Principles for Generalised Testing of Knowledge Bases

Tim Menzies
BSc (Com. Sci.)-UNSW;
MCogSc.- UNSW

Artificial Intelligence Laboratory
School of Computer Science and Engineering
University of New South Wales
P.O. Box 1 Kensington,
NSW, Australia, 2033

A thesis submitted as partial requirements of a
Doctor of Philosophy (Computer Science).
September 28, 1995

ABSTRACT

Modern KA views KBS construction as the construction of inaccurate surrogates models of reality. We argue that such potentially inaccurate models must be tested, lest they generate inappropriate output for certain circumstances. Testing can only demonstrate the presence of bugs (never their absence) and so must be repeated whenever new data is available. That is, testing is an essential, on-going process through-out the lifetime of a knowledge base.

This view motivated our development of a general computational model for automatically testing models in *vague domains*. A vague domain is (i) poorly-measured; and/or (ii) lacks a definitive oracle; and/or (iii) is indeterminate/non-monotonic. Testing in such vague domains necessitates making assumptions about unmeasured variables and maintaining mutually exclusive assumptions in separate worlds. Most domains tackled by KBS are vague; i.e. our definition of test is widely applicable.

Surprisingly, our generalised test engine is also a generalised inference engine. Domain-specific modelling constructs are mapped into a directed and-or graph of edges E and vertices V. Consistent subsets of E are extracted which explain known observations in terms of known inputs to the model. This extraction process directly operationalises the model extraction process which Clancey and Breuker argue is the core of expert systems inference. We find that the second generation knowledge level modelling approaches (e.g. KADS) complicates and separates processes that can be unified and simplified via our abductive framework. Consequently, we propose replacing methodologies like KADS with our abductive architecture that supports and simplifies *both* inference and testing.

Limits to model testing are also limits to model construction since potentially inaccurate models that can't be tested should not be used. We find that our process is practical for model at least up to $|V| = 850$ and $|E|/|V| < 7$. However, we lose the ability to test models in vague domains above $|E|/|V| = 7$. Based on surveys of known fielded expert systems, we conclude that: (i) our technique is practical for the models seen in current KA practice; however, (ii) modern KA is teetering on the edge of model testability/construction.

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgment is made in the text.

Tim Menzies

September 28, 1995

The gods did not reveal, from the beginning; All things to us, but in the course of time; Through seeking we may learn and know things better; But as for certain truth, no man has known it; Nor shall he know it, neither of the gods; Nor yet of all the things of which I speak; For even if by change he were to utter; The final truth, he would himself not know it; For all is but a woven web of guesses: Xenophanes

I raise my quill, I am responsible: Voltaire *I'll play with it first and tell you what it is later:* Miles Davis. *I have made this letter longer because I did not have time to make it shorter:* Blaise Pascal (1656). *If the journey is not what you expect, do not be surprised:* Anonymous

1. Introduction

A shrine is a venue for reverently worshipping the wisdom of others. A coffee shop is a venue for heated debate. We believe that we should not view knowledge bases as shrines for possibly out-dated insights. Rather, we should view them as coffee shops; i.e. venues for constant intense argument and/or review and/or revision of contemporary ideas. This research is about general engineering principles for such "argument environments".

In this chapter, we discuss why we believe that argument environments are so important. We also offer a statement of thesis, a chapter plan, some notes on our pseudo-code notation, and our acknowledgments.

1.1. About Doubting

1.1.1. Humans and Uncertainty

A general trend in the twentieth century is an increasing level of doubt about the things we speak or write or try to enter into programs[1]. Many factors have combined to reduce our belief that we can know the "truth" (in some absolute sense) about our world; for example: relativity[2], Heisenburg's uncertainty principle[3], the indeterminacy of quantum mechanics[4], Gödel's theorem[5], the failure of AI to replicate human cognitive via

[1] Doubts about the certainty of human knowledge are not a twentieth century invention, as testified by Xenophane's poem at the start of this chapter.

[2] Contrary to popular belief, Einstein did not say that "everything was relative". However, he did demonstrate that some of our oldest notions about basic physical concepts such as mass and time were only approximations yielding useful results at velocities much less than the speed of light. Einstein's challenge to certainty was to illustrate that certain "truths" may only be approximately true, within certain boundaries.

[3] Given pairs of certain measurements (position and momentum), knowing one exactly introduces an unavoidable error into the measurement of the other. This error is small (proportional to $6.634 * 10^{-34}$). However, the realisation that everything is not precisely measurable shocks advocates of absolute truth.

[4] Sub-microscopic events can just happen, without some identifiable cause. The extrapolation to macroscopic events having no rational explanation is a possible consequence, but has not yet been demonstrated to be commonly true.

[5] Within any interesting axiomatic system (i.e. at least powerful enough to express basic arithmetic), there exist

manipulation of symbols according to classical logic[6], chaos theory[7], and the inability of human designers to get designs right the first time[8].

Even if knowledge and truth existed in some abstract form, it seems human beings have less-than-perfect access to it. Anderson observes:

> *Human reasoning does not always correspond to the prescriptions of logic. People … fail to see as valid certain conclusions that are valid, and they see as valid certain conclusions that are not. ([7] p264)*

For example, experiments with human comprehension of a syllogisms[9] demonstrates how illogical human thought can be. In one study with the syllogism

> *P implies* Q
> Q *is not true*
> *therefore* P *is also not true*

39% of the subjects incorrectly stated that the syllogism could only sometimes be true while 4% wrongly stated that it was never valid ([7] p267). Why is it that the same problem can be processed in different ways by different people? Anderson discounts explanations based on the different reasoning prowess of the sampled humans. In another study, 90% of subjects incorrectly understood a syllogism, including trained logicians; presumably, the most rational of all human beings ([7] p362). The way a problem is presented can effect how it is solved. Information is not accepted by human beings at face value. Rather, the way data is presented has an effect on the importance the decision maker attaches to that data. Popular examples of this process are common:

> *It is evident that when the instances on one side of a question are more likely to remembered and recorded that those on the other, especially if there be any strong motive to preserve the memory of the first, but not of the latter, these last are likely to over-looked, and escape the observations of the mass of mankind:* John Stuart Mill.

comment on the importance of the unprovable sentences. You may be unable to know precisely the length of the 39th hair above your left hear, but it hardly matters. However, Gödel's theorem should still makes you nervous since you cannot prove that conclusions which do matter are not in your personal Gödel set.

[6] If the universe was as logical and as rational as Plato suggested, then a fast theorem prover should be able to reproduce human intelligence. This turns out not to be the case. Significant progress has been made only in well-defined, limited, domains (e.g. [9, 25, 73, 117, 144, 157, 164, 171, 206, 236]). Attempts to replicate a general-purpose, wide-ranging intelligence in AI continue. The CYC project aims to build an explicit representation of all the commonsense knowledge required to understand 1000 paragraphs out of an encyclopedia [132]. This project has yet to publish conclusive results and we have some doubts that it ever will.

[7] Even simple, apparently deterministic, systems can exhibit widely variable behaviour (especially under feedback) [98]. Chaos theory is more of a threat to certainty than atomic physics. It could be argued that seemingly distributing results about certainty in the sub-atomic world need not concern us out here in the macroscopic world. However, chaos theory concerns itself with macroscopic events we encounter everyday (e.g. the weather).

[8] In a Platonic rational world, designers should be able to infer their way to correct initial designs. Norman documents numerous examples where this is not the case, even for simple devices we have been building for hundreds of years (e.g. doors and ovens) [179]. Norman argues convincingly that only through trial-and-error (which he politely euphemises to "iterative design") can we generate good designs. The reader who doubts this conclusion is invited to re-consider their position the next time they (i) can't tell whether to pull or push a door open; (ii) can't work out which knob turns on which oven hot plate; or (iii) they install version N (N>1.0) of a piece of software.

[9]

> *Popular induction depends upon the emotional interest of the instances, not upon their number.* Bertrand Russell.
>
> *The death of a single Russian solider is a tragedy. A million death is a statistic.* Joseph Stalin.

More scientifically, Kahneman & Tveresky report experiments demonstrating that different expressions of the same problem can consistently bias the way people process that problem. In one study, physicians consistently choose one of two options according to the way a problem was framed. Both options were actually identical, but one was expressed in terms of absolute numbers and the other in terms of percentages. The problem was framed in terms of lives-saved or lives-lost. Physicians presented with the lives-saved frame were generally risk-avoiding; i.e. they elected to maximised the absolute number of lives saved. Physicians presented with the lives-lost frame were risk-seeking; i.e. elected to minimise the percentage of lives lost. Their general conclusion was:

> *The same decision can be framed in several different ways; different frames can lead to different decisions. ([118] p139)*

1.1.2. Modelling and Uncertainty

The experience with expert systems is that the process of building consensus between individuals or creating an explicit record of it in a knowledge base introduces biases/errors. Kuhn notes that data is not interpreted neutrally, but (in the usual case) processed in terms of some dominant intellectual paradigm [122]. Silverman cautions that systematic biases in expert preferences may result in incorrect/incomplete knowledge bases [235, 237]. Preece & Shinghal document fielded expert systems that contain numerous logical anomalies such as unused inputs, unsatisfiable conditions and unusable consequences [205][10]. These expert systems still work, apparently because in the context of their day-to-day use, this erroneous logic is never exercised.

The concept of "context-of-use" has become a key issue in knowledge acquisition research. Human "knowledge" appears in some social context and that context can effect the generated "knowledge". Phillips [192] and Bradshaw *et. al.* [17] describe model construction as a communal process that generates structure that explicate a community's understand of a problem. If the community changes then the explicit record of the communities shared understanding also changes; i.e. "truth" is socially constructed. Such an explicit expression of current beliefs may prompt further investigation and model revision; i.e. writing down models of "truths" can cause "truth" to change. We later document cases where, quite clearly, the generated model changed over time and did not seem to terminate on some idealised "true" model[11]. Agnew, Ford

[10] These anomalies are listed in section 3.3.3.

[11]

& Hayes summaries contemporary thinking in the history, philosophy and sociology of science as:

> *…expert-knowledge is comprised of context-dependent, personally constructed, highly functional but fallible abstractions [4].*

A model of some thing is different to that thing (the map is not the territory). Hence, the model may contain less information than the modelled thing and may behave differently in certain crucial situations. Thus the modelling process introduces errors into the representation. As to recognising a "true" a model, we agree with Karl Popper [203] that such a task is be fundamentally impossible:

- All "proofs" must terminate on premises; i.e. some proposition that we accept as true without testing. If we request proofs of premises, then we potentially recurse forever. It may be that we can terminate the recursion if the premises become self-evident. However, note that many "self-evident" premises have not stood the test of time (e.g. "the world is flat" or "the earth does not move").
- In terms of most human knowledge, a recursion to base premises is fundamentally impractical. For example, consider one individual trying to reproduce all the experiments that lead to our current understanding of atomic physics. Such an undertaking could longer than a lifetime and would be beyond the resources of most individuals (e.g. building a five kilometre long linear accelerator). Such a task has to be divided up and, sooner or later, our single researcher would have to *accept on faith* the validity of another researcher's statement that "while you were busy elsewhere, I did this, and I saw that. Trust me.".

1.1.3. Knowledge-Based Systems and Uncertainty

Knowledge representation theorists acknowledge that the knowledge inside experts systems is only an approximate surrogates of reality [17, 55, 260]; i.e. their accuracy is doubtful. O'Hara notes that some knowledge representation theorists still make occasional claims that their knowledge representation theory has some psychological basis. However, when pressed, their public line is that representations are models/ surrogates only [181][12].

Clancey argues the knowledge structures found during knowledge acquisition (frames, rules, etc) are structures created on-the-fly in response to the specifics of the situation in which they were elicited: the example being studied; the experts used; etc. That is, they have little/no isomorphism with structures present in an expert's information processing system. Clancey is silent on where these structures come from but hints that the substrate may be neural [39].

[12]

Feldman & Compton [47, 79] make a similar claim, arguing that our explicit KRs are not records of structures inside the head of an expert. Rather, the "knowledge" tricked out of an expert is a report customised to the specific problem, the specific justification, the expert, and the audience; i.e. like Phillips and Bradshaw *et.al.* , they argue that "truth" as expressed by human experts varies according to who says.

In summary, the extreme *doubting Thomas' position* is as follows. Attempts to write down human "knowledge" on paper or in a program:

- will reflect sub-optimum reasoning;
- will reflect certain biases in the way the domain is perceived by an individual;
- will reflect some consensual hallucination of a particular community;
- will not mirror the actual structures used in successful reasoning systems; i.e. people;
- will be probably inaccurate;
- ultimately, can't be proven correct.

1.1.4. Certainty

Just as extreme as Thomas' position is the opposite Platonist/Baconist approach which we will call the *Optimist position*. A Platonic optimist would expect to "see" the expert's constructs and simply write them down. One prominent knowledge optimist is Edward Feigenbaum whose pioneering work on expert systems lead to a view of knowledge acquisition as a "expertise-transfer" approach (which Feigenbaum poetically describes as "mining the jewels in the expert's head" [78], p104)[13]. Compton summaries this Platonic approach, which he strongly objects to, as follows:

> *The reductionist assumption that one should be able to dig deep enough to find primitive concepts and the relationships between them on which knowledge is built finds its origins in Plato's concept of archetypes. That is, that there exist (literally) archetypes for all the things in the world and the concepts we use… Proposals such as (expertise transfer) are essentially statements of belief that if the archetypes and relevant logical relationships can be found and manipulated intelligent thought can be reproduced. ([46], p280).*

Plato's metaphor for explaining human confusion was a cave where we built our feeble fires against ignorance. These fires threw light onto the archetypes of the things that were really true. Sadly, according to Plato, us poor cave dwellers can only see the dim shadow of the archetypes, flickering and randomly distorted by the wind on the fire and the bumps on the wall.

Medewar sarcastically summaries the Baconian perspective as follows :

[13]

> *…truth lay all around us- was there for the taking-waiting like a crop of corn, only to be harvested and gathered in. The truth would make itself known to us if only we would observe nature with that wide-eyed and innocent perceptiveness that mankind is thought to have possessed in those Arcadian days before the Fall - before our sense became dulled by prejudice and sin. ([150], p70)*

If the optimist's position was correct then the decline in belief of certain knowledge in the twentieth century[14] would not have occurred. In fact, given the rapid increase in scientific research in the last few decades, our Platonic optimist would have expected greater certainty as long lists of proven truths rolled out of our new research labs. Nevertheless, there must be some value in the optimist's position. If all human knowledge is as relative and inaccurate as Thomas tells us, then we would not have any basis for rejecting obvious absurdities. For example:

> *Scientists in Tasmania recently report a set of revolutionary studies on amphibian cognitive processing. Two samples of frogs were placed on a laboratory bench. Half the frogs had their legs amputated. In a statistically significant number of trials, after a researcher ran towards the bench shouting "Jump! Jump!", only those frogs with legs left the bench. Researchers concluded that amphibian understanding of English was stored in their legs.*

We believe that this story is ridiculous. Further, we can believe that we can tell this story to numerous people and share a chuckle since knowledge about frogs, legs, jumping, and Tasmanians seems constant amongst most humans. Thomas would be at a loss to explain how such consensual knowledge could arise[15]. Clearly, some consensual view of the world exists between people. Perhaps not all knowledge is a whim invented on the spur of the moment. Antibiotics really do save lives. Men really walked on the moon. This report is really being written on a complicated combination of hardware and software. If human knowledge is the poor tool that Thomas claims, then these technological advances would be very unlikely. As Agnew *et. al.* see it, the constructs we write down to represent human knowledge are grounded in something, and sometimes that something might even be true.

> *Some knowledge and expertise are more than disposable cultural myths or highly personal empirical or symbolic fabrications… Anyone who has carried out experiments… has not only experienced social context pressure but also felt the non-negotiable force of the constraints imposed by the ontic (real) world. Ontological reality manifests itself to use (sometimes quite emphatically) when we bump into some of the constraints that it imposes on our activities…and our beliefs. Some of the shadows on the wall of Plato's cave kicks back. [4].*

Like Agnew *et. al.*, we believe that not all knowledge is relative. Concepts that we have been working with for a while are now fairly well debugged. With time, inductive generalisation of the human experience has lead to a set of beliefs that accurately predict the future about certain things (usually). Predictions relating to the physical or basic

[14] See section 1.1.1.

[15] Although Thomas may be pleased to see that the reference to Tasmania may have confused non-Australian readers. Unfortunately for Thomas, these overseas readers may be able to derive a common interpretation

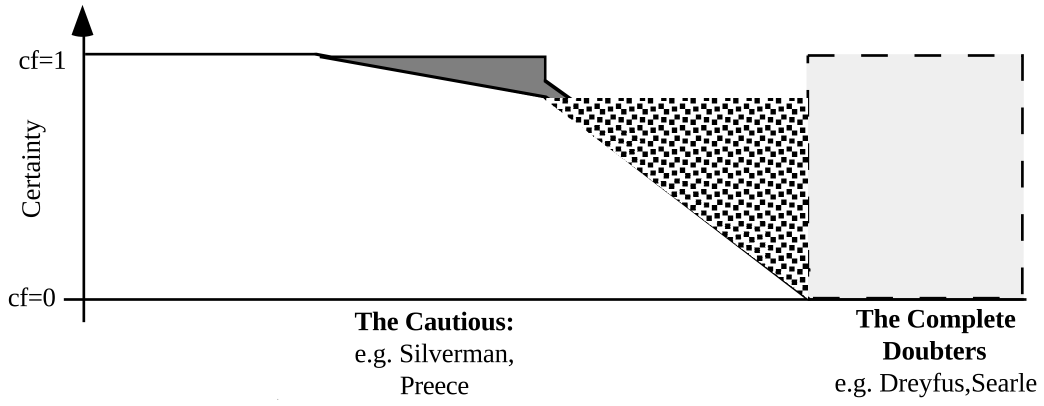
biological events that our Neanderthal ancestors encountered will probably be fulfilled. For example, we don't really doubt that we can predict what will happen when we drop something, stub our toe, or eat when we are hungry. Such old knowledge might even be "true", in a Platonic sense. Gaines & Shaw comment:

> *In a well-established scientific domain it is reasonable to suppose that there will be consensus among experts as to the relevant distinctions and terms- that is* objective knowledge *that is independent of individuals. However, the "expert systems" approach to systems development has been developed for domains where such objective knowledge is not yet available. [90]*

We agree with Gaines & Shaw that most of the knowledge entered into expert systems is new knowledge that is still being debugged; for example: how to best configure a computer; what constitutes a risky loan applicant; how to diagnosis biochemical disorders; what antibiotics we should prescribe; or how to best schedule a Space Shuttle mission.

1.1.5. Structured Testing

Sadly, human knowledge is not marked with labels indicating whether or not it is "old knowledge we can trust" or "new knowledge we are still debugging". In terms of writing knowledge bases, one reaction to this uncertainty is to abandon all attempts to codify human knowledge. This is the reaction of Dreyfus, who argues that the context-dependant nature of human knowledge makes it fundamentally impossible to codify symbolically [71]. Searle takes a similar stand, claiming that the only device that can replicate human intelligence is another human [229]. Figure 1.1 places Dreyfus' and Searle's position into a continuum of other researchers.



However, we believe that Searle and Dreyfus are taking an unnecessarily extreme position[16]. Knowledge may be wrong, in some absolute Platonist sense, but still suffice. For example, Newtonian mechanics was superseded by relativity but is still useful for those bits of the universe travelling much slower than the speed of light. Many researchers argue for a process of *structured testing* for the generation of sufficing knowledge:

- Popper argues the authority of an idea arises from its ability to survive active attempts to refute it [203]. To Popper, all ideas should be aggressively attacked, as a matter of routine.
- Agnew *et al*, reacting to a rising tide of knowledge relativism, argue for continued application of the scientific method since it "has evolved and institutionalised mechanisms for evaluating the durability of knowledge constructions" [4]. That is, the scientific method is like a life-insurance policy for an idea. To insure survival, ideas have to be developed according to certain criteria (e.g. criticised via peer review).
- Preece and Zlatereva describe test programs based on the logical structure of rule-based expert systems. Price's tools detect anomalies in those structures (e.g. circularities) [204] while Zlatereva's tools analyse that structure to generate a test suite which will exercise all parts of the rule-base [272].
- Compton proposes a structured patch environment for rule-based systems. Applying the heuristic *Si fractum non sit, noli id reficere* (if it ain't broke don't fix it), Compton's ripple-down-rule environment forbids re-organising rules that have proved their utility in the past. Instead, if a new case is handled inappropriately by the current rules, new rules are added onto existing rules that patch the error in the context of it arising. This seemingly-naive technique has proved its worth in the domain of biochemical interpretation [206] and has out-performed techniques based on apparently more-sophisticated representations [159, 187][17].
- Gaines & Shaw explore techniques for resolving conflicts in terminology. The conceptual systems of different experts are explicated and compared using a technique called entity-attribute grid elicitation [90][18].
- Boose *et al.* describes group decision support environments containing suites of tools combined to form a knowledge acquisition environment [14]. Boose *et.*

[16] For an illuminating public debate between Searle and defenders of the AI program, see Hofstadter and Dennett's review (p373-382 of [105]) of Searle's position [229], and the subsequent argument [67, 230, 231]. Our own reading is that Searle can only repeat his initial eloquence, while Hofstadter and Dennett can marshal a broader and more substantial case.

[17] For more on ripple-down-rules, see section 2.3.7.

[18]

al.'s system focuses on the development of models of the group decision support process[19].

- Silverman advises that attached to an expert system is an expert critiquing system which he defines as:

  > *…programs that first cause their user to maximise the falsifiability of their statements and then proceed to check to see if errors exist. A good critic program doubts and traps its user into revealing his or her errors. It then attempts to help the user make the necessary repairs. [237]*

  Silverman divides an expert critiquing system into (i) a *deep model* which can generate behaviour; (ii) a *differential analyser* which compares the generated behaviour with the expected behaviour; and (iii) a *dialogue generator* that explains the errors and assists in correcting them.

1.1.6. Implementing Structured Testing

Only Compton, Silverman and Zlatereva discuss testing via assessing the possible inputs-outputs of their systems.

- Popper and Agnew *et.al* discuss general principles for human exploration. They do not propose automatic tools that support the process they advocate.
- Preece's preferred testing tools assess a knowledge base according to internal syntactic criteria (which we will call verification). Such an internal syntactic test may be irrelevant to the issue of a model's ability to reproduce known behaviour[20].
- Gaines & Shaw focuses on identifying and resolving conflicts in the meaning of individual terms, not on conflicts in the semantics of the models built using those terms as primitives. A model-level conflict detection requires a execution module.
- Boose's *et al* system lacks an execution module for the generated models as part of the group decision support environment. Boose *et al* assume that once the group's mode is elicited, it will be subsequently exported into an executable form.

However, in terms of general engineering principles for the construction of structured testing environments, the performance modules of Compton, Zlatereva, and Silverman are incomplete:

- Compton's design is optimised for maintenance of propositional rule-bases only. Experts cannot browse and recognise their models inside the patch tree. This

[19] For more on Boose *et al*'s work on decision support systems, see section 7.2.3.

research began as an experiment in a maintenance environment where experts could browse and freely modify the knowledge[21].

- Zlatereva's tools can generate the *inputs* for a test suites that will exercise an entire rule-base. However, an expert still has to decide what *output* is appropriate for each generated input. This can introduce a circularity in the testing procedure. After an expert describes their world-view in a model, that same expert will be asked to specify the results of certain inputs. If the expert then uses the same model to predict the output, then they would be using a potentially faulty model to generate a potentially faulty prediction about the output[22]. Our preferred approach is for the input-output test pairs to be generated totally separately to the current model; e.g. from real-world observations of the entity being modelled in the rule-base.
- Silverman's research seems to be aimed at an implementation-independent analysis of the process of "critiquing" a program. His focus seems to be on defining "critiquing" as an add-on to existing systems, not as a built-in that is fundamental to the whole KBs life cycle[23]. We believe that while this approach is useful, a more extensible approach would to change the structure of knowledge-bases systems such that critiquing is built into the system. In the case where the design of the system can be altered to integrate a testing module, we believe that the built-in approach is superior since built-in critics could guide the knowledge acquisition and maintenance process.

1.1.7. Goals

Model review is a two-stage process: *fault + fix*. Here, we explore faulting models only. We are sensitive to criticisms that this work is incomplete without a working "fix" module. In our defence, our experience has been that experts have no problems with generating any number of new models/ revisions to existing models. However, assessing those new models is very difficult. Hence, our goal: general implementation principles for *deep models* and *differential analysers*. Mahidadia explores *dialogue generation* in this domain using inductive logic programming [137, 140][24].

Our goal is somewhat ambitious in that a generic testing module implies a generic execution engine. We will argue that an abductive inference procedure, with customisable inference assessment operators, is such a device[25]. This claim will be

[21] See chapter 2.

[22] For more on Zlatereva's work, see section 3.3.2.

[23] Expert critiquing systems are discussed further in section 7.2.4.

[24] For more on Mahidadia's work, see section 2.3.6.

[25] For more on inference assessment, see the discussions on the *BEST* operator in section 2.3.5.4. For more on

supported by a literature review that maps our validation-as-abduction process into a surprisingly wide range of knowledge-based systems (KBS) tasks[26]. We will discuss general implementation techniques for such a module, supply a detailed design, experimentally examine its complexity, and uncover computational limits to doubt. Given our belief that testing is an important part of building knowledge bases, we will also argue that these limits to testing are also the limits to knowledge acquisition. Further, we will argue that numerous "knowledge level" tasks such as diagnosis, prediction, and monitoring are actually blurred reflections of a single, more basic, process. That is, our generalised testing architecture will help to clean up confusions at the knowledge level.

1.2. Statement of Thesis

This research is a generalisation of prior work by Feldman and Compton on *qualitative hypothesis testing* [79, 80]. We will argue that:

> *Generalised qualitative hypothesis testing is a essential, simple, customisable, practical, general, and radical method of developing and executing a wide range of knowledge tasks.*

1.3. Chapter Plan

The subsequent chapters expand the underlined sections of the statement of thesis:

Chapter 2: Describes the historical predecessor to this work. QMOD/JUSTIN was a structured hypothesis testing environment for qualitative models in data-poor environments.

Chapter 3: Argues that a process like QMOD/JUSTIN is an essential extension to modern knowledge based systems (KBS) practice.

Chapter 4: Discusses the implementation of a generalised hypothesis tester. At its core, it is a simple process (called `Core`).

Chapter 5: `Core` executes over a very low-level structure. Chapter 5 describes a customisable layer on top of `Core` that is suitable for knowledge acquisition.

Chapter 6: The practical utility of `Core` is demonstrated here via three experimental studies.

Chapter 7: Formally, `Core` is abduction and abduction is general to many KBS tasks. Chapter 7 explores the use of the `Core` validation algorithm as an inference engine.

Chapter 8: Given that chapter 6 demonstrated that `Core` can act as a test engine and chapter 7 demonstrated that `Core` can act as an inference engine, chapter

[26]

8 argues for replacing a wide range of knowledge tasks with abductive architectures such as `Core.` Given the current interest in the KA community for high-level abstracted representations, proposing the use of a low-level routine such as `Core is` a radical proposal.

Chapters one and three present the motivation for this work. Chapter two describes the related work that spawned this research. Other related work is discussed through out this report, the major one being the abductive literature reviewed in chapter seven. Chapters four, five, and six describe the implementation and experimental work that is the basis of this work. Chapter seven is the most theoretical and explores the equivalence of this work to other knowledge representation research. Chapter eight is somewhat provocative and argues that the proposal presented here is not only equivalent, but better, than the knowledge level modelling proposal currently favoured by the knowledge acquisition community.

Certain chapters can be read in relative isolation to the rest of the report. For details on how to build an generalised qualitative hypothesis tester, see chapter four. For a "symbol level" critique of current trends in "knowledge level" modelling, see chapters seven and eight. For relaxation, read chapter ten.

Non-technical readers may find chapters one, three, eight, nine and ten the most approachable.

Each chapter starts with *N* (*N*> 1) quotes. Quote *N=1* are historical examples of a classic error in reasoning. Chapter quote *N* (*N>1*) set the tone of that chapter.

1.4. Publications

Portions of this work have been published previously [152-156, 158-161]. Note that the terminology used here supersedes the terminology defined previously.

1.5. Notation

Chapters two and four use a Pascal-ish pseudo-code notation, with certain additions:

- Some polymorphism; e.g. a special non-typed records called `any`; variable procedure names called `methods` that can be bound at runtime; a generic number type that works for both integers and reals
- Efficient `bitstring` processing;
- An explicit `return` statement in functions.
- Functions with one statement do not require `begin-end` statements.
- The ability to define type/procedure/function above or below the definition of the types/procedures/functions that it uses;
- The use of `--` to denote comments;

- Built-in `list` types which can grow to arbitrary length, hold `any` type, respond to `size(list)` and `average(list)` (the latter returns the mean of the numbers in the `list`).
- Lists auto-initialise to [], sets to `{}`, and integers auto-initialise to zero.
- $\{X\}$ and $[X]$ denote sets and lists respectively containing one element X.
- If X is a set or a list, then $X + Y$ adds the item Y to X. If X is a list, Y is now the last item in that list. If X and Y are sets or lists, then $X \cup Y$ is the combination of the two. In the case of sets, the result is guaranteed to have no repeats. In the case of lists, Y is appended to the end of X. If either one is empty, then the other is returned unchanged. If both are empty, then the empty set/list is returned. If X and Y are sets or lists, then $X \cap Y$ is the intersection of the two. If X is a set or list, then $X + Z$ denotes adding item Z to X. In the case of lists, it is appended at the end.
- "For" loops that can iterate over all items in a lists or all subsets of a list.
- Automatic garbage collection (so we don't have to `free`).

1.6. Acknowledgements

This work was supported partially by an Australian Research Council Grant #A49030091.

My supervisors each had their special role to play. A/Prof. Paul Compton fanned the fire when the spark was dim while A/Prof. Claude Sammut stamped out the forest fires[27]. They also performed several miracles regarding funding, for which I will be eternally grateful and promise never to tell a soul.

Dr. "Enrico" Coiera (HP Labs, Bristol), Dr. Ashwin Srinivasan (Logic Programming Group, Oxford University) and Dr. Cindy Mason (AMES Research Labs, NASA) were my ambassadors to the international AI community. Thanks to their friendship, I had an unusually close contact with a wide range of AI researchers around the world. Also, "Enrico" (who will always be Ric to me) deserves extra thanks for demonstrating that PhDs were not only for other people.

I owe the *QMOD* long-suffering domain expert, Dr. George Smythe of the Department of Chemical Pathology, St. Vincent's Hospital, Sydney, a large debt and marvel at his endless patience.

Special thanks to Dr. Max Kanovich of the Moscow Humanitarian University. At a time when I was suffering from a bad dose of methodological constipation, Mad Max arrived

[27] Though sometimes the experience was like playing with matches round a petrol tank and walking to the South

and convinced me that theorem provers were just programs which real people like me were allowed to implement.

Dr. Clarke Quinn, Dr. Ray Lister, and Dr. Bob Cohen were kind enough to read various versions of this thesis in great detail, and offer many useful suggestions. Bob's reading was the most arduous since it necessitated reading the penultimate 200-page draft.

Dr. Michael Cameron-Jones politely corrected a glaring flaw in our treatment of explaining "steadies".

My fellow PhD students all helped, in their own ways:

- Graham Mann's contribution was as indescribable as it was unique, enthusiastic, sometimes inspirational, sometimes confusing, but definitely top of the list.
- Ashesh Mahidadia and Maria Lee made significant and timely contributions to the many features of this work. I wish them well in their related research.
- Hugh Clapin tried to keep me honest about the philosophy (though I fear he did not totally succeed: my fault, not his).
- Dr. Andrew Taylor (who, in his heart, is still a PhD student) took the time to explore my work. His surgical precision was very useful.
- Grasp Heaven (a.k.a. Byeong Ho Kang) was always cheerful and polite, until I taught him to swear in Australian. I hope his family can forgive me.
- And to the rest (Phil, Ashley, Michael, and the rest of the AI Lab gang), many thanks.

The Newcastle University Departments of Information Science and Computer Science helped the latter stages of the PhD with coffee, part-time work, and an environment where I could complete this work. Special thanks to Dr. Jo Caldwell for excusing me from a mountain of first-year marking at a time when my thesis needed a lot of attention.

Extra thanks to Rowena Claire-Noble Yatsamatzu for coping with my strange work practices and long-hours. See you in eight years' time.

I've saved the very best for last. Clare Morgan helped, just by being there. While my fellow PhD students were going crazy in their own strange ways, Clare kept me balanced (moderately) and sane (relatively) during the last two years of this process. She was always good for company, love, support, and the offer of a gin. Well, if you're having one…

They couldn't hit an elephant at this dist… : Last words of General John Sedgwick, Battle of Spotsylvania, 1864.

The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model it is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work: John von Neumann.

2. Hypothesis Testing

Experiments with qualitative hypothesis testing (QMOD/JUSTIN) were the historical pre-cursor to this work. This chapter discusses qualitative and quantitative hypothesis testing and introduces compartmental modelling (a technique we use later).

We begin with a discussion on quantitative hypothesis testing and give an example of how we would develop a mathematical model suitable for quantitative hypothesis testing via compartmental modelling. This mathematical approach requires numerous measurements of the domain in question. We then introduce QMOD/JUSTIN: an experiment in adapting compartmental modelling to poorly-measured domain.

2.1. Quantitative Hypothesis Testing

Quantitative hypothesis testing is a well developed statistical technique for testing that two sets of numbers are similar. If a domain supports a mathematical model, then quantitative hypothesis testing can be used to generate a set of numbers representing the behaviour of a model. This output can then be compared to measurements from the entity being modelled. A model passes this quantitative hypothesis test if the measurements are statistically the same as the model output.

2.1.1. Example

For example, consider a drug injected into the blood. The level of the drug in the blood decreases as (i) it diffuses into body tissues and (ii) the drug is cleared by the liver. Also, (iii) the drug in the blood tissues may diffuse back to the blood. We can model this system using compartmental modelling [148]. Compartmental models utilise the principal of conservation of mass and assume that the sum of flows of substance in and out of a compartment must equal zero. Flows are typically modelled using a time-dependant expediential function since the rate of flow is often proportional to the amount of stuff in the compartment. We can model our drug with the three compartment system of Figure 2.1.

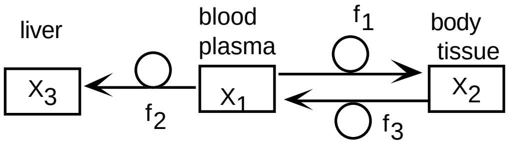


Figure 2.1 *Representing drug levels in the blood using a three compartment model. Example from [97] p183-185.*

The functions f_i model the flows between the compartments using three constants: k_1, the rate of flow of the drug into the tissues; k_2 the rate of clearance by the liver and k_3 the rate of flow of the drug into the blood plasma. Applying conservation of mass, we get A, the matrix for the system:

$$A = \begin{pmatrix} \frac{dx_1}{dt} = -(k_1 + k_2)x_1 + k_3 x_2 \\ \frac{dx_2}{dt} = k_1 x_1 - k_3 x_2 \end{pmatrix}$$

This systems characteristic equation has roots p calculated as follows:

$$\det(A - pI) = \begin{bmatrix} -(k_1 + k_2) - p & k_3 \\ k_1 & -k_3 - p \end{bmatrix} = 0$$

$$\therefore p_{1,2} = \frac{-(k_1 + k_2 + k_3) \pm \left[(k_1 + k_2 + k_3)^2 - 4k_2 k_3\right]}{2}$$

Since this is a well-measured domain, we have values for the initial conditions of this model: $X_{1(t=0)} = C$; $X_{2(t=0)} = 0$; and the flow rates: k_1 =0.5; $k_2 = k_3 = 1$. Therefore, p_1 = -1/2 and p_2 = -2. Given this knowledge of the roots, we can re-express our differential equation as follows:

$$\begin{pmatrix} dx/dt = Ax \\ x(t = 0) = x_o \end{pmatrix} \Rightarrow x_i(t) = \sum_{i=1}^{N} c_i e^{p_i t} = De^{-t/2} + Ee^{-2t}$$

Using our initial conditions again, this equations becomes:

$$D + E = C, {}^{dx_1}/_{dt} = {}^{-3C}/_{2} = -D/2 - 2E$$

$$\therefore D = {}^{C}/_{3}, E = 2C/3$$

$$\therefore x_1(t) = {}^{C}/_{3}\, e^{-{}^{t}/_{2}} + {}^{2C}/_{3}\, e^{-2t}$$

Figure 2.2. graphs this function for C = 0 to 10000 and T = 0 to 3. We see that blood plasma levels x_1 varies from 0 to 6000 and degrades smoothly as a simple exponential function.

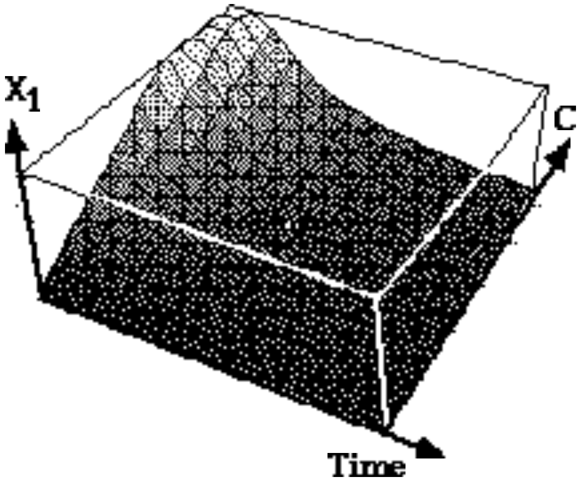


Figure 2.2. *Change blood-levels for X_1 from the model of figure 2.1.*

2.1.2. Assessment

Our example seems impressive. We have been able to deduce a detailed mathematical model suitable for statistical hypothesis testing from a seemingly simplistic approximation to human physiology (three compartments). However, recall the amount of data we required:

- 3 of the 3 flow rates (100%)
- Measurements of 2 of the 3 compartments (67%) at the same time interval.

Further, in order to assure statistical significance, we would have to make many such measurements of the entity being modelled. In many poorly-measured domains, this is not possible. Consider, for example, neuroendocrinology: the study of the interaction of glands and nerves. Obtaining values for certain chemicals within the body is not as simple as, say, attaching a volt meter to an electric circuit:

- In one extreme case, 300,000 sheeps brains had to be filtered to extract 1.0 milligrams of purified thyroptin-releasing hormone [121].
- In the usual case, delicate measurements have to be made by skilled staff using expensive equipment. Some of the values measured are in the pico-MOLE range (10^{-12}).

Measurement in this domain can therefore be an expensive process and not all entities are fully measured. For example, in the QMOD study (described below), data on a sketch of a model of glucose regulation [240] was collected from six journal articles. In all, none of the flow constants were known and only 39% of the compartments were measured and not all at the same time interval[28].

The problems associated with quantitative modelling in poorly-measured domains encouraged Feldman & Compton to explore qualitative approaches.

[28] The Smythe '89 data set is shown in Table 6.5 in section 6.2.1. Only 141 values are available for the 12

2.2. Qualitative Hypothesis Testing: QMOD

The QMOD/JUSTIN project was Feldman & Compton's experiment in creating "live" knowledge about scientific publications [79, 80]. That system is introduced below using their terminology of [79, 80]. The precise semantics of these two systems was the focus of this research and will be detailed in subsequent chapters.

2.2.1. Motivation

Modern science produces mountains of paper. There are now at least 2000 medical scientific articles published per week in internationally recognised journals [79]. In the medical literature alone, there now over a 1000 on-line databases (e.g. MEDLINE) with half a billion entires (1990 figures, from [250]). Clearly, without automatic tools, no researcher could ever hope to keep up-to-date with it all.

In their current form, this mountain of published material is "dead" knowledge. For example, if a researcher in Britain publishes a paper that describes a model that subtly disagrees with a publication from Argentina, we have no automatic method for detecting the inconsistency:

- The current generation of on-line systems support only a small number of syntactic indexes, usually only on parts of the paper such as the abstract.
- While a paper may discuss some new model, or proposes an edit to an existing model, we can't call up that model, execute it, see what behaviour it generates, and note how any differences between the behaviour of different models proposed for the same domain.

The QMOD dream was the creation of an international electronic book used by researchers around the world were to record all their experimental data and hypothetical models. The book was meant to "wake up" and complained if someone wrote down something that disagreed with its contents. When a new model is proposed, it could be entered into the knowledge base and checked for consistency with both the existing knowledge base and data.

The QMOD philosophy was based on a division of labour between people and computers:

- Human beings use their insight to formulate new models. This is an imagination-intensive exercise that is (currently) best down by people.
- Computers check the consistency of the proposed new models. This is a clerical-intensive exercise that is best done by computer.

Rather than automating creativity, QMOD aimed at supporting human insight. Humans don't need know when their theory is right. Such a discovery prompts zero activity except, perhaps, some patting on the back or a round of drinks at the bar. However, the discovery of an error can prompt frenzied activity. Tedious, global searches of large knowledge bases in order to test the applicability of data to a model is best implemented by computer.

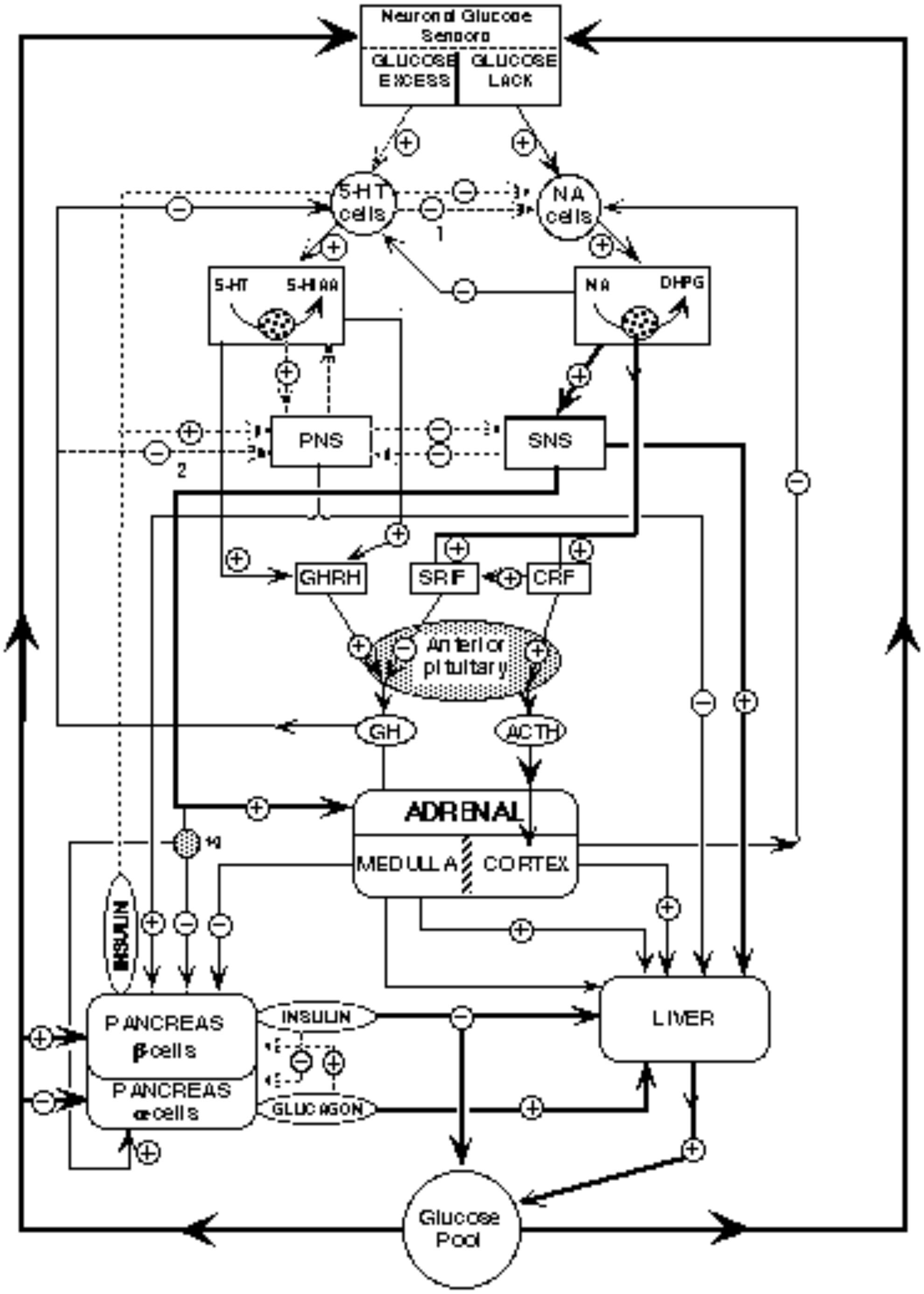


Figure 2.3: *The vague causal diagram from [240] used by Feldman & Compton as the input to their experiment .*

2.2.2. QMOD Components

2.2.2.1. Graphical Editor of Qualitative Structures

Feldman & Compton started with the vague causal diagram of Figure 2.3 which they derived from a review paper on glucose regulation [240]. Figure 2.3 was converted into a qualitative compartmental model; i.e. a compartmental model with no numbers attached. A fast-entry format for scientific models was defined and implemented using a Hypercard-based [8] point-and-click editor. Figure 2.3 became 13 models, one of which is shown in Figure 2.4.

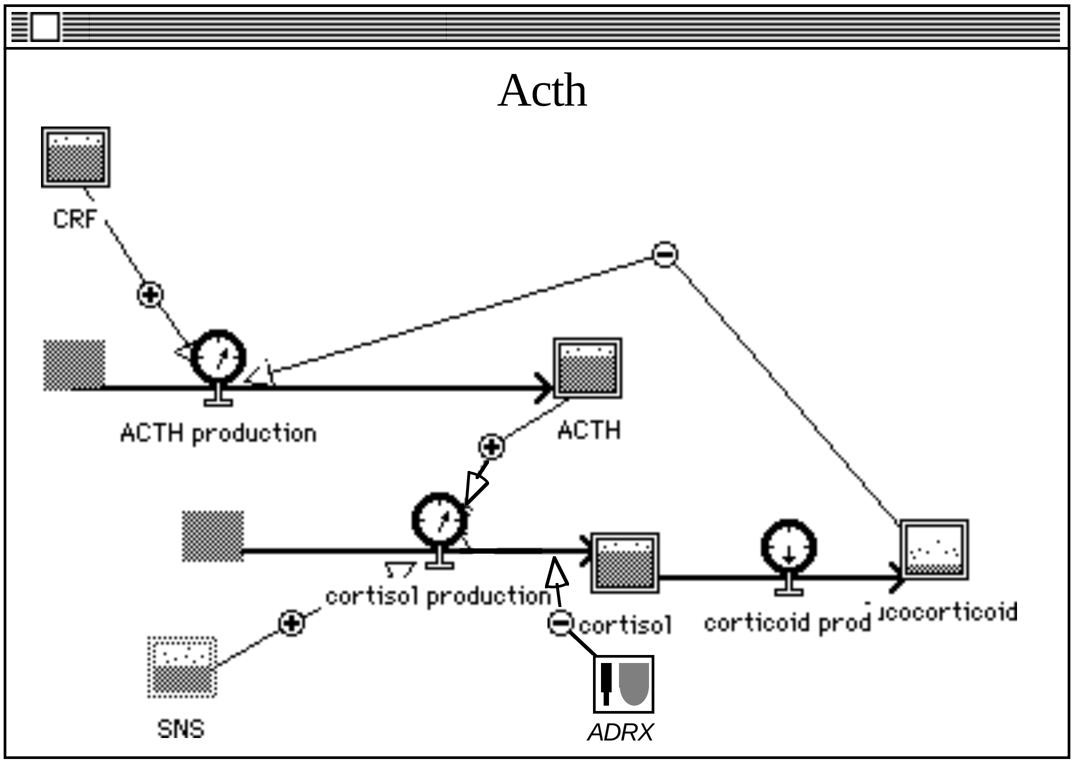


Figure 2.4: *A QMOD screen. In part, this figure is "saying" that (i) acth promotes cortisol production; (ii) glucocorticoid inhibits acth production; (iii) an adrenalectomy (adrx) disables cortisol production by removing the input from cortisol production. The two unnamed boxes (left-hand-side) are "sources"; i.e. unlimited streams of "stuff".*

Models of neuroendocrinology were created using a graphical syntax taken from compartmental modelling, with some changes:

- Causal links were added between compartments and flows or compartments and compartments. These links were of two kinds- *plus*/ *minus* nodes that respectively increased/decreased the level of stuff in a compartment or a flow rate.
- Facts that represent experimental perturbations of the model. These perturbations either enabled (added) or disabled (removed) links.
- No numeric data was required for the model. QMOD's models were qualitative. Model construction was not inhibited by a lack of numeric data (as is the case with quantitative compartmental models).

2.2.2.2. Database of Observations

A database of observations relating to entities in the QMOD models was maintained. Each observation had a "context" *CXT* ; i.e. an associated set of facts. Modelled entities could be measured in various contexts including *control*, the empty set context.

For example, the Smythe '87 study [239] made observations relating to connection between serum corticosterone and neuro-noradrenergic activity (measured as the ratio of noradrenaline to its post-cursor, 3,4-dihydroxphenylethethyleneglycol (DHPG)). Neuro-noradrenergic activity (nna), serum corticosterone (cortico) and adrenocorticotropin (ACTH) were measured in lab rats in different contexts:

- the *dex* group, which were given an injection of dexamethasone (*dex*) at 100 mg/kg;
- the *cold swim* group, which were subjected to a two minute swim in a bath of ice cold water;
- the *cold swim plus dex* group, which were subjected to both a cold swim and an injection of *dex*;
- the *control* group, which received no treatment at all.

Some of the observations from Smythe '87 are shown in Table 2.1.

	Context			
Value	**{} = control**	**{dex}**	**{coldSwim}**	**{dex , coldSwim}**
nna (no units)	0.122	0.105	0.210	0.246
serum cortico (nmol/L)	129.0	11.3	1232.0	32.8
serum acth (pg/ml)	89.0	0.0	240.0	0.0

Table 2.1: *Observations from various experimental contexts (from Smythe '87 [239]).*

2.2.2.3. JUSTIN: Model Error Detection

QMOD models were assessed according to their ability to reproduce known behaviour (as expressed in the database of observations). Pairs of contexts were compared. Changes in the measurements were called the effects *FX* and changes in the context were called the causes *C*. The JUSTIN model error detection component (short for *justification-in-context*, written in Prolog) sought explanations for the effects in the terms of the changes in the contexts; i.e. some pathway through the model starting at a cause and ending at an effect. If such pathways traverse compartments for which we have no measurements, we *assume* a value for that compartment that is required for the pathway. If after (i) making every assumption possible; and (ii) generating all pathways possible, a model still cannot used to explain changes in the observations, then the model was deemed faulty.

For example, returning to Table 2.1, when comparing CXT_1 = *{Cold Swim}* with CXT_2 = *{Dex}*, causal pathways would have to start on one of *C* = {*ColdSwim/less,* *Dex/more*} and end on one of the effects *FX = {nna/less, cortico/less, acth/less}*. In between: (i) compartments can be assigned either "less" or "more" as required by the explanation, providing that assignment does not conflict with known causes/effects; (ii) compartments can use any of the arcs provided by the expert via the Hypercard editor. If some link was conditional on the presence or absence of an experimental perturbation, then using that link was conditional on that perturbation appearing in *C*.

Pathways can contain mutually exclusive assumptions (e.g. *cortisol production/more, cortisol production/less*). Therefore, once all possible pathways are generated, they had to be "resolved": i.e. separated into compatible subsets (which we call *worlds*). Each world was then explored to find the number of effects it could explain. JUSTIN reasoned generously; i.e. it would return the world(s) that *covered* (i.e. explained) the greatest number of effects. If cover < 100% then JUSTIN reported a faulty model.

2.2.2.4. Non-Cyclic Pathways

While QMOD theories can be cyclic (see Figure 2.4: *cortisol* → *corticoid prod* → *glucocorticoid* → *acth production* → *cortisol production* → *cortisol*), JUSTIN generated pathways are non-cyclic. Cyclic explanations are required for *time-series data*; e.g. *X* went up, then it went down, then it went up again. However, JUSTIN cannot generate such cyclic explanations. If such cycles were permitted, then JUSTIN's search for all possible pathways would enter an infinite loop.

QMOD/JUSTIN could be used for such time-series data. For problems that require such cyclic explanations over T time intervals: (i) use one new compartment for each combination of known compartment plus time-stamp; (ii) add a *time_increment* flow that represents a compartment at time T connecting to a compartment at time $T+1$.

The practicality of this proposal for proofs for large T is an open-research issue. Subsequent research showed that QMOD/JUSTIN runtimes are apparently worse-than-cubic and potentially exponential on model size[29]. The model required for N compartments measured over T time intervals would be of size $N*T$ and have runtimes $O(X^{N*T})$. QMOD/JUSTIN is hence indicated for domains where *either*:

- The model is non-cyclic.
- The model is cyclic, but behaviour does not include time-series data and experts do not require time-based explanations (e.g. the QMOD test data).
- The model is cyclic and behaviour only includes a small number of time intervals.
- The model is cyclic and behaviour includes many of time intervals, and the processing can be somehow heuristically culled. The general QMOD/JUSTIN processing is the generation of all possible explanatory pathways. However, in

[29]

special-case domains where an non-exhaustive approach will suffice, then we can cull some of this processing[30].

2.2.3. Experiments with Qualitative Hypothesis Testing

QMOD/JUSTIN analysed Smythe '89: a summary paper on glucose regulation [240] and 343 data points collected from six related research papers. The summary model included 27 nodes, and 82 links (and included figure 2.4). Causal explanatory pathways could not be generated for 109 of the data points (32%). The types of inconsistencies included clerical errors in translating models into the representation. Some of the inconsistencies were due to deliberate simplifications of the model by the researcher. However, the most important result was that the norepinephrine data in hypothyroid rats who had been given an alpha-2 adrenegeric blocker could not be explained. This was a novel finding that the authors of the [242] research paper were not aware of. The authors of [242] had only considered the effects of the alpha-2 adrenegeric blocker on hypothyroid rats rather than effect of hypothyroidism on alpha-2 adrenegeric blocker treated rats. That is, they had not considered the cross-experiment data comparisons. Although the data was highly statistically significant, the cross-comparison was not made since the authors were primarily interested in stress, not hypothyroidism. They therefore studied the effects of stress in the presence of hypothyroidism, to see whether or not the same mechanisms were operative as in other stress situations. The reverse comparison looks at the effect of hypothyroidism in the presence of stress, a question that the authors were not addressing. The result is of importance since it suggests that the interaction between serotonin and norepinephrine described in [242] will have to be relocated. This represented a major re-organisation of the [240] model and to our understanding of the interaction between norepinephrine and serotonin.

Our own subsequent study corrected some features of the Feldman & Compton study to increase the inexplicable percentage from 32% to 45%[31]. Another smaller study [153] found faults in another published scientific theory (Smythe '87 [239][32]).

Apart from the insights into neuroendocrinological models, the above result is interesting for several reasons:

- The faults detected by QMOD/JUSTIN were invisible to existing model review techniques in neuroendocrinology. All the analysed models and data were taken from international refereed, journals. 32-45% inexplicable data seems surprisingly high for models that have run the gauntlet of international peer review. We will later find that the complexity of the QMOD/JUSTIN inference

[30] For examples of non-exhaustive domains, see chapter 7.

[31] See section 6.2.3.

process is very slow, perhaps even exponential. It is therefore no surprise that human beings, with their limited short-term memory [175], do not completely test their models.

- Significantly, this study faulted a model using the data published to support that model. Clearly, human researchers do not rigorously explore all the consequences of their observations (perhaps since the process is so computational complex). The results of the QMOD/JUSTIN suggest that waiting in all the publications in the books in all the libraries around the world is a backlog of *extra inferences* that we could make about existing knowledge, without having to perform expensive further experimentation.

2.3. QMOD: Related Work

This section describes research closely related to QMOD/JUSTIN. Some of the references discussed here were not known to Feldman & Compton at the time of their 1989 research.

2.3.1. Active Documents

Swanson shares the QMOD/JUSTIN dream of an active document repository [250]. He describes one study that found extra inferences hidden within existing publications. Texts were manually examined for syllogisms. If text 1 supplied $A \rightarrow B$ and text 2 supplied $B \rightarrow C$, then Swanson makes the extra inference that $A \rightarrow C$. Some non-trivial examples of this described: e.g. (i) fish oil can help Reynaud's disease; (ii) magnesium could benefit migraine sufferers; and (iii) arginine intake assists aging patients with their declining levels of thymic function and protein synthesis.

Swanson's approach emphasises the use of existing texts, which implies a manual processing of that material. Until the day when natural language processing research matures sufficiently to generate active models from such texts, these texts will be unable to automatically generate behaviour. Hence, while we find his results pragmatically useful, we believe his approach to be limited and their scalability unlikely.

Executable documents are the focus of the ROUNDSMAN system [219]. ROUNDSMAN is a publication-centred tool for augmenting a physician's reasoning. The salient details of a patient's case are matched against cases stored in published medical literature represented as frames in the ROUNDSMAN knowledge base. A comparison is made between the case presented and the type of patients mentioned in the trials used in the literature. Treatment is critiqued based on the trials. Trials are assessed according to how close they are to the actual patient.

Unlike QMOD, the ROUNDSMAN system does not attempt to model the underlying physiology of the domain. The internal knowledge structures of ROUNDSMAN are declarative descriptions of the publications and pointers to related publications. The system has no causal knowledge of disease processes. In essence, ROUNDSMAN is a representation of the *discussion* and not the *domain* of the medical research literature. ROUNDSMAN's critiques of a clinicians plan is made with respect to the knowledge base. No validation tools are proposed for this knowledge base. Hence, ROUNDSMAN is not a tool for hypothesis testing.

2.3.2. Model Anomaly Localisation

Darden [53] discusses theory anomaly localisation based on an analysis of the development of genetic theory in the early part of this century. While it was not their intention, the study also demonstrated the central role of causal links[33] in model anomaly localisation.

The technical appendix to the Darden study describes how their theory was represented in a system called FR. While the FR representation was useful for structuring a complicated domain, most of the architecture was not needed for the anomaly localisation. The essential part of the implementation required for the localisation process were the causal links between parts of the theory (modelled as "function frames"). Anomaly localisation was a process of walking backwards from the final state back towards the initial state, inquiring at each point whether the intermediate state had been entered. Later versions of the program, as yet unpublished, are more sophisticated and used more of the FR architecture. Entities within the domain are bundled into groups (using functional knowledge) and anomaly localisation proceeds by groups, rather than by mere entities [169].

The goal of the program used in the Darden study was to illustrate how a functional representation such as FR could yield a systematic generation of possible faults that could be fixed in a process of redesign. That is, unlike our work, they were exploring an existing representation rather than seeking the minimal architecture needed for model refutation. At most, we could argue that the Darden study demonstrates that in terms of model revision, the useful features of a representation are the causal links between entities. At the very least we observe that validation does not fall out straight away from all knowledge representations, but requires some additional architecture.

2.3.3. QR: Qualitative Reasoning

Mainstream qualitative reasoning (QR) focuses on the processing of systems which are (i) piece-wise well-approximated by low-order linear equations or by first-order non-linear differential equations; and (ii) whose numeric values are replaced by one of three

[33]

qualitative states: up, down, or steady [263]. Differential equations in the format of (ii) are called qualitative differential equations (QDEs) [111].

A fundamental property of qualitative systems is their *indeterminacy*. Consider two influences (I_1 = up) and (I_2 = down) on a variable X . Lacking quantitative information about the relative size of I_1 and I_2 , we cannot determine if X goes (i) up, (ii) down, or (iii) remains steady since the I_1 influence cancels out I_2 . The three alternatives must all be considered and processed in separate worlds. When extended over several levels in a network, this can lead to an intractable branching of behaviour. Meta-knowledge can be used to tame some of this indeterminacy. For example, the Waltz filtering of the QSIM system[34] ruled-out a transition of the first derivative of a variable from increasing to decreasing without first going through a zero state. In practice, however, many possible behaviours will still be generated [86] and must be somehow handled by the program calling the qualitative simulator.

Initially, two qualitative ontologies were proposed: DeKleer & Brown's parts-based CONFLUENCES [60, 61] and Forbus's process-based qualitative process theory (QPT) [81, 84]. The distinction between parts and processed-based ontologies is the construct given modelling primacy. A parts-based ontology gives the parts within the model primacy and higher-level processes (e.g. heat flow) are deduced concepts. QMOD is a parts-based ontology.

Later work recognised that both confluences and QPT processed QDEs. Qualitative modelling systems were then devised to process equations. Kuipers describes his QSIM system as a special-purpose theorem prover for QDEs [123, 125]. Compilers were written to convert QPT models into the QSIM representation [52]. In other work, Iwasaki & Simon's causal ordering system used meta-knowledge of the order in which an equation solver solved its equations to generate an network of the causal influences of variables on each other [112, 114][35].

Subsequent work studied the asymptotic long-term behaviour of more complex systems of equations [108, 263]. Meta-knowledge of mathematical models was applied to deduce the phase-portrait of the equations. For example, Yip used meta-knowledge to infer a library number of valid phase-portraits[36] and deducing a system's long-term behaviour (i.e. its phase portrait) was a process of matching known system properties to this library [263-265]. Ishida used results taken from process control theory to

[34] See section 2.3.5.3.

[35] This early research into qualitative reasoning is well-documented. For extensive introductions see [43, 111]. Tutorial material can be found in [26]. Summaries of the then state-of-the-art may be found in [77, 261]. For updates, see the proceedings of IJCAI and AAAI. Interesting historical perspectives can be found in [60, 84, 112, 124, 125]. A famous public debate between proponents of different QR approaches can be found in [62, 113, 114].

[36] I.E. the area-preserving properties of a conservative Hamiltonian system with two degrees of freedom. See the

demonstrate that the eventual behaviour of certain mathematical models can be accurately predicted from an analysis of real parts of the eigenvalues of a system's sign matrix[37] [108, 109]. Interestingly, Ishida's work seems to parallels earlier independent work by Puccia & Levins [207]. Puccia & Levins show that their intuitive loop tracing approach is really calculating the eigenvalues of a sign matrix. While the "loop analysis" approach of Puccia & Levins mimics JUSTIN's tracing out of causal pathways, it is inappropriate for our purposes:

- Ishida's pessimism about the complexity limits of his approach [109] also applies to Puccia & Levins' process.
- The maths that underlies loop tracing and Ishida's technique crucially depends on a construct that is illegal in QMOD: inhibitory loops of length one.
- Our reading of loop analysis is that it can only process symmetrical causal connections (e.g. mathematical proportionality). It is not clear how this technique can be applied to arbitrary causal connections (which may be asymmetrical[38]).

Standard QR grants mathematical equations modelling primacy. After some experiments with such an equation-based approach, Feldman & Compton rejected equational approaches. The explicit pathway generation/resolution process of JUSTIN seemed a more direct method of implementing a qualitative hypothesis tester. Other reasons for moving away from standard QR are its poor handling of causality and unnecessary behaviour generation. These are discussed in the next two sections.

While standard QR is equation-based, their are several notable exceptions:

- *Bond graphs*: In the bond graph approach, models are built out of components representing abstract energy sources, sinks, storage, and dissipater devices [252, 253]. We still group bond graphs with the rest of equation-based QR since bond graph models serve as a front-end to equation specification.
- *KARDIO*: KARDIO generated a rule-base for heart disease via a machine learning program that condensed the output from an indeterminate qualitative model of heart disease [18].
- *Clancey's work*: Clancey argues that the feature that distinguishes a conventional program from an expert system is that the latter contains qualitative models. Further, Clancey characterises expert system inference as constructing the system-specific model (SSM) from the general qualitative model in the KB. The SSM is an instantiated subset of the general model in the KB that is relevant to the task at hand. Inference proceeds via introspection on a blackboard that contains the current copy of the SSM [38, 40].

[37] A signed matrix A_s of a set of simultaneous equations A is defined as follows: $(A_s)_{ij} = +,-,0$ if $(A)_{ij} >0, <0, =0$ respectively.

[38]

We do not explore Clancey's work or KARDIO in this section since neither focused primarily on causality[39].

2.3.4. QR = Unnecessary Behaviour Generation

Early attempts to model QMOD/JUSTIN with QSIM proved unfruitful. QSIM produced so many behaviours from the Smythe '89 model[40] that it quickly overwhelmed the resources available. Further, for the purposes of assessing whether a set of effects *FX* were possible, QSIM can generate irrelevant behaviours. For example, consider figure 2.5 with known input causes *C = {b}* and effects we wish to explain *FX = {j}*.

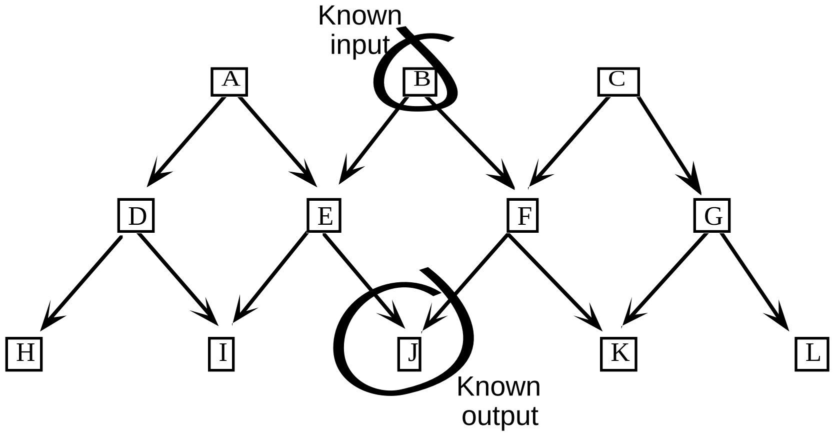


Figure 2.5: *An or-graph of possible inferences. For example,* e if a or b; k if f or g. *Known inputs and outputs are circled.*

Given *b*, we could infer *{b,e,f,i,j,k}* using a QSIM-like algorithm. However, for the purposes of testing if *j* can be explained, the inference to *{i,k}* is unnecessary. For indeterminate models, we should restrict our inferencing as much as we can. We should not compute *total envisionment*: i.e. all behaviours inherent in some fixed collection of objects in some configuration, for each possible state; e.g.*{a,b,c,d,e,f,g,h,i,j,k,l}*. Nor should we compute the QSIM-style *attainable envisionment*: i.e. all behaviours possible from some given initial state; e.g. *{b,e,f,i,j,k}*) [82]. In order to tame our indeterminacy we should only compute the *relevant envisionment*: i.e. the subset of the attainable envisionment that leads to desired output states; e.g. *{b,e,f,j}*.

2.3.5. QR and Problems with Causality

In this section we will discuss QMOD's interest in causality and review the QR's community research into causality. In summary, QR has not resolved the causality issue and QMOD/JUSTIN explored alternative approaches.

[39] For more on Clancey's work, see section 7.3, 8.2, and 8.3.

[40]

2.3.5.1. Causality & QR

A concern at the heart of QMOD is causality. If we listen to two feuding experts using experimental evidence to argue about the internal mechanisms of a model, we will hear statements of the form "this causes that" or "what caused that?". Understanding/ explanation need some causal sequence; e.g. statements of the form "*A* can be explained by *B*".

Causality is difficult to formalise.

> *One might say "my earning poor grades caused the class average to drop" as well as "heat caused the ice to melt". Both seem to be reasonable uses of the word* cause*,yet the sense of causation in each case seems slightly different. [112]*

Separating causality from other inferences is just as difficult. Consider the two rules:

```
rule1 drunk ⇒ obnoxious
rule2 in oncology_clinic ⇒ cancer
```

Such rules could be used to generate explanations for observations such as `obnoxious` and `cancer`. Our explanation generation meta-rule could be:

```
rule3 ((a ⇒ b) & b) ⇒  a explains b
```

Applying `rule3` to `rule1`, we could conclude that `drunk` is an explantion for `obnoxious`. Naturally, this is only a guess based on the available knowledge (i.e. `rule1`). This is not a definite inference since other factors may have caused the observed `obnoxious` behaviour; e.g. loss of job on that day. Nevertheless, it is a somewhat reasonable guess. A less reasonable guess would result from applying `rule3` to `rule2` to conclude that being in the `oncology_clinic` is a cause for `cancer`. This clearly wrong since only patients with `cancer` enter the `oncology_clinic`. `Cancer` is the cause of their presence in oncology, not visa versa (example adapted from [34], chapter 8). The confusion here results from `rule1` being strictly-causal while `rule2` is merely a conjunction of events.

Separating causality from mathematics is also problematic, as two examples demonstrate:

- Consider the statement "when I flick the switch, the lights go on". This expression could be converted into a mathematical equation relating the angle of the switch in its housing to voltages across lightbulbs. However, in doing so, the one-way causality in the original statement would be lost. A mathematical algorithm could incorrectly deduce that when the light goes on, for what ever reason, then this will cause the switch to flick.
- Ohm's law $R=V/I$ relates resistance R, voltage V, and current I. Note that changes in voltages and current do not cause changes in resistance, even though the mathematical formula suggests this is possible. Resistors cannot be manufactured to a certain specification merely by attaching an arbitrary wire to

> some rig and altering the voltage and current over the rig. Ignoring the effects of temperature and high-voltage breakdown, resistance is an invariant built into the physics of a wire. Hidden within Ohm's Law are rules regarding the direction of causality between voltage, current, and resistance. Such rules are invisible to a mathematical formalism.

Despite of its problems, causality is a commonly-used construct. Iwasaki comments:

> *... it is clear that causality plays an essential role in our understanding of the world ... to understand a situation means to have a causal explanation of the situation. [110]*
>
> *Human beings seem to have an innate need to understand and explain everything which happens in the world in terms of cause and effect…Causality is a highly intuitively concept and as such, an ill-defined one. But it is crucial for understanding human ability to reason about the world. [112]*

Causality was a central concern in QR until the mid-80s:

> *A concern at the heart of qualitative reasoning is the notion of causality. Indeed, one of the motivations for early work in qualitative physics was to develop an explicit treatment of causality that was unavailable in traditional physics. [43]*

Systems developed in the late 1970s let the domain expert specify explicit networks of causal connections. Early work demonstrated the utility of this approach; e.g. CASNET [258] and MECHANISMS LAB [221][41]. This latter system had some similarities with QMOD/JUSTIN; some of its causal links were conditional on some pre-condition[42]. Subsequent work argued for adding abstraction hierarchies to causal models; e.g. ABEL [187]. Causal models at various levels of abstraction permit inferencing down/up/across abstraction level(s) if more/less/same abstraction is useful in the reasoning.

These early systems were significant pieces of original and innovative research and successes in their own right. However, they failed to provide general principles for later work and did not address the issue of hypothesis testing. None of these systems regarded their causal models as hypothetical constructs that required review and revision. The additional architecture required for such review was the focus of QMOD/JUSTIN. QMOD/JUSTIN could not use CASNET-style causation strengths on its connections between compartments since like most numbers in our domain, these strengths are unknown. Nor are ABEL-style abstraction hierarchies relevant since in JUSTIN's search for all possible answers, the entire theory is explored across every abstraction level; i.e. a bigger search.

2.3.5.2. Equation-based Causality

In a search for such general principles, many researchers in the QR field moved to equation-based approaches in the 1980s. Causality in equation-based QR is a deduction

[41] For other work, see the list [261], p2.

[42] For example, in Figure 2.4 in section 2.2.2.1, the link into the *cortisol* node is conditional on the absence of

from some non-causal source. For example, CONFLUENCES assigned causality according to the way a disturbance is propagated from one variable to others around a model (controlled by the QDEs in the nodes and arcs of its representation) [62]. Causal ordering and its clones [11, 92, 112, 114, 131] deduce causal networks from equations.

As an example of equation-based causality, consider the bathtub of Figure 2.6.i and its associated equations in Figure 2.6.ii. In figure (i), *qOut* is the quantity of water flowing out of the system; *qIn* is the quantity of water flowing into the system; *I* is the pressure at the bottom of the bathtub; *a* is the amount of water in the bathtub; *k* is the valve opening at the base of the tub; and *c1, c2,* and *c3* are constants. The five equations *E* of Figure (ii) are saying that (E_1) the value opening *k* and (E_5) the input flow rate *qIn* are exogenous (externally controlled); that (E_2) the output rate *qOut* is proportional to the pressure *p*; that (E_3) the pressure *p* is proportional to the amount of water *a*; and (E_4) when the system is in equilibrium, the input flow *qIn* equals the output flow *qOut*.

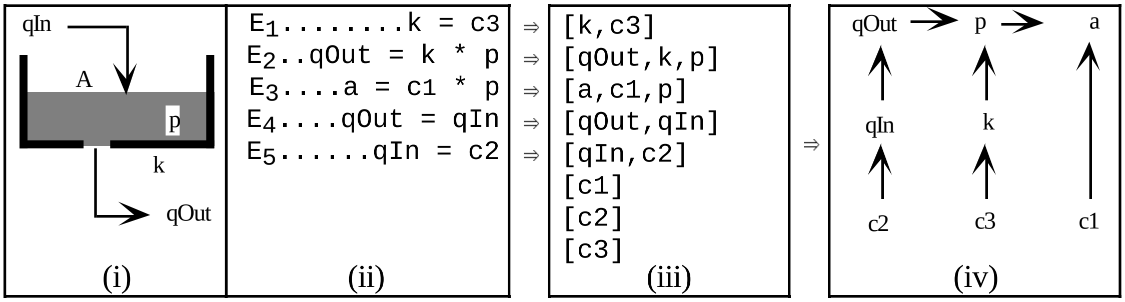


Figure 2.6. *Casual ordering: the bathtub example. From [110],*

Figure 2.6.iii converts each equation into its qualitative form; i.e. E_1 is a relationship between *k* and *c3*. Causal ordering uses meta-knowledge of the inference engine of a equational solver to deduce the order in which an inference engine processes the variables within a model. Causal ordering looks for collections of *N* statements in the qualitative representation of an equation with *N* unknowns. Since *N* equations with *N* unknowns can be solved algebraically, causal ordering declares that these *N* unknowns are "solved" and the process repeats till all the variables are solved. If at any cycle, *N* equations with *N* unknowns can't found, then causal ordering declares "solved" the variables in the smallest number of equations that could be solved simultaneously. If variables at cycle *N* require variables solved for at cycle *N-1*, then the latter are "caused" by the former. For our above example, the first cycle finds E_1 and E_5 and declares known *k* and *qIn*. In the next cycle, causal ordering finds E_4, which declares *qOut* solved. The next two cycles uses E_2 to solve for *p*, then E_3 to solve for *a*. The resulting causal model is shown in Figure 2.6.iv.

2.3.5.3. Issues with Causality

Causal ordering reserves the term "structural equation" to describe equations that articulate the core mechanisms of a system. Deciding which equations are structural is a subjective process. Experts may differ as to what are the relevant or active processes in a domain[43]. If different world views yield different structural equations then there is no guarantee that a unique "true" causality will be deduced from a QR system that computes a casual network from some given model.

Causality within QSIM is an open question. Kuipers' own summary of his early work notes that his program(s) did less "causal" inferencing than constraint propagation and satisfaction ([124], p130). Optimisation techniques for constraint-satisfaction may preclude the generation of causal sequences. Kuipers [125] acknowledges that the Waltz filtering of QSIM is a textbook application of Mackworth's arc-consistency algorithm [135, 136]. This algorithm builds a set of all the possible links between states, then removes (in polynomial time) the links that violate known constraints. The resulting set can be queried for what states assignments for a variable are possible, but not for an explanatory causal sequence of state transitions from known causes to those legal states[44].

After an inconclusive public debate in 1986 between the causal ordering and CONFLUENCES approaches [62, 113, 114], the term "causality" was avoided by many QR researchers. Forbus's 1992 summary of causality in QR is somewhat negative. After observing that his QPT system does not handle causality properly, he adds:

> *"... In terms of violating human intuitions, each system of qualitative physics fails in some way to handle causality properly. Like (QPT) theory, deKleer and Brown's confluence theory... fails to distinguish between equations representing causal versus non-causal laws. Kuipers QSIM contains no account of causality at all..." [83]*

In summary, experiments with understand causality via qualitative equations were inconclusive. A mathematical equation is fundamentally not suited for causal descriptions, be it a qualitative or quantitative equation. Causal modelling requires primitives that can specific causal directions. Mathematics condones many directions (e.g. our Ohm's Law example above). QMOD's causal links do not model that "causality" of standard equation-based QR.

Since latter-day QR has no modelling primitive for single direction causality, techniques like CONFLUENCES, QPT, QSIM and causal ordering an unsuitable for QMOD/JUSTIN. Feldman & Compton therefore returned to the earlier QR style of CASNET/ ABEL/ MECHANISMS LAB, with one important difference. The causal network supplied by humans are open to critique and review. QMOD/JUSTIN added testing tools that assessed the utility of the entered causal links.

[43] See section 1.1 for a discussion why this is so.

[44] Mackworth comments that once the legal transitions are deduced, then a standard search algorithm could traverse the reduced sets of links [136]; i.e. explanations could be generated by a post-processor. Such a post-

2.3.5.4. Causality as Optional Inference

Standard QR research does not resolve the issue of the semantics of the causal links in QMOD graphs. Are the links really "causal" or do they are kludged up version of other kinds of inferences? We finesse the issue with the following argument.

The modelling intent of Figure 2.7 is, in part, that increases in *ACTH* can possibly lead to increases in *CORTICO*.

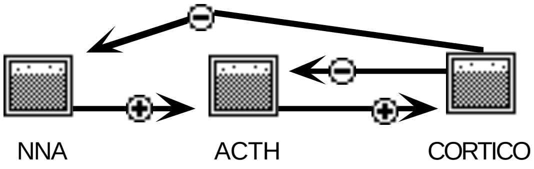


Figure 2.7. *A sample QMOD graph. What do* −⊕→ *and* −⊖→ *mean?*

A variety of inference operators "can possibly lead" to one thing from another; e.g. logical implication. However, the edges of a QMOD graph are not simple logical implication. The first-order-provable statement:

$$ACTH(up) \vdash CORTICO(up).$$

could be derived from Figure 2.7. Such a statement would have to be processed by a meta-interpreter that assessed the utility of using this inference on the basis of a criteria global to the entire model: i.e. "does it lead to the most number of explanations?". Nor should we convert these models to equations of the form (e.g.)

$$CORTICO=k_1*ACTH$$

(where k_1 is some constant of proportionality). Such an equation potentially condones more explanations than our expert intended. For example, using this equation, we could conclude that we could explain decreases in *ACTH* due to decreases in *CORTICO* (using $ACTH = CORTICO/k_1)$. However, the modelling intent is exactly the opposite: *CORTICO* increases can explain *ACTH* decreases via an −⊖→ edge.

Our only pre-condition for asserting *X can possibly lead to Y* is that a user will accept *X* as an explanation for *Y*. This is an uncertain inference. Uncertain inferences generate assumptions. Some assumptions are *controversial*; i.e. clash with other assumptions. Worlds are generated via the separation of the controversial assumptions into subsets which are maximal with respect to the number of pathways they permit and are internally consistent; i.e. th JUSTIN *resolve* process. Each world contains some subset of the inputs, some subsets of the outputs, a set of world-defining controversial assumptions that are internally consistent, and a list of explanatory pathways that are consistent with the assumptions. The believed worlds are selected by a domain-specific *BEST* operator. Sample *BEST* operators are shown in Table 2.2.

$BEST_1$	Select the world(s) that use the least number of assumptions.
$BEST_2$	Select the worlds that use the fewest causes.
$BEST_3$	Select the "simplest" world(s); i.e. those worlds with smallest proof size.
$BEST_4$	Select the world(s) that include the most number of effects.
$BEST_5$	Select the world(s) that include the least number of edges from a source you disagree with; e.g. some rival expert.
$BEST_6$	Select the world(s) that include the smallest percentage of *base controversial assumptions* . A base controversial assumption is an controversial assumption that has no other controversial assumption upstream[45].
$BEST_7$	Select the world(s) that uses edges of highest probability. For example, a MYCIN rule could link a set of premises to a conclusion and this inference was assessed via a certainty factor attached to the rule [24].

Table 2.2. *Sample BEST operators. QMOD/JUSTIN used* $BEST_4$*.*

That is, edges in a QMOD graph are an optional inference that we can make, if that will satisfy a *BEST* operator. Note that:

- There is no one "best" *BEST* operator. QMOD/JUSTIN used $BEST_4$. One advantage with $BEST_4$ is that it is very hard to argue against. A model that can't reproduce known behaviour is definitely faulty[46]. Whatever *BEST* is selected, it amounts to a small piece of code that is a post-processor to an inference engine[47]. Hence, it can be customised. Any of the above, or more, could be used singularly or in combination. Whatever *BEST* is used, feuding experts must agree to it before the process of hypothesis testing begins.
- The effects that are termed inexplicable are those that do not appear in the *BEST* worlds. In the case where *BEST* returns more than one world, we argue that all the effects in the *BEST* worlds should be deemed explained. This is an overly-generous inference since not all the worlds may be compatible. However, it is the simplest resolution of the multiple *BEST* worlds problem and even with this generous approach, we find we can still generate a significant level of model critique[48]

We are sensitive to the criticism that this definition of *can possibly lead to* could blur important distinctions. Various researchers caution against mixing up different inference operators in the same knowledge base:

- For example, Poole argues that the difference between different styles of diagnostic reasoning are defined only for knowledge bases that use fundamentally different inference operators [197]. DeKleer-style consistency

[45] The connection between base controversial assumptions and the minimal environments of the ATMS are explored in chapter 4.2.2.4 and 4.2.2.5.

[46] Counter argument: the data collection was faulty. For the purposes of this research we will ignore this counter argument and assume that our measurements are perfect but our models may be faulty.

[47] For sample code for $BEST_1$, $BEST_2$, $BEST_3$, $BEST_4$, and $BEST_6$, see section 4.3.8.

diagnosis uses inference as logical implication while Poole-style set coverage requires inference as causality[49].

- For other examples, see above text[50].

Perhaps if we were to explicate the differences between different types of *can possibly lead to*, then we could restrict the range of valid explanations permitted by each sub-type of *can possibly lead to* . We do not explore such distinctions since:

- The framework we develop defines *can possibly lead to* as a small table (the *links* relation of the data-compiler[51]) describing which value V_1 of what an object of class C_1 can lead to downstream objects of class C_2 to have the value V_2. More restrictive versions of *can possible lead to* could be implemented as different *links* tables.
- Due to its lack of distinctions between different causal inferences, our *can possible lead to* operator is over-generous with its proposed explanations. Even with this over-generous explanation capacity, we can achieve a non-trivial level of critique[52]. We therefore believe that our generic critiquing process need not await more precise definitions of *can possible lead to* . This is a fortunate observations since…
- The precise semantics of causality have been unsuccessfully explored by numerous researchers[53] and a consensus semantics has yet to emerge.
- We are not overly concerned with possibly incorrect inferences in our system since we embed causal constructs inside editors that allow an expert to specify directed causal networks that are subsequently checked against a library of observations (e.g. QMOD/JUSTIN).

2.3.6. Belief Networks & Machine Learning

We find that other researcher share our view of "causality" as probable inferences that must be assessed. One well-developed assessment criteria is mathematical probability, as used in Pearl's *belief networks* (BNs). BNs deduce causal connections from a statistical analysis of the frequency distributions of variables in a sample to deduce acyclic "networks" (which are really trees) of causal relationships between variables [188, 190]. Current state-of-the-art BNs assumes acyclic models [93] (whereas the models in our test domain are usually cyclic[54]).

[49] See Figure 7.3 in section 7.1.3 for an example of the difference between consistency-based diagnosis and set-covering diagnosis.

[50] See the examples in section 2.3.5.1.

[51] A sample *links* relation is described in section 5.4.1.

[52] See sections 6.2 and 6.3 (in particular, Figure 6.22).

[53] See the previous section on causality research in QR in section 2.3.5.3.

[54]

Generating belief networks requires access to large amounts of data is available on all the entities in the domain. Our domain is characterised as being hypothesis-rich, but data-poor. Obtaining values for certain chemicals within the body is not as simple as, say, attaching a volt meter to an electric circuit. Often delicate measurements have to be made by skilled staff using expensive equipment. The measurement must be repeated a statistically meaningful number of times. Also, in certain domains, it may take years to gather the data (e.g. large-scale epidemiological studies). Hence, the data required to assist feuding experts debating different versions of the same model may be unobtainable or incomplete. Consequently, in the Smythe '89 study, the majority of the model was poorly measured.

Like most inductive generalisation machine learning algorithms[55], BNs make little use of the current background theory; i.e. they make no attempt to preserve current beliefs. In our domain, we find this unacceptable. Various users would treasure their favourite portions of their model(s) (typically, the ones they have developed and successfully defended from all critics). It would be unacceptable to permit a learning algorithm to scribble all over this knowledge. Learning programs for this domain must strive to preserve the current background theory, an approach explored by Mahidadia in this domain in an inductive-logic-programming framework. His research seeks the minimal change to a first-order theory (minus function symbols) such that it can prove all the known positive examples E^+ of model behaviour. Mahidadia's background theory B is a Prolog version of the Smythe models represented as a set of edges E and vertices V. The hypothesised change to the theory H is a set of minimal vertices V_1 and edges E_1 such that the new model $<V_0 + V_1, E_0 + E_1>$ satisfies $B \wedge H \vdash E^+$ [137, 139-141][56].

2.3.7. Ripple-Down-Rules (RDR)

It is informative to contrast QMOD/JUSTIN with Compton's other main research area: building maintainable classification systems using the *ripple-down-rules* technique (RDR) [47]. In RDR, whenever a case results in an inappropriate conclusion, the patch knowledge is entered in as an *unless* test beneath the rule that resulted in error. As the specification develops, it grows into a binary tree with knowledge patches stored at every node (see figure 2.8). At runtime, the final conclusion is the conclusion of the last satisfied node.

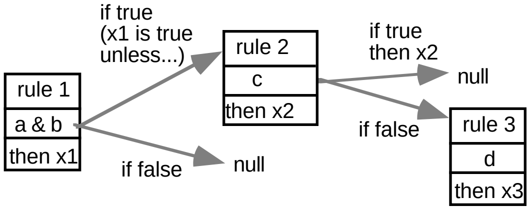


Figure 2.8: *An RDR-tree. At runtime, the output conclusion is the conclusion of the last satisfied node.*

[55] E.G. the ID3 system described in section 3.5.

[56]

The advantage of this approach is that knowledge that works is never changed. New knowledge is always an addition to the specification, never a re-write. Whenever a new rule is added, the input case that prompted the rule addition is cached with the new rule. Such cases are called *cornerstone cases.* The *unless logic* is generated by the expert via a simple selection from a *difference list.* To generate this difference list, the RDR shell computes the set difference between all the possible descriptors for the input case and the cornerstone case of the incorrectly last-satisfied node. This is presented to the expert who can select N (N >= 1) items off this list for inclusion into the new rule. For example:

- If the relevant cornerstone case had referred to a measurement of thyroid stimulating hormone (*TSH*) as *high* and the *T4* hormone as *low*; and
- If the incorrectly classified case referred to *TSH* as *high* and *T4* as *high*;
- Then the difference list would comprise one item: *T4 = low*. References to *TSH* would be dropped since the *TSH* attribute(s) of the incorrectly classified case was the same as the *TSH* attribute(s) of the relevant cornerstone case.

With this difference list, the condition of the patch rule could only be *if T4 = low then ...*.

The astute reader will note that this is only a patch on the rule that was discovered to be anomalous for this case. The same logic anomaly could exist in other rules in different branches and would remain unpatched by the above process. In practice, the process of tracking down these other anomalies occurs as part of correcting other cases. Hence, tracking down anomalies on other branches does not noticeably increase the maintenance effort described above.

2.3.7.1. Optimising for Maintenance

Note that the RDR formalism makes no commitment to tree structures that are optimal. An RDR tree can contain repeated tests, redundant knowledge, and its sub-trees can overlap each other semantically. There exists some experimental evidence suggesting that the redundancy rate may not be significantly large:

- RDR trees tend to be broad and flat (maximum number of patches in a 2037 rule system = 8, average = 2-3).
- Gaines and Compton describe techniques for the machine learning of RDR trees. When given cornerstone cases from existing RDR trees, only a 50% size reduction was observed [89].
- The seemingly inefficient RDR trees have never proven to be too slow in practice.

Further, while redundancies and logically overlapping sub-trees are less than optimal in a computational sense, it is somewhat misguided to attempt to optimise an RDR tree to (e.g.) remove the redundancies or separate out the overlaps. The important feature of an RDR tree is that it is optimised for maintenance. Alternative knowledge representation schemes *may* run faster but incur the penalty of harder maintenance. To see why RDR simplifies maintenance, consider how a conventional expert system would encode the knowledge of Figure 2.8. Most probably, they would enumerate all the logical paths in the RDR tree and write one rule for each path. Assuming the RDR interpreter, then the logical paths for Figure 2.8 are shown in Table 2.3.

Rule	*If*	*Then*
1	a & b & not c & not d	x1
2	a & b & c	x2
3	a & b & not c & d	x3

Table 2.3: *A propositional system that is equivalent to Figure 2.8.*

If the knowledge of the system is patched, then in a conventional rule-based expert system, this patch could extend over many rules. Repeating our above example, the patch on the *x1* anomaly requires an edit to one rule (*rule1*) and the creation of another (*rule2*). Further, the new logic refers to *c* which is a new concept that must be propagated down to all related rules (*rule3*). The more related rules, the more edits. As the knowledge base grows, so to does the number of edits. Hence the time taken to make a change increases and we have a bad maintenance environment.

In RDR, existing knowledge is frozen and we only *extend* the knowledge base. The time taken to change the knowledge does not increase as the knowledge grows since the knowledge base author does not have to tour all the knowledge to make a patch. Instead, at patch time, the system presents the author with a list of candidate delta logic and the author selects item(s) off that list. These items are added beneath the incorrect node. This is the action at *every* knowledge patch time. Maintenance time is hence reduced. For example, see the section on the PIERS systems

2.3.7.2. RDR Example: PIERS

Large expert systems are notoriously hard to build and maintain [243, 255]. Neither of these problems were found to be true with the use of RDR for the PIERS system. PIERS is an expert system for interpreting biochemistry results in routine daily use at St. Vincent's Hospital Sydney [206]. PIERS's expertise covers 20% of the biochemical tests performed at the hospital. PIERS processes 500 cases per day at 95% accuracy, contains 2037 rules, and is one of the largest expert systems in routine use in the world today. PIERS solves a non-trivial problem:

- PIERS models 20% of human biochemistry sufficiently to make diagnoses that are 99% accurate.
- PIERS was originally built for one "domain" (thyroid tests). It now covers a dozen domains.
- Many areas of human biochemistry are open research issues.

- Prior experiments in the same domain used extensive and elaborate abstractions, but never made it out of the prototype stage (e.g. the ABEL system described above [187]).

Development was simple. Minimal preliminary analysis was required. After the database connections were made (using standard software engineering techniques), experts just considered the cases presented on a particular day and told PIERS what to say for each such case. Maintenance time is constant (2-6 new rules per day) and very simple (a total of a few minutes each day).

2.3.7.3. RDR Drawbacks

Despite the advantages of the RDR approach (e.g. easy maintenance), RDR makes two assumptions that are not relevant in all domains.

- The expert does not wish to browse the knowledge base. RDR allows experts only to view the portion of the KB used by a particular case. Any other, more global, analysis of the KB is prevented by the RDR interface.
- The KBs generated via RDR will not be used for purposes other than interpretation by an RDR system. A KB generated by a RDR system cannot (e.g.) by ported to a qualitative reasoning system for simulation purposes.

Therefore, for the purposes of modelling neuroendocrinological theories, Compton encouraged Feldman to work on tools that explicitly represented an experts current perception of a model. Hence QMOD/JUSTIN.

2.3.7.4. RDR vs QMOD/JUSTIN

Despite the many differences between RDR and QMOD/JUSTIN (e.g. the explicitly represented model), the two systems are instantiations of Compton's view on how to develop knowledge based systems.

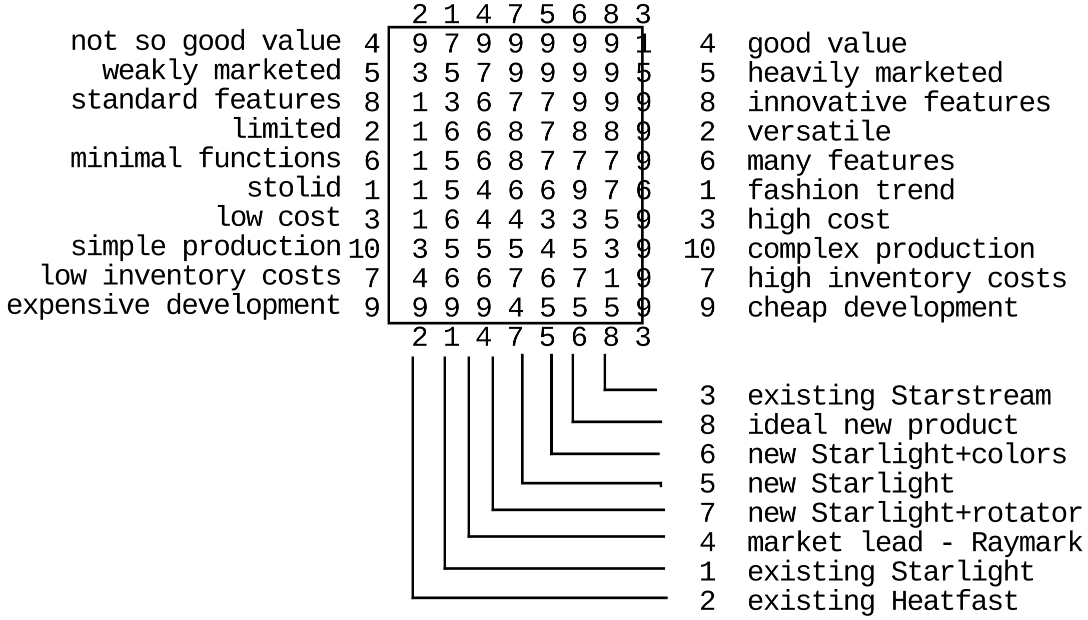
To Compton, all knowledge is a context-dependant construct. Compton-style KBS systems represent their knowledge using the lowest possible representational primitives. To do otherwise, in the Compton view, allows person-specific abstract constructs to pollute general representations. Hence (i) RDR represents its knowledge as simple propositional object-attribute-value triples; and (ii) QMOD/JUSTIN assume no level of abstraction in its models above the level of directed edges which connect vertices that can only be in one of N mutually exclusive states.

Knowledge base development should be driven by direct experience with the domain and not by gedankens experiments. In a Compton-style KBS development approach, test cases play a prominent role. Hence (i) RDR only modifies its knowledge base as a result of an incorrect inference for a particular experiment and (ii) QMOD/JUSTIN defines a good/bad model only in terms of that model's ability to explain known test cases.

Compton-style systems use very simple inference procedures. So simple, in fact, that the pre-experimental intuition is that they will not work. For example:

- RDR makes no use of intermediary conclusions; i.e rules cannot make an assertion that another rule uses later in the inferencing. Such a system may not adequately model some time-based domain (such as the progression of a disease).
- Consider the indeterminate qualitative models process by QMOD/JUSTIN. Such a poorly-constrained system could offer explanations for very many possible behaviours. If so, then its ability to categorically state that a particular behaviour was impossible would be very limited.

Both these pessimistic pre-experimental predictions proved to be unfounded. Using RDR large systems that interpret time-series data (e.g. PIERS is a 2037 rule system) can be built and easily maintained without knowledge engineers. Using QMOD/JUSTIN, data found in published papers can be used to fault the models proposed by those papers (these faults are invisible to other techniques). The constructs used in these systems were significantly simpler than constructs required by other approaches. We wonder if underlying the intricacies of existing expert systems are a minimal set of KR techniques that are the essential components of artificial expert competency. We ask proponents of alternative methodologies requiring more intricate design constructs if they have experimented with simpler alternatives? We note that designs that seem naive at first glance may in fact produce satisfactory competency with comparatively less effort.



```
                        2 1 4 7 5 6 8 3
     not so good value  4  9 7 9 9 9 9 9 1    4  good value
       weakly marketed  5  3 5 7 9 9 9 9 5    5  heavily marketed
     standard features  8  1 3 6 7 7 9 9 9    8  innovative features
               limited  2  1 6 6 8 7 8 8 9    2  versatile
     minimal functions  6  1 5 6 8 7 7 7 9    6  many features
                stolid  1  1 5 4 6 6 9 7 6    1  fashion trend
              low cost  3  1 6 4 4 3 3 5 9    3  high cost
     simple production 10  3 5 5 5 4 5 3 9   10  complex production
   low inventory costs  7  4 6 6 7 6 7 1 9    7  high inventory costs
 expensive development  9  9 9 9 4 5 5 5 9    9  cheap development
                           2 1 4 7 5 6 8 3

                                              3  existing Starstream
                                              8  ideal new product
                                              6  new Starlight+colors
                                              5  new Starlight
                                              7  new Starlight+rotator
                                              4  market lead - Raymark
                                              1  existing Starlight
                                              2  existing Heatfast
```

Figure 2.9: *Repertory grid analysis of terms used by a product development group concerned with relevant existing and competitive products, and with new product proposals under consideration. From [234].*

2.3.8. Repertory Grids

Both QMOD/JUSTIN and RDR emphasis the use of examples when changing a knowledge base. The repertory grids of Gaines & Shaw build non-executable models of conceptual systems in an analogous manner. Experts are asked to identify dimensions along which items from the domain can be distinguished. The two extreme ends of these dimensions are recorded left and right of a grid. New items from the domain are categorised along these dimensions. This may illicit new dimensions of comparisons from the expert which will cause the grid to grow [90]. For example, see Figure 2.9. The grids can be used to compare terms in the domain (e.g. Figure 2.10).

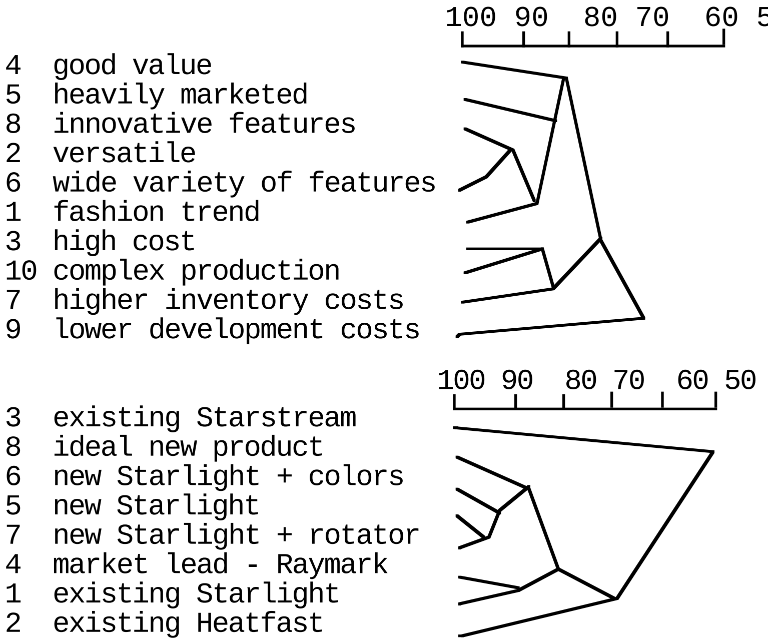


Figure 2.10: *Repertory grid analysis of the differences between terminology and items from the domain of Figure 2.9. For example, an* ideal new product *seems closest to the* Starlight *product and furthest away from the* Starstream *product. From [234].*

Once the dimensions stabilise, and a representative sample of items from the domain have been categorised, then the major distinctions and terminology of a domain has been defined. Differences between the conceptual views of different experts can be identified (e.g. their categorisations are different). Gaines & Shaw describe automatic tools for generating plots representing the proximity of different expert's conceptual systems [91, 234]. Their work focuses on identifying and resolving conflicts in the meaning of individual terms, not on conflicts in the semantics of the models built using those terms as primitives. Such model-level conflict detection requires a execution module (e.g. QMOD/JUSTIN).

2.4. QMOD: Important Features

In this section, we step back from the Feldman & Compton implementation to describe aspects of their approach which were crucial to this subsequent research.

The basic QMOD problem was multiple expert knowledge acquisition in poorly-measure domains. This problem is characterised by a *lack of oracle*:

- There exists no human with final authority to tell us "what is really true".
- Machine learning or statistical techniques have insufficient data to automatically generate any theory with confidence.
- The models/ knowledge bases of QMOD were constructed from many sources, some of which were in opposition. For example, an on-going debate within neuroendocrinology concerns how glucose production is regulated. Certain European researchers argue for peripheral control by the pancreas while certain Australian researchers argue for central control by the brain. Neither group has convincingly refuted the other, despite years of debate.

Feldman & Compton did not take the usual verification approach for knowledge-based systems (e.g. search for inconsistencies, tautologies, redundancies or circularities as done in other systems such as [178, 205]). Their experience with neuroendocrinological theories was that such concerns were orthogonal to the process of model review. Models may contain less-than-optimum internal features, but still be optimum in terms of their comparative ability to explain known behaviour. Feuding experts will not reject their preferred models due to (e.g.) the presence of tautologies within the model. However, reports of the failure of their model to explain certain known behaviour can prompt furious activity.

The union of a set of possibly conflicting ideas has certain interesting properties:

- The total model may be inconsistent; i.e. we can use it to generate false.
- We will often be in the situation where we have to use models that are known to contain inconsistencies. In hypothetical domains such as neuroendocrinology, the process of removing an inconsistency may take years of research. In the meantime, we will use the best knowledge available knowledge to solve our immediate problems. Our current models that may well be inconsistent, but represent our best understanding to-date of the domain.
- The total model may not explain all known behaviours. This is an unexpected experimental result from the Feldman & Compton experiments[57]. That is, only subsets of the model may be consistent and only subsets of the data may be reproducible from these consistent subsets.

[57]

- These models are not neutral to our feuding experts. An expert may go to great lengths to defend a model they have built up over years of research.
- Models may refer to entities that are not fully measured. For example, entities may be fundamentally hard to measure, or the purpose of constructing the model may be to explore a previously unstudied/unmeasured phenomena. Therefore:
    - QMOD models include some qualitative component and, hence, are indeterminate.
    - QMOD-style model review can not dictate to the environment what data it required for model review. Instead, the review process has to make do with whatever observations are available.

The goal of the QMOD inference is the resolution of arguments between feuding experts; i.e. some decision that one model is "better" than another. "Better" is defined in terms of coverage of known behaviour. The output of QMOD should be a categorical statement that, after comparing model M_1 and model M_2, that one of them is *BEST* at reproducing known behaviour. Since QMOD models are indeterminate, all indeterminate possibilities must be explored. Heuristic statements that M_1 is only probably worse than model M_2 may not terminate the argument[58]. We can tell an expert that their model cannot explain known observations only *after exploring every possible behaviour of their model*. Note that this exploration of all possibilities is fundamentally a slow process. One of our subsequent goals, therefore, must be some exploration of the runtimes for this system.

Note that "better" does not mean "perfect". One model may be "better" than another without being able to reproduce all known behaviours. Hence, the goal is not "prove the model is wrong" but to comparatively evaluate competing models. Feldman & Compton were proponents of Popper's view of knowledge: i.e. models are never "true" in some absolute sense. Rather, the models we currently believe are the ones that have survived active attempts to refute them [203]. Significantly, if a Popperian model can explain all currently known behaviour of the entity being modelled, there is no guarantee that it will explain subsequently observed behaviour. Popperian models should be re-tested whenever new data is available for verification.

If we map the above discussion into computational terms, then we can implement Popperian model review (e.g. QMOD) as a search for a consistent subset of a model (i.e. one that cannot generate false) that explains the largest subset of the known behaviour (e.g. using deduction). We describe that search below. For the sake of clarity, this will be a simplified version of QMOD/JUSTIN.

[58] The M_1 author could realise that "You mean that there is a chance that my model is still better than theirs? I

2.4.1. Pseudo-Code for QMOD

The input to QMOD is a set of `models` and known `behaviours` for those `models`. A `model` is a `graph` comprising `edges` and `vertices` reflecting the dependencies between entities in the expert's world view. Each `vertex` contains a list of `vertices` that it `contradicts`. Each model is augmented with a numeric measure of its value; i.e. the percentage of known outputs it can explain. `Behaviour` is a list of inputs `ins` and outputs `outs` which are lists of vertices.

```
  vertex      = record about : any;   -- some entity in the
                                      -- expert's world view
                     contradicts: vertices end;
  edge        = record from,to: vertex end;
  vertices    = set of vertex;
  edges       = set of edge;
  graph       = record v: vertices; e: edges end;
  model       = record g : graph; value: integer end;
  models      = list of model;
  behaviours  = list of record in, out: vertices end;
```

From a set of edges in a user model, we can extract the `vertices`, `roots`, and `leaves` they contain as well as the number of `outputs` they explain (called the `cover`):

```
function contents(es: edges) : vertices
  -- returns all e.from and e.to for e ∈ es;
function roots(es : edges) : vertices
  -- return all contents(es) that have no parents in es
function leaves(es : edges) : vertices
  -- return all contents(es) that have no children in es
function cover(es : edges, outputs : vertices) : integer
  return size(contents(es) ∩ outputs);
```

A set of edges is inconsistent if any two `vertices` in that set exist within each other's `contradicts` set.

```
function inconsistent(es: edges) : boolean
  return contents(es) ∩forbidden(es) <> Ø;

function forbidden(es: edges) : boolean
var out : vertices; begin
  for e ∈ es do out ← out ∪ e.from.contradicts ∪ e.to.contradicts;
  return out;
end;
```

A set of edges constitutes a valid explanation if it all its `roots` are inputs and all its `leaves` are outputs and it is not `inconsistent`.

```
function valid(es : edges; in,out: vertices) : boolean
var roots, leaves: vertices;
  roots  ←  roots(es); leaves ← leaves(es);
  return   size(roots) > 0  and roots ⊆ in      and
           size(leaves) > 0 and leaves ⊆ out    and
           not inconsistent(es);
end;
```

The main loop of SIMPLIFIED_QMOD allows an expert to create and/or edit models and/or libraries of behaviours, then assess those models with respect to those behaviours.

```
procedure SIMPLIFIED_QMOD
  repeat
    M ← edit_models; B ← edit_behaviour;
    print(best_models(M,B));
  until stopped
end;
```

Edit_models and edit_behaviour are routines that allow experts to sketch their models and enter their data[59]. The core semantics is inside best_models, which calls SIMPLIFIED_JUSTIN as a sub-routine.

```
function best_models(M : models; B : behaviours) : models
var  m: model begin
  for m ∈ M do SIMPLIFIED_JUSTIN (m,B);
  return models_with_most_value(M);
end;

function models_with_most_value : models
  -- return the models with largest m.value
```

SIMPLIFIED_JUSTIN set a model's value to its average percentage maximum coverage of the valid explanations which can be generated from the model for each behaviour. Note that the easy explanations are the member of out that are also within in. The hard outputs to explain are the members of the outs set that are not easy. We only need to search for edges between ins and the hard set.

```
1.   procedure SIMPLIFIED_JUSTIN(var m : model; B : behaviours)
2.   var  done, max, b       : integer;
3.        coverage           : list of integer;
4.        in,out, easy, hard : vertices;
5.        xplains            : edges;
6.   begin
7.     for b in 1 to  size(B) do
8.     begin  in  ← B[b].in     ; out ← B[b].out;
9.            easy ← in ∩ out   ; hard ← out - easy;
10.           done ← size(easy) ; max ← done;
11.           for  xplains ⊆ m.g.e do  -- for all edges subsets do
12.              if     valid(xplains,in,hard)
13.              then   max ← maximum(max, done + cover(xplains,hard));
14.           coverage ← coverage + (max*100/size(out));
15.    end;
16.    m.value ← average(coverage);
17.  end;
```

The core computation of QMOD/JUSTIN is the generation and evaluation of explanations (lines 11,12 and 13). SIMPLIFIED_JUSTIN is clearly impractical since it is exponential on the number of edges in a model. Feldman & Compton's implementation first ran a pre-processor to generate all paths from outputs back to inputs

[59]

using a naive depth-first backtracking search (with no memoing). These paths were then *resolved* into `valid` subsets. This implementation permitted the proof-of-concept work described in [79, 80]. However, their implementation was still very slow. One goal of this research is to explore alternative inference procedures.

The energy produced by the breaking down of the atom is a very poor thing . Anyone who expects a source of power from the transformation of these atoms is talking moonshine: Ernest Rutherford.

For this, indeed, is the main source of our ignorance- the fact that our knowledge can be only finite, while our ignorance must necessarily be infinite: Karl Popper. We *are all fallible, and prone to error; let us then pardon each other's folly. This is the first principle of natural right:* Voltaire.

3. Essential

At its core, qualitative hypothesis testing (QMOD/JUSTIN) was a tool for checking qualitative models without requiring large amounts of data. In this chapter we will argue that such a tool is an essential component for knowledge acquisition systems.

3.1. Most Domains Are Poorly Measured

This section argues that many non-toy domains are insufficiently measured to use quantitative hypothesis testing; i.e. a generalised test module must include a qualitative component.

Many domains are poorly-measured:

- **Neuroendocrinology:** See above discussion regarding the QMOD/JUSTIN work[60] .
- **Economics:** Experiments with data collection for economic modelling indicate that economics is a poorly-measured domain. The (in)famous "Limits to Growth" study attempted to predict the international effects of continued economic growth [149]. Less than 0.1% of the data required for the models was available [44].
- **Ecology:** Puccia & Levins comment on the utility of exhaustive data collection on ecological modelling:

  > *In a complex system of only a modest number of variables and interconnections, any attempt to describe the system completely and measure the magnitude of all the links would be the work of many people over a lifetime* (*[207]*, p5).

  They claim that this observation from ecological modelling also applies to sociological models. For example, it is well known that many crimes go unreported. A literature review on crime statistics shows that the resources required to gather empirical data on the level on unreported crime is prohibitively high [151].

[60]

With only one partial exception[61], all the domains explored by the author in his knowledge engineering career (1986-1994) can be characterised by insufficient available measurements for the construction of a quantitative model. The reasons for this lack of data, and the methods used to cope with this problem, varied from domain to domain:

- **Process control:** ICI Chemicals Australia required an automatic controller for one of its petrochemical plants. A mathematical controller could have been constructed, but would have necessitated the purchase of a set of parameter values from the engineering firm that built the plant. Further, generating intuitive explanations from these system suitable for the average plant operator would have been very difficult[62]. These two issues of purchase price and explanation motivated ICI to build a heuristic rule-based controller [164].
- **Farm management:** The PIGE system was an intelligent back-end to AUSPIG, a mathematical model of pigs growing in a piggery developed by the Australian Commonwealth Scientific Industrial Research Organisation (CSIRO). Using PIGE, the intelligence of CSIRO experts can be applied to increase the profit of a piggery by recommending changes to pig diet, housing, or (in the extreme case) genotype [157]. Raising pigs is big business. As of 1988, Australia raised 4.8 million pigs a year (net worth $AUS 500 million). This represented one-third of one percent of the international pig herd (1,440,000,000 pigs). Despite the enormity of the international porcine enterprise, much of the pig remains unmeasured. Building and verifying the AUSPIG model required the collection of new data, especially for that model. This data collection/ model development process took decades of work by CSIRO's top experts in pig nutrition. The package was then sold on a one-off basis to an American manufacturer of feed stocks. Note that (i) the important data required for the model construction/ verification was not originally in the public domain and had to be especially collected over several years; and (ii) once the data was collected, it was promptly bought and hidden by a private corporation for their exclusive use.
- **Consumer lending and superannuation:** For 3 years, the author worked for a large Australian insurance company. Experiments with machine learning in a consumer lending and superannuation[63] domain demonstrated that consumer loan approval and superannuation plan generation are poorly-measured domains. The organisation we worked for had large databases of (i) prior loan approvals; (ii) past and current superannuation plans; (iii) defaulted loans. Much of this data

[61] See section 3.1.1.

[62] Mathematical models are very poor at generating intuitive explanations This was the original motivation for much of the qualitative reasoning work described in section 2.3.5.2.

[63] Superannuation is a special insurance policy that matures near the holder's retirement date that supplements

pre-dated certain crucial changes in Federