by considering satisfaction of the invariant by an operation on the network and **calculation** of the necessary and sufficient pre-condition by constructive proof. Those who advocate formal logic for proof are invited to do likewise in their method.

The material in this section was first presented by the author at a guest lecture in the Technical University of Delft, December 1992, and the material appears here in print for the first time. The results are believed to be new.

4.1 The Network Model

Let us consider a distributed system whose network topology is a ring. Using a graph representation, one may identify the nodes with processes and edges with communication channels (Tel 1991, 27). A graph may, in turn, be represented as a map μ in the space (Mac an Airchinnigh 1991, 221):

$$\mu \in X \ \rightarrow \ \mathcal{P}X$$

But not every μ will represent a ring. We know that the same space models directed acyclic graphs, which are, for example, representations of bills of material (Mac an Airchinnigh 1991, 218). To identify precisely that subspace which models rings we supply an invariant. For the present we illustrate the model of an n-node unidirectional ring by

$$\mu = \left[\begin{array}{ccc} x^1 & & \{x^2\} \\ x^2 & & \{x^3\} \\ \vdots & \mapsto & \vdots \\ x^{n-1} & & \{x^n\} \\ x^n & & \{x^1\} \end{array} \right]$$

In a general network topology, it must be the case that every node x is connected to at least one other node. Supposing that the edges are to represent

bidirectional channels, then

$$x \mapsto \{x^1, x^2, \ldots, x^j\}$$

would represent a typical node's connectivity. A more detailed analysis on the distinctions bewteen unidirectional and bidirectional links will be developed later when we make a distinction between a fixed network topology and the dynamic selection of routings based on sender-receiver pairs.

4.2 The Invariant

Whatever the intended network topology, it is clear that it is never the case that a node be self-connected, i.e., there are no 'direct' cycles in the network. Thus elements of the form

$$x \mapsto \{ \ldots, x, \ldots \}$$

are definitely to be prohibited. We may, therefore, use this information to constrain the space $X \to \mathcal{P}X$ by imposing the invariant:

$$inv(\mu) \stackrel{\triangle}{=} \forall x \in \mathsf{dom}\,\mu, x \notin \mu(x)$$

There may well be other constraints that one might impose. We shall however, content ourselves with an analysis of that given.

From the perspective of the logician, the given invariant is concise, accurate and elegant. But from the perspective of the formal methodist who insists on the application of **formal** logic, then the calculation we are about to perform, may cause certain technical difficulties—a challenge!

In the VDM$^{\clubsuit}$ we eschew writing universal and existential quantifiers. Thus are we impelled to seek an alternative constructive form of the invariant. One solution follows.

The general strategy is to take an element of the form

$$x \mapsto S,$$

tuple the domain and range to give

$$x \mapsto (x, S),$$

and iterate over the resulting range with the characteristic function to produce

$$x \mapsto \chi(x, S),$$

which is formally equivalent to

$$x \mapsto \chi[\![x]\!]S.$$

If x is not a member of S, then our construction will produce $x \mapsto 0$. We now have some choices on how to proceed to establish the required invariant. Before illustrating these, let us first present the construction that achieves the above result.

For each $\mu \in X \to \mathcal{P}X$ we introduce the corresponding 'identity map' I_μ which takes each element x in the domain of μ to itself:

$$\mathrm{I}_\mu \triangleq \{x \mapsto x \mid \chi[\![x]\!]\mu\}$$

Alternatively, since I denotes **the** identity map of $X \to \mathcal{P}X$, then

$$\mathrm{I}_\mu \triangleq \lhd\, [\![\mathsf{dom}\, \mu]\!]\mathrm{I}$$

Now, for each $\mu \in X \to \mathcal{P}X$, we form the extended map

$$\mathrm{I}_\mu \bowtie \mu$$

where $\bowtie$ is the join or 'pairing operator' which tuples a pair of maps with common domains (Mac an Airchinnigh 1990). Iteration over the range with respect to the characteristic function is readily achieved:

$$(\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu)$$

which constructs a map in $X \to \mathbb{B}$.
Remark: One will note that I_μ is referred to as an 'identity' map, the quotes being significant. The issue here is that we already have a unique identity function I. In our effort to be rigorous we have written I_μ for the construction of the required tupled map, but I for the iteration. The difference arises from our insistence that the operator $\bowtie$ is defined over a pair of maps if and only if said maps have the same domain.

The required invariant may now be given in either of the two forms

$$\Big((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu)\Big)^{-1}(1) = \emptyset$$

where we use the inverse image to check for the non-occurrence of x in S, or

$$\mathsf{rng}(\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) = \{0\}$$

where we rely on the property of non-duplication of elements in a set to arrive at the equivalent result.

4.3 The Operation

Let us suppose that nodes in the network are taken out of service, for whatever reason, and then brought back on line later. In a non-stop system, the outage may be constrained to be as little as 5 minutes per year! Let x' denote a node that is to be brought back on line. From the fixed network topology, it has been determined that x' is to be linked to node x. There will be other linkages to be restablished, of course. Confining ourselves to this single re-connection, we specify it as an enter command E as follows:

$$E[\![x, x']\!]\mu = \begin{cases} \mu \sqcup [x \mapsto \{x'\}], & \neg\chi[\![x]\!]\mu \\ \mu \dagger [x \mapsto \mu(x) \cup \{x'\}], & \text{otherwise} \end{cases}$$

Note the care that we take in considering whether or not x' has already been linked to some other node in the network. The mathematical expression we have seen before. Thus we might prefer to have written:

$$E[\![x, x']\!]\mu = \mu \Cup [x \mapsto \{x'\}]$$

No pre-condition is given. We shall now calculate the necessary and sufficient pre-condition by attempting to demonstrate that the enter operation satisfies the invariant.

4.4 The constructive calculation

For convenience, we write $\nu = E[\![x, x']\!]\mu = \mu \Cup [x \mapsto \{x'\}]$.

We are not yet able to reason efficiently with the $\Cup$ operator and must resort to its basic parts. In other words, we note that there is still work to be done. Hence, the proof will be accomplished by cases.

Case A: $\nu = E[\![x, x']\!]\mu = \mu \sqcup [x \mapsto \{x'\}]$, if $\neg\chi[\![x]\!]\mu$.

The first step is to construct the identity map and apply the join operator to tuple the resulting pair of maps:

$$\begin{array}{lll} 8 & \mathrm{I}_\nu \bowtie \nu & \\ .1 & = (\mathrm{I}_\mu \sqcup [x \mapsto x]) \bowtie (\mu \sqcup [x \mapsto \{x'\}]) & \\ .2 & = (\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto (x, \{x'\})] & \end{array}$$

Next we iterate over the result by applying the characteristic function to the range, to give:

$$\begin{array}{lll} 9 & (\mathrm{I} \to \chi)(\mathrm{I}_\nu \bowtie \nu) & \\ .1 & = (\mathrm{I} \to \chi)((\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto (x, \{x'\})]) & \\ .2 & = (\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \sqcup (\mathrm{I} \to \chi)[x \mapsto (x, \{x'\})] & \\ .3 & = (\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto \chi(x, \{x'\})] & \\ .4 & = (\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto \chi[\![x]\!]\{x'\}] & \end{array}$$

At this point we have a choice between application of a range operator, which readily establishes the pre-condition we are searching for, or application of the more exotic looking inverse image operator which is technically challenging. We choose the latter, exhibiting new theoretical results as a consequence, and leave the former as a simple exercise. Thus, application of the inverse image operator gives:

$$\begin{array}{lll} 10 & ((\mathrm{I} \to \chi)(\mathrm{I}_\nu \bowtie \nu))^{-1} & \\ .1 & = ((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto \chi[\![x]\!]x'])^{-1} & \\ .2 & = ((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1} \Cup [x \mapsto \chi[\![x]\!]x']^{-1} & \\ .3 & = ((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1} \Cup [\chi[\![x]\!]x' \mapsto \{x\}] & \end{array}$$

Remark: The passage from **10.1** to **10.2** is accomplished with a new theorem on inverse image maps:

Theorem 2 *For any pair of disjoint maps μ and ν in $X \to Y$,*

$$(\mu \sqcup \nu)^{-1} = \mu^{-1} \Cup \nu^{-1}$$

where μ^{-1}, ν^{-1}, $(\mu \sqcup \nu)^{-1}$ *in* $Y \to \mathcal{P}'X$, *and, in addition, for an evaluation at some* y

$$\begin{aligned}(\mu \sqcup \nu)^{-1}(y) &= (\mu^{-1} \circledcirc \nu^{-1})(y) \\ &= \mu^{-1}(y) \cup \nu^{-1}(y)\end{aligned}$$

Now we simply evaluate the result as required:

11 $((\mathrm{I} \to \chi)(\mathrm{I}_\nu \bowtie \nu))^{-1}(1)$
.1 $= ((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \sqcup [x \mapsto \chi[\![x]\!]\{x'\}])^{-1}(1)$
.2 $= ((\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1}(1) \cup [\chi[\![x]\!]\{x'\} \mapsto \{x\}](1)$
.3 $= \emptyset \cup [\chi[\![x]\!]\{x'\} \mapsto \{x\}](1)$
.4 $= [\chi[\![x]\!]\{x'\} \mapsto \{x\}](1)$
.5 $= \emptyset$ iff $\chi[\![x]\!]\{x'\} \neq 1$
.6 $\Rightarrow x \notin \{x'\}$
.7 $\Rightarrow x \neq x'$

which establishes the pre-condition for Case A. Note the evaluation step in the passage from **11.1** to **11.2**.

In the course of the proof, there were many other steps that we ought to have annotated, in particular giving the details of the various lemmas and theorems that follow as a consequence. However, such a detailed exposition would have clouded the overall structure of the proof, and the pride of place given to the theorem highlighted, might have been obscured.

The proof of the first case was relatively simple as expected. Proofs which involve disjoint maps usually are. The real technical problems we expect to meet when we consider the override operator, to which we turn next. However, before doing so, it is quite proper to remark on the distinction between the two resultant forms of the pre-condition: $x \notin \{x'\}$ versus $x \neq x'$. The latter we must use if we adhere to the form of the parameters supplied to the enter operation. The former is, in fact, more elegant and more general, suggesting a different form of the parameter list that permits a more general enter operation.
Case B: $\nu = E[\![x, x']\!]\mu = \mu \dagger [x \mapsto \mu(x) \cup \{x'\}]$, if $\chi[\![x]\!]\mu$.

The procedure is exactly the same as for Case A. The noteworthy aspect is the rewriting of the override operator in terms of the extend operator:

12 $\mathrm{I}_\nu \bowtie \nu$
.1 $= \mathrm{I}_\mu \bowtie (\mu \dagger [x \mapsto \mu(x) \cup \{x'\}])$
.2 $= (\lhd [\![x]\!]\mathrm{I}_\mu \sqcup \mathrm{I}_x) \bowtie (\lhd [\![x]\!]\mu \sqcup [x \mapsto \mu(x) \cup \{x'\}])$
.3 $= (\lhd [\![x]\!]\mathrm{I}_\mu \bowtie \lhd [\![x]\!]\mu) \sqcup (\mathrm{I}_x \bowtie [x \mapsto \mu(x) \cup \{x'\}])$
.4 $= \lhd [\![x]\!](\mathrm{I}_\mu \bowtie \mu) \sqcup ([x \mapsto (x, \mu(x) \cup \{x'\})])$
.5 $= (\mathrm{I}_\mu \bowtie \mu) \dagger ([x \mapsto (x, \mu(x) \cup \{x'\})])$

In the next stage the inter-conversion between the override operator and the extend operator and vice-versa has been condensed for simplicity. One may follow the paradigm established above to check the actual intermediate details, if necessary.

13 $(\mathrm{I} \to \chi)(\mathrm{I}_\nu \bowtie \nu)$
.1 $= (\mathrm{I} \to \chi)(\mathrm{I}_\mu \bowtie \mu) \dagger [x \mapsto \chi[\![x]\!](\mu(x) \cup \{x'\})]$

Choosing once more the application of the technically difficult inverse image operator, gives:

$$
\begin{array}{ll}
14 & ((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\nu \bowtie \nu))^{-1} \\
.1 & = ((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\mu \bowtie \mu) \dagger [x \mapsto \chi[\![x]\!](\mu(x) \cup \{x'\})])^{-1} \\
.2 & = (\mathrm{I} \rightarrow \lhd [\![x]\!])'((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1} \Cup [\chi[\![x]\!](\mu(x) \cup \{x'\}) \mapsto \{x\}]
\end{array}
$$

Remark: Justification of this result is based upon the more general theorem which is the counterpart of that given above:

Theorem 3 *For any pair of maps μ and ν in $X \rightarrow Y$,*

$$(\mu \dagger \nu)^{-1} = (I \rightarrow \lhd [\![\mathit{dom}\, \nu]\!])'\mu^{-1} \Cup \nu^{-1}$$

where μ^{-1}, ν^{-1}, $(\mu \dagger \nu)^{-1}$ in $Y \rightarrow \mathcal{P}'X$, and, in addition, for an evaluation at some y

$$
\begin{array}{rcl}
(\mu \dagger \nu)^{-1}(y) & = & ((I \rightarrow \lhd [\![\mathit{dom}\, \nu]\!])'\mu^{-1} \Cup \nu^{-1})(y) \\
& = & (I \rightarrow \lhd [\![\mathit{dom}\, \nu]\!])'\mu^{-1}(y) \cup \nu^{-1}(y)
\end{array}
$$

The priming on $(\mathrm{I} \rightarrow \lhd [\![\mathsf{dom}\, \nu]\!])$ denotes the fact that if as a result of the deletion we obtain an entry of the form $[y \mapsto \emptyset]$, then this must be deleted from the resulting inverse image map. We may establish this theorem by noting that

$$
\begin{array}{rcl}
(\mu \dagger \nu)^{-1} & = & (\lhd [\![\mathsf{dom}\, \nu]\!]\mu \sqcup \nu)^{-1} \\
& = & (\lhd [\![\mathsf{dom}\, \nu]\!]\mu)^{-1} \Cup \nu^{-1}
\end{array}
$$

Finally, for any set S in $\mathcal{P}Y$, it may be shown that

$$(\lhd [\![S]\!]\mu)^{-1} = (\mathrm{I} \rightarrow \lhd [\![S]\!])'\mu^{-1}$$

from which the theorem follows immediately.

It goes without saying that this result reduces to that given earlier for the inverse image of disjoint maps. Other significant theoretical consequences arise from the two theorems given, but this is not the place to present them.

Continuing with the rest of the calculation, we arrive at the expected result:

$$
\begin{array}{ll}
15 & ((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\nu \bowtie \nu))^{-1}(1) \\
.1 & = (\mathrm{I} \rightarrow \lhd [\![x]\!])'((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1}(1) \cup [\chi[\![x]\!](\mu(x) \cup \{x'\}) \mapsto \{x\}](1) \\
.2 & = ((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\mu \bowtie \mu))^{-1}(1) \cup [\chi[\![x]\!](\mu(x) \cup \{x'\}) \mapsto \{x\}](1) \\
.3 & = [\chi[\![x]\!](\mu(x) \cup \{x'\}) \mapsto \{x\}](1) \\
.4 & = \emptyset \text{ iff } \chi[\![x]\!](\mu(x) \cup \{x'\}) \neq 1 \\
.5 & \quad \Rightarrow x \notin (\mu(x) \cup \{x'\}) \\
.6 & \quad \Rightarrow x \notin \mu(x) \vee x \notin \{x'\} \\
.7 & \quad \Rightarrow x \notin \{x'\} \\
.8 & \quad \Rightarrow x \neq x'
\end{array}
$$

which gives the pre-condition for Case B. Happily, it is the same pre-condition as in Case A. A different result would have been disasterous.

In this example of invariant satisfaction we were fortunate indeed to have two distinct forms of the invariant, both of which are suitable for calculating the pre-condition. That proofs using both forms give the same result, is pre-eminently satisfying.

Regretfully, space does not permit us to explore this issue further here. Work that remains to be done, includes a complete listing of all operations

on the network and the accompanying pre-conditions, computed in the manner demonstrated. Although the details may (appear to) be tedious, it is comforting to know that once done, the results will endure forever. Therein lies the real benefit of the formal methods.

Having introduced the notion of a network topology, it seems fitting that we consider some further aspects of a communications network in connection with the medical part of our system. The next section is dedicated to the network view of the notify operations and other network issues.

Chapter 5

Network Topology, Routing and Node storage

The communications network for the medical system was assumed to be a star configuration. Let us now, though still keeping that same configuration, introduce the distinction between information held at a site and information at a network node as illustrated in Fig. 5.1.

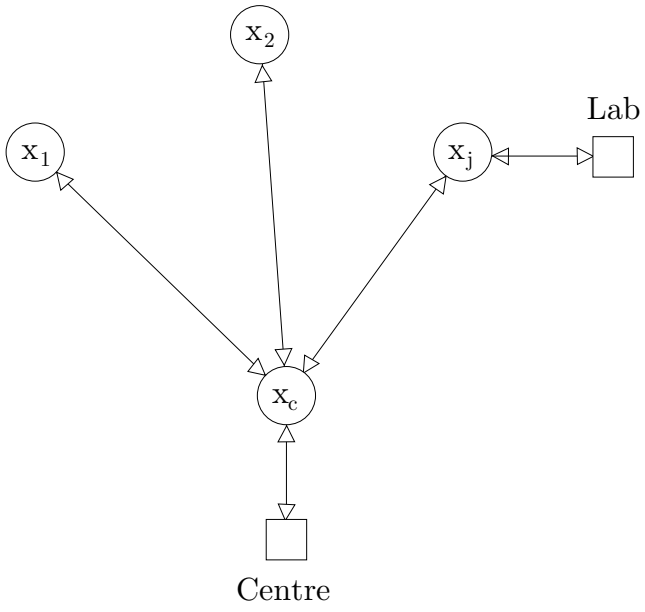

Figure 5.1: Network topology

For this section, we will use the more general single models for both lab site and centre:

$$\begin{aligned} \delta &\in DOC \rightarrow (PAT \rightarrow B\_TEST) \\ \psi &\in LAB \rightarrow (DOC \rightarrow (B\_TEST \rightarrow \mathbb{N}')) \end{aligned}$$

The network topology is modelled, as in the previous section, by

$$\nu \in X \rightarrow \mathcal{P}X$$

subject to the invariant that there are no self-loops:

$$((\mathrm{I} \rightarrow \chi)(\mathrm{I}_\nu \bowtie \nu))^{-1}(1) = \emptyset$$

Such a model is inherently asymmetrical, recording essentially unidirectional connectivity. That there is connectivity in the opposite direction, is modelled by

$$\nu^{-1} \in X \rightarrow \mathcal{P}X$$

In addition we will specify an invariant to capture the notion that all of the links are bidirectional:

$$\nu = \nu^{-1}$$

Alternatively, we may introduce a new operation based on set intersection and formulate the invariant as

$$\nu \mathbin{⩀}' \nu^{-1} = \nu$$

where $⩀$ is defined by

$$\nu \mathbin{⩀} [x \mapsto S] = \begin{cases} \theta, & \text{if } \neg\chi[\![x]\!]\nu, \\ \nu \dagger [x \mapsto \nu(x) \cap S], & \text{otherwise} \end{cases}$$

As usual, the prime denotes the removal of resulting entries of the form $x \mapsto \emptyset$.

Then for a network with unidirectional links only, the invariant is

$$\nu \mathbin{⩀}' \nu^{-1} = \theta$$

Networks with both bidirectional and unidirectional links will have an invariant of the general form

$$\theta \prec \nu \mathbin{⩀}' \nu^{-1} \prec \nu$$

Given a sender-receiver pair, and the network topology, we may then extract a model of the possible routings, ignoring for the present any issue of link capacity. Such routings are also modelled by

$$\rho \in X \rightarrow \mathcal{P}X$$

Finally, to complete the conceptual framework, we need to give a model for node storage, since we are implicitly assuming a store and forward packet switched network. Each packet will contain a header, giving the send-receiver 'addresses' and the data, which we will model as:

$$\sigma \in X \rightarrow \mathcal{P}(X^2 \times (DOC \rightarrow \mathcal{P}B\_TEST))$$

5.1 Forward & Store

Whereas the lab and the centre view the transmission of data as notify operations, the network sees the same transmission as the forwarding of data from one node (which is a removal operation) to some other node (which is an enter operation). Such removal and enter operations will involve the network storage model only. In other words, we will work entirely on the assumption of the star configuration, it being one of the simplest topologies. In particular, we can safely ignore the routing problem.

For convenience, we will work with a typical network storage example of the form

$$\sigma = \left[\begin{array}{lcl} x_1 & \mapsto & \{(\eta_1^1, \phi_1^1), (\eta_1^2, \phi_1^2)\} \\ x_2 & \mapsto & \{(\eta_2^1, \phi_2^1), (\eta_2^2, \phi_2^2)\} \\ \dots & & \dots \\ x_c & \mapsto & \{(\eta_c^1, \phi_c^1), (\eta_c^2, \phi_c^2), \dots, (\eta_c^n, \phi_c^n)\} \end{array} \right]$$

where $\eta = (x_s, x_r)$ is the header, and $\phi \in DOC \to \mathcal{P}B\_TEST$ is the data. The example illustrates the fact that at each lab node, there are two possible types of data, those that it is sending to the centre, and those that has been sent to it by the centre. For information, we note that

$$(\mathrm{I} \to \mathcal{P}\pi_1)\sigma$$

extracts a derived map of all the header information. The derived map of data information is

$$(\mathrm{I} \to \mathcal{P}\pi_2)\sigma$$

Let us deal with the removal operation first. We expect the specification to have the form

$$\begin{array}{ll} 16 & R\colon X^2 \times (DOC \to B\_TEST) \longrightarrow (\textsc{storage} \longrightarrow \textsc{storage}) \\ .1 & R[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \stackrel{\triangle}{=} \dots \end{array}$$

where it is clear that the data part of the (virtual) packet is a doctor, bloodtest pair. We really ought to introduce another model for the virtual packet and bring some more structure into the conceptual framework.

As far as the remove operation is concerned, we might consider a precondition of the form $\chi[\![x_j]\!]\sigma = \chi[\![\pi_1\eta]\!]\sigma$, where we suppose that the forwarding is from node x_j to x_k.

But from the model of storage, and repeated forwarding from the same node, it is clear that we may arrive at the situation where

$$\sigma(x) = \emptyset$$

We may wish to interpret this to mean that there is no data at the x node for onward transmission. We will treat such states as bugs, preferring to delete the entry. Similarly, there is also the possibility of $\chi[\![(\eta, \theta)]\!]\sigma(x)$ occurring, which indicates that all the packets associated with the header η have been forwarded. Again this is interpreted as a bug. We resolve this as above.

Given a header $\eta = (x_j, x_k)$, we use it to pick out the corresponding ϕ from $\sigma(x_j)$ and give the specification as a recursive algorithm:

$$
\begin{array}{ll}
17 & S\colon X^2 \longrightarrow (X^2 \times (DOC \to \mathcal{P}B\_TEST) \longrightarrow (DOC \to \mathcal{P}B\_TEST)) \\
.1 & S[\![(x_j, x_k)]\!]\sigma(x_j) \triangleq \\
.2 & \quad \text{let } (\eta, \phi) \in \sigma(x_j) \text{ in} \\
.3 & \quad\quad (x_j, x_k) = \eta \\
.4 & \quad\quad\quad \to \phi \\
.5 & \quad\quad\quad \to S[\![(x_j, x_k)]\!] \lhd [\![(\eta, \phi)]\!]\sigma(x_j)
\end{array}
$$

where, for validity, we must have the pre-condition $\chi[\![(x_j, x_k)]\!]\sigma(x_j)$. The recursive algorithm may be converted into an operator form such as $(x_j, x_k) \Downarrow \sigma(x_j)$, a process with which we are not very happy, leading as it does to a proliferation of needless operator symbols, and against which we often rail.

The next step in the specification is the removal of $[d \mapsto t]$ from $\phi = S[\![(x_j, x_k)]\!]\sigma(x_j)$. We will denote this by a primed removal:

$$R'[\![[d \mapsto t]]\!]\phi$$

taking care to delete entries of the form $d \mapsto \emptyset$. Such a removal is the analogue of bag diminution, which is formulated similarly to the subtraction of natural numbers—the predecessor function. Specifically, the bag diminution operator, $\ominus$, is defined by

$$\beta \ominus [x \mapsto 1] = \begin{cases} \lhd [\![x]\!]\beta, & \text{if } \beta(x) = 1, \\ \beta \dagger [x \mapsto \beta(x) - 1], & \text{otherwise} \end{cases}$$

The counterpart for the monoid of sets, $(X \to \mathcal{P}'Y, \bigcirc\!\!\!\!\!\cup, \theta)$, may be defined similarly:

$$R'[\![[x \mapsto \{y\}]]\!]\mu = \begin{cases} \lhd [\![x]\!]\mu, & \text{if } \mu(x) = \{y\}, \\ \mu \dagger [x \mapsto \lhd [\![y]\!]\mu(x)], & \text{otherwise} \end{cases}$$

It should be quite obvious that the operator symbol, $\lhd$, and the function symbol, R, are inter-changeable.

To denote the updated information at node x_j as a result of forwarding $[d \mapsto t]$, we may now write

$$\lhd [\![(\eta, \phi)]\!]\sigma(x_j) \cup \{(\eta, R'[\![[d \mapsto t]]\!]\phi)\}$$

a specification that recalls the definition of the override operator in terms of extend. Hence, the specification of removal, which we have developed so far, reads

$$R[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \sigma \dagger [x_j \mapsto \lhd [\![(\eta, \phi)]\!]\sigma(x_j) \cup \{(\eta, R'[\![[d \mapsto t]]\!]\phi)\}]$$

But we are now very conscious of the bugs that may arise. In this case, by repeatedly forwarding the same doctor's blood test data, it is possible to obtain an entry of the form

$$x_j \mapsto \{(\eta, \theta), \dots\}$$

Thus, in the event that $R'[\![[d \mapsto t]]\!]\phi = \theta$, we add the extra clause

$$R[\![(x_j, x_k), [d \mapsto t]]\!]\sigma = \sigma \dagger [x_j \mapsto \lhd [\![((x_j, x_k), \phi)]\!]\sigma(x_j)$$

Finally, this new clause may lead to an entry of the form

$$x_j \mapsto \emptyset$$

giving us the bug on the higher level. This bug is eliminated by priming the outer removal operator. To complete the definition of the removal operation, we really ought to eliminate the temporary variable that we have been using, $\phi = S[\![(x_j, x_k)]\!]\sigma(x_j)$, and which we leave as an exercise.

Quite clearly, the definition of the forward part has resulted in a very complex mathematical expression. The complexity arose out of dealing with null structures, θ, and $\emptyset$, introduced as a direct result of the removal operator. As a result we are led to consider whether the specification can be lifted to a higher level of abstraction by introducing new monoids or new morphisms. Alternatively, we also consider changing the basic model representation. But all such activity is meat for the School of the VDM♣. For all its complexity, the expression is at least manageable compared to the actual code that one might find in the real system.

Let us now look at the store part of the forward and store operation. Fortunately, the expression will not be so complex. Due to our policy of deleting (η, ϕ) entries, then it is quite possible that there is no destination entry in the model, i.e., $x_k \mapsto \emptyset$. Hence, we expect

$$E[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \sigma \sqcup [x_k \mapsto \{((x_j, x_k), [d \mapsto t])\}]$$

If there had been a destination entry, but no information associated with (x_j, x_k), i.e., $\neg\chi[\![((x_j, x_k), \phi)]\!]\sigma(x_k)$, then the enter operation would have read

$$E[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \sigma \dagger [x_k \mapsto \sigma(x_k) \cup \{((x_j, x_k), [d \mapsto t])\}]$$

From our previous work we are led to combine the two specifications using a new operator the meaning of which should be obvious from context:

$$E[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \sigma \circledcirc_2 [x_k \mapsto \{((x_j, x_k), [d \mapsto t])\}]$$

This form of the enter operation deals conclusively with the two situations where the two outermost null bugs would have arisen. In the other innermost null bug case, the specification is

$$E[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \begin{cases} \sigma \dagger [x_k \mapsto \lhd [\![(\eta, \phi)]\!]\sigma(x_k) \cup \{(\eta, \phi \sqcup [d \mapsto \{t\}])\}] \\ \sigma \dagger [x_k \mapsto \lhd [\![(\eta, \phi)]\!]\sigma(x_k) \cup \{(\eta, \phi \dagger [d \mapsto \phi(d) \cup \{t\}])\}] \end{cases}$$

where the distinction is based on whether or not $\phi(d) = \theta$. We recognise our monoid of sets in this expression and can condense it to the form

$$E[\![(x_j, x_k), [d \mapsto t]]\!]\sigma \triangleq \sigma \dagger [x_k \mapsto \lhd [\![(\eta, \phi)]\!]\sigma(x_k) \cup \{(\eta, \phi \circledcirc [d \mapsto \{t\}])\}]$$

The process leading to the specification of the enter part of the store and forward operation seems much shorter and easier than the removal part. This is due to two very important reasons. By beginning with the removal part we were forced to address the pernicious null bugs. That we took this approach is almost entirely due to our familiarity with certain monoids. The second reason is

simply that remove and enter are the inverses of each other. Once the removal was complete, then we presented the specification of enter in the reverse order. There is, of course, a third reason—the use of the auxillary variables η and ϕ which guided us in the specification of the removal operation. We found it very convenient to rely on these for the last part of the enter operation specification.

What can we say about the specifications that have beem produced? The first question that must be asked is: Are they correct? We must confess that we can not be too sure. This text is the third time that the specifications have been developed/elaborated. They must still be regarded as work in progress. The usefulness of the η and ϕ variables suggest splitting up the original models. It appears that we ought to introduce some auxillary models in order that η and ϕ achieve a permanent status. We did hint at this in the beginning of the section when we referred to headers, η, and collections of data packets, ϕ.

Another way in which we might check the correctness of the specifications, is to introduce an invariant that captures the amount of information in the storage in the network at any instant, and then prove that appropriate forward, store operations conserves that invariant.

5.2 Routing

To complete this Chapter, we will now address the distinction between the 'static' network topology and the dynamic network routing, in a completely general setting. The quotes around static are intended to emphasise that at any instant, the topology of the network is fixed, in contrast to the dynamic nature of routing extraction from that fixed topology. However, as nodes may be taken out of service, and later switched back in, or as communication links may fail, etc., then the topology itself will no longer be static but vary over time. It will be clear from the specification of the routing, that the same mathematics may be used to analyse and specify the fragility of the network topology itself, in the presence of failure[1].

For definiteness, we may work initially with a topology such as that shown in Fig. 5.2. In the VDM we may demonstrate that such a topology is a reification

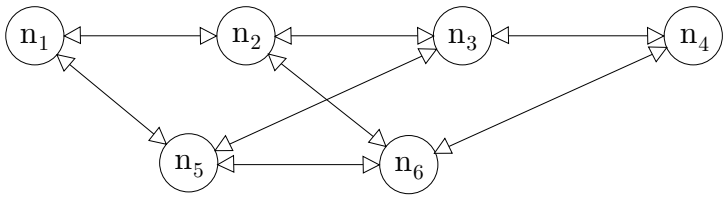

Figure 5.2: Network topology

of the star configuration, together with extra connectivity between lab nodes.

[1]Apparently there is an error in this work (somewhere). Dr. Andrew Butterfield, who appears to be the only one of whom I am aware to have checked the specification, informs me that I only detect *cul de sac*s of depth 1 and that I ought to have used transitive closure. I am sure he is right. The reader is invited to detect the defect and to repair it appropriately.

But that takes us into the verification part of the method. We are concerned with validation in this tutorial.

Now let us suppose that the node n_1 wishes to send data to node n_4. Then, from the diagram it is clear that we obtain the possible routings given in Fig. 5.3. Formally, we will extract the routings from the network topology by giving a

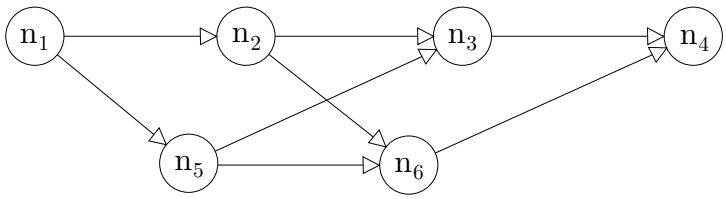

Figure 5.3: Network routing

recursive algorithm, denoted by $\mathcal{E}$. Since both network topology and routing are essentially graphs, then the algorithm is based on graph traversal. We have modelled such graphs and graph traversals in the style of the VDM♣ before and presented preliminary results in (Mac an Airchinnigh 1991), where we noted our uncertainty as to the equivalence of two depth-first search algorithms, drawn from the literature.

The routing extraction problem has given us a fresh opportunity to look at the problem again *ab initio* and to obtain, what we believe to be, an elegant result. Relying on the development presented in (Mac an Airchinnigh 1991), we propose to be very brief in our annotation of the following extraction algorithm. First we give the algorithm for the simple case that all the links are bidirectional.

18 $\quad \mathcal{E}\colon X^2 \longrightarrow \text{TOPOLOGY} \longrightarrow (\text{TOPOLOGY} \times \mathit{Routing} \times \text{BACK\_EDGES})$

.1 $\quad \mathcal{E}[\![s,t]\!]\gamma \triangleq \mathcal{E}_t^s[\![\gamma(s)]\!](\gamma, [s \mapsto \gamma(s)], [s \mapsto \gamma(s)]^{-1})$

Annotations

18 For convenience we give names for the models rather than their structure. The TOPOLOGY and ROUTING models we have seen before. The BACK_EDGES model is exactly the same structure as the other two.

18.1 We hand over the computation to another extraction algorithm of the same name. The form of the right hand side is standard in the VDM♣. Of particular note is the inclusion of the topology γ as a read-only (data) structure. As well as the result to be computed, the routing, we will also compute its complement, the back edge structure.

The transformation of the source and target nodes, s, and t, from parameters to a superscript, and subscript, respectively, adds to the clarity of the functionality of the algorithm. Again this is standard.

19 $\quad \mathcal{E}\colon X^2 \times \mathcal{P}X \longrightarrow$
$\qquad ((\text{TOPOLOGY} \times \mathit{Routing} \times \text{BACK\_EDGES}) \longrightarrow$
$\qquad\quad (\text{TOPOLOGY} \times \mathit{Routing} \times \text{BACK\_EDGES}))$

.1 $\quad \mathcal{E}_t^s[\![\emptyset]\!](\gamma, \gamma_r, \gamma_b) \triangleq (\gamma, \gamma_r, \gamma_b)$

$.2\ \ \mathcal{E}_t^s[\![\{n\} \uplus S]\!](\gamma, \gamma_r, \gamma_b) \stackrel{\triangle}{=}$
$.3\ \ \quad \chi[\![n]\!]\gamma_r \vee (n = t)$ — visited(n) or target(n)?
$.4\ \ \quad\quad \rightarrow \mathcal{E}_t^s[\![S]\!](\gamma, \gamma_r, \gamma_b)$
$.5\ \ \quad\quad \rightarrow \mathcal{E}_t^s[\![S]\!]\mathcal{E}_t^n[\![\lhd [\![\,\mathsf{dom}\,\gamma_r]\!]\gamma(n)]\!](\gamma,$
$.6\ \ \quad\quad\quad\quad \gamma_r \sqcup [n \mapsto \lhd [\![\,\mathsf{dom}\,\gamma_r]\!]\gamma(n)], \gamma_b \mathbin{\textcircled{\cup}} [n \mapsto \lhd [\![\,\mathsf{dom}\,\gamma_r]\!]\gamma(n)]^{-1})$

Annotations

19 The algorithm has the signature of a tail-recursive function. But from **19.5** we see that the algorithm is, in fact, binary recursive. Thus, this algorithm belongs to the more general class of curried functions.

19.1 $\mathcal{E}_t^s[\![\emptyset]\!]$ is the identity function.

19.3 If the new node n has already been visited, then we can ascertain this fact from the current routing. In addition, we check whether or not it is the target node ...

19.4 ... and, if either case is true, then no further depth searching is possible.

19.5 This is the heart of the algorithm. $\mathcal{E}_t^n$ denotes the depth-first search component. Space does not permit an extended treatment on the rationale for the result. The interested reader is urged to apply the algorithm to a concrete simple example and discern the reason for the choice of expressions. This is how the author developed them in the first place!

Of particular note is the fact that the algorithm computes all possible routings. It does not concern itself with optimal routings. For that additional structure and information is necessary.

Now let us turn to the general algorithm which deals with the extraction of routings from a network topology in which some of the links are unidirectional. Such may be the case, if there is a unidirectional ring embedded in a bidirectional network. The algorithm follows:

$20\ \ \mathcal{E}\colon X^2 \longrightarrow \textsc{topology} \longrightarrow (\textsc{topology} \times \mathit{Routing} \times \textsc{back\_edges})$
$.1\ \ \mathcal{E}[\![s,t]\!]\gamma \stackrel{\triangle}{=} \mathcal{E}_t^s[\![\gamma(s)]\!](\gamma, [s \mapsto \gamma(s)], [s \mapsto \gamma(s)]^{-1} \mathbin{\textcircled{\cap}'} \lhd [\![\gamma(s)]\!]\gamma)$

Annotations

20.1 The noteworthy difference from the previous algorithm is in the computation of the back edges. This is only to be expected since some of our links are unidirectional. Note the occurrence of the primed intersection operator that we have already introduced earlier—a nice surprise! In addition, a restriction operator is essential for correctness. Once more, the only advice we can offer for the present, is the exercising of the algorithm on a simple example (see below).

$21\ \ \mathcal{E}\colon X^2 \times \mathcal{P}X \longrightarrow$
$\qquad ((\textsc{topology} \times \mathit{Routing} \times \textsc{back\_edges}) \longrightarrow$
$\qquad\quad (\textsc{topology} \times \mathit{Routing} \times \textsc{back\_edges}))$
$.1\ \ \mathcal{E}_t^s[\![\emptyset]\!](\gamma, \gamma_r, \gamma_b) \stackrel{\triangle}{=} (\gamma, \gamma_r, \gamma_b)$
$.2\ \ \mathcal{E}_t^s[\![\{n\} \uplus S]\!](\gamma, \gamma_r, \gamma_b) \stackrel{\triangle}{=}$

$$
\begin{array}{ll}
.3 & \quad \chi[\![n]\!]\gamma_r \vee (n = t) \vee \lhd\!\!\lhd [\![\mathsf{dom}\, \gamma_r]\!]\gamma(n) = \emptyset \quad \text{— visited}(n) \text{ or target}(n) \\
.4 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{or } \textit{cul de sac}? \\
.5 & \quad \rightarrow \mathcal{E}^s_t[\![S]\!](\gamma, \gamma_r, \gamma_b) \\
.6 & \quad \rightarrow \mathcal{E}^s_t[\![S]\!]\mathcal{E}^n_t[\![\lhd\!\!\lhd [\![\mathsf{dom}\, \gamma_r]\!]\gamma(n)]\!](\gamma, \\
.7 & \qquad\qquad \gamma_r \sqcup [n \mapsto \lhd\!\!\lhd [\![\mathsf{dom}\, \gamma_r]\!]\gamma(n)], \\
.8 & \qquad\qquad \gamma_b \Cup ([n \mapsto \lhd\!\!\lhd [\![\mathsf{dom}\, \gamma_r]\!]\gamma(n)]^{-1} \Cap' \lhd [\![\lhd\!\!\lhd [\![\gamma_r]\!]\gamma(n)]\!]\gamma))
\end{array}
$$

Annotations

21.3 Of particular interest is the added test. Because of the unidirectional links, we can find ouselves in a *cul de sac*, a dead end. This was one of the most unexpected results discovered in the course of developing the algorithm.

To illustrate this case, we propose the example of a triangular unidirectional ring network (n_1, n_2, n_3), with bidirectional links ($n_1 \leftrightarrow n_4$, $n_2 \leftrightarrow n_5$, $n_3 \leftrightarrow n_6$) connecting each of the vertices to 'terminal nodes', (n_4, n_5, n_6). Computation of the routings for n_4 to n_6 produces a dead end example.

21.8 It took a full eight hours to arrive at this back edge computation and to obtain the given expression. Upon looking at it again, we must confess that we are astonished that we discovered it at all, and most important of all, we wonder whether there is not a simpler expression for the same result.

Fortunately, the corresponding encoded prototype may be exercised to determine, if indeed, the expression gives the desired result. In addition, the expression is given in terms of known operators and, therefore, amenable to subsequent theoretical analysis.

The presentation of the two extraction algorithms concludes the main body of the technical content of the tutorial. We feel that sufficient material has been presented to illustrate most of the issues involved in the validation of requirements and specifications via the use of formal methods. It is now time to conclude.

Chapter 6

Epilogue

The text is dead, long live the dialogue!

Mathematics and Formal methods are something that one does, and when there are results, whether simple or great, one shares. How can one possibly convey the pain of labour, the excitement of discovery? This tutorial text has been written in a typical development style. A broad brush was used, where we boldly sketched out some of the models that arise in a typical application. We had intended to finish with a flourish by showing how the various domains could be brought together in a multimedia services application, incorporating the views of the customer, the service provider, and the network operator. Unfortunately, this was not to be.

In the domain of testing, the most obvious technical contributions were presented as models and their invariants, operations and their pre-conditions, and the use of retrieve functions. Several classes of common pernicious bugs were carefully chosen and illustrated. But were we to draw the reader's attention to the most important aspect of the use of the mathematics for testing, we would have to mention the significance of the Feynman test.

All of our programming and system building, and all of our testing, does rest on a very sound theoretical basis. The mathematics is not (yet) very difficult. Only the notation looks strange. In our opinion, we can find no reason why software engineers ought not to be as well-versed in applied constructive mathematics, such as that of the VDM♣, as are 'ordinary' engineers in classical mathematics. One may not have to use the mathematics in daily work, but having the right mathematical engineering culture that is appropriate to computing, will surely imbue the software engineer with the appropriate spirit. When can we expect an analogous Feynman test for computing?

6.1 Acknowledgements

To persons: First and foremost, we express our gratitude to **K&M Technologies**, Limited (Ireland), of which the author is a Technical Director, for the considerable support in both time and other resources, without which both the first and this current version of the tutorial could not have been prepared.

We also owe a debt of gratitude to the other members of the School of the VDM♣, Dr. A. Butterfield, A. Donnelly, E. McDonnell and G. O'Reagan, who

endured the development and presentations of many of the theoretical results. Although the theory on inverse images had been developed in February 1991, it is clear from the published proof of satisfaction of the annihilator invariant by an enter operation (Mac an Airchinnigh 1991), that the inverse image theory had not yet become an integral part of our methods. Since the publication of the first version of the tutorial, in May 1993, other members of the School have contributed to improvements. In addition to the above, we must mention A. Farrell, C. O'Reilly and A. Hughes. Although not strictly a member of the School, D. Gallagher is gratefully acknowledged for his significant contributions.

For the specific problem on eliminating self-loops in a network presented in § 4, and the very idea of the calculation of the pre-condition, I am indebted to Dipl.-Ing. E. H. Dürr, who asked the author to review a paper which he was preparing, together with Prof. S. J. Goldsack, on the modelling of distributed systems based on work for the ESPRIT AFRODITE project.

For this second version of the tutorial, Prof. Peter Lucas, now at the Technical University of Gratz, Austria, added important historical insights both on the original research group as well as correcting erroneous technical 'insights' of my own. In addition he proof-read the original version carefully after publication and has contributed substantially to the clarity of the present one. Needless to remark all of the remaining errors, whether of a technical nature or of a quirkiness of style are truly my own.

To technologies: Naturally, the cognoscendi will recognize TEX joutput when they see it. However, the document was actually typeset with LATEX 2e (Goossens, Mittelbach, and Samarin 1994). For a variety of historical and æsthetic reasons I have tried to adhere to the recommendations in the 14th edition of *The Chicago Manual of Style* (Grossman 1993) and therefore, acknowledge the use of the BibTeX style based on the 13th edition (Paulley 1992).

For the Greek text, Linguists Software; for the typesetting, TeXtures from BlueSky.

Bibliography

Aristotle (1987). Poetics. In J. L. Ackrill (Ed.), *A New Aristotle Reader*, pp. 540–56. Oxford: Oxford University Press.

Armstrong, M. A. (1983). *Basic Topology*. Berlin: Springer-Verlag. A reprint of the 1978 McGraw-Hill Book Company text for the *Undergraduate texts in mathematics* (UTM) series of Springer-Verlag.

Beizer, B. (1990). *Software Testing Techniques* (Second ed.). New York: Van Nostrand Reinhold.

Bjørner, D. and C. B. Jones (Eds.) (1978). *The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science* **61**. Berlin: Springer-Verlag.

Bjørner, D. and C. B. Jones (Eds.) (1982). *Formal Specification and Software Development*. Englewood Cliffs, New Jersey: Prentice/Hall International.

Dawkins, R. ([1976] 1989). *The Selfish Gene* (New ed.). Oxford: Oxford University Press.

Dieudonné, J. (1992). *Mathematics—the Music of Reason*. Berlin: Springer-Verlag. H. G. and J. C. Dales (trans.), from the French *Pour l'honneur de l'esprit humain*, Hachette, Paris, 1987.

Diller, A. (1990). *$\mathcal{Z}$: An Introduction to Formal Methods*. Chichester, England: John Wiley & Sons.

Epstein, D. and S. Levy (1995, June). Experimentation and Proof in Mathematics. *Notices of the American Mathematical Society 42*(6), 670–4.

Garnerin, P. et al. (1991). A Visualization Tool for Representing Epidemic Spread. *The Mathematica Journal 1*(4), 59–61.

Gleick, J. (1992). *Genius, Richard Feynman and Modern Physics*. London: Little, Brown and Company.

Goldblatt, R. ([1979] 1984). *Topoi, The Categorial Analysis of Logic* (Revised ed.). Amsterdam: North-Holland.

Goossens, M., F. Mittelbach, and A. Samarin (1994). *The LaTeX jCompanion*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Grossman, J. (Ed.) ([1906] 1993). *The Chicago Manual of Style* (Fourteenth ed.). Chicago: University of Chicago Press.

Hennessy, M. (1988). *Algebraic Theory of Processes*. Cambridge, Massachusetts: The MIT Press.

IEEE Computer Society (1990). IEEE Guide to Software Requirements Specifications, ANSI/IEEE Std 830–1984. In M. Dorfman and R. H. Thayer (Eds.), *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, pp. 16–38. Los Alamitos, California: IEEE Computer Society Press.

Jacobson, N. (1985). *Basic Algebra I* (Second ed.). New York: W. H. Freeman.

Kline, M. (1972). *Mathematical Thought from Ancient to Modern Times*. Oxford: Oxford University Press. The 1990 three volume paperback work is cited.

Liddell and Scott ([1889] 1986). *An Intermediate Greek-English Lexicon* (Seventh Abridged ed.). Oxford: Oxford University Press.

Mac an Airchinnigh, M. (1990). *Ph.D. Thesis: Conceptual Models and Computing*. University of Dublin, Trinity College, Dublin, Ireland: Department of Computer Science.

Mac an Airchinnigh, M. (1991). Tutorial Lecture Notes on the Irish School of the VDM. In S. Prehn and W. J. Toetenel (Eds.), *VDM'91, Formal Software Development Methods Volume 2: Tutorials, Lecture Notes in Computer Science* **552**, pp. 141–237. Berlin: Springer-Verlag.

Mac an Airchinnigh, M. (1993, May). Formal Methods & Testing. In *Tutorials of the Sixth International Software Quality Week*, 625 Third Street, San Francisco, CA 94107–1997. Software Research Institute.

Mac an Airchinnigh, M. and H.-J. Kugler (1994). Service Engineering versus Software Engineering, a Foundational Study. In H.-J. Kugler, A. Mullery, and N. Niebert (Eds.), *Towards a Pan-European Telecommunication Service Infrastructure—IS&N'94, Lecture Notes in Computer Science* **851**, pp. 39–49. Berlin: Springer-Verlag.

National Bureau of Standards (U.S.) (1990). Guideline for Lifecycle Validation, Verification, and Testing of Computer Software, FIPS PUB 101. In M. Dorfman and R. H. Thayer (Eds.), *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, pp. 63–100. Los Alamitos, California: IEEE Computer Society Press.

Paulley, G. (1992, August). chicago.sty. Data Structuring Group, Department of Computer Science University of Waterloo Waterloo, Ontario, Canada N2L 3G1. Contains the LaTeX style command definition for the Chicago BibTeX styles chicago.bst and chicagoa.bst. Formatted according to the 13th edition of the Chicago Manual of Style. May be obtained by access to ftp.math.utah.edu.

Rudin, W. (1970). *Real and Complex Analysis*. McGraw-Hill.

Stoll, R. R. ([1961] 1974). *Sets, Logic, and Axiomatic Theories* (Second ed.). San Francisco: W. H. Freeman and Company.

Tanaka, Y. (1995, June). Meme Media and its World-wide Pool for the Exchange and Evolution of Knowledge. In *Proceedings of the 20th International Conference with Summer School*, Sofia, Bulgaria, pp. 22–40. FOI-COMMERCE.

Tel, G. (1991). *Topics in Distributed Algorithms*. Cambridge: Cambridge University Press.

Tierney, M. (1992). Software Engineering Standards: the 'Formal Methods Debate' in the UK. *Technology Analysis & Strategic Management 4*(3), 245–78.

Toetenel, W. J. (1992). *Ph.D. Thesis: Model Oriented Specification of Communicating Agents*. Technical University of Delft, The Netherlands: Department of Computer Science.

Woodcock, J. and M. Loomes (1988). *Software Engineering Mathematics, Formal Methods Demystified*. London: Pitman Publishing.

Notation Glossary

Sets

$\mathcal{P}$	the powerset functor.
$\mathcal{P}X$	the powerset of set X.
$\mathcal{P}'X$	$\mathcal{P}X$ excluding $\emptyset$.
$\mathcal{P}f$	the iterating of function f over a set.
$\emptyset$	the unique null set.
$S \cup T$	set union.
$S \sqcup T$	disjoint set union.
$S \cap T$	set intersection.
$\lhd_S T$	set difference;
	classical mathematics uses $S - T$ or $S \backslash T$.
$\chi[\![a]\!]S$	test for set membership $a \in S$.
$card\, S$	the cardinality of a set S; $\|S\|$ is also used.
$\pi_\epsilon S$	a projection operator that selects a random element of a set S.

Sequences

*	the sequence functor.
$\Sigma^\star$	the sequence of elements over (alphabet) Σ
$f^\star$	the iterating of function f over a sequence.
Λ	the unique null sequence.
$len\, \sigma$	the length of a sequence σ; $\|\sigma\|$ is also used.
$elems\, \sigma$	the set of elements in a sequence σ.
$items\, \sigma$	the bag of elements in a sequence σ.
$\pi_j \sigma$	a projection operator that selects the jth element of a sequence σ.

Maps

$\rightarrow$	the map functor.
$X \rightarrow Y$	the space of maps from X to Y.
$f \rightarrow g$	the iterating of a pair of functions; f is bijective.
θ	the unique null map.
I	the identity map.
$\mathsf{dom}\,\mu$	the domain of the map μ.
$\mathsf{rng}\,\mu$	the range of the map μ.
$\mu \sqcup \nu$	the extend, or merge of two disjoint maps.
$\mu \cup \nu$	the glueing of two maps which agree on $\mathsf{dom}\,\mu \cap \mathsf{dom}\,\nu$.
$\mu \dagger \nu$	the override or overwrite of two maps.
$\nu \circ \mu$	the composition of two maps $\mu \in X \rightarrow Y$ and $\nu \in Y \rightarrow Z$; defined over $\mathsf{rng}\,\mu \cap \mathsf{dom}\,\nu$.
$\mu \bowtie \nu$	the join of two maps $\mu \in X \rightarrow Y$ and $\nu \in X \rightarrow Z$; defined over $\mathsf{dom}\,\mu \cap \mathsf{dom}\,\nu$.
$Y \rightarrow \mathcal{P}'X$	the space of inverse image maps.
μ^{-1}	the inverse image of the map μ.
$\lhd_S \mu$	the removal of a map μ with respect to a set S. other variants are $\lhd [\![S]\!]\mu$ and $S \lhd \mu$; classical mathematics uses $\mu \backslash S$.
$\lhd_S \mu$	the restriction of a map μ with respect to a set S. other variants are $\lhd [\![S]\!]\mu$ and $S \lhd \mu$; classical mathematics uses $\mu \mid S$.
$(\mathrm{I} \rightarrow \lhd_S)'$	$(\mathrm{I} \rightarrow \lhd_S)$ with removal of $y \mapsto \emptyset$ elements.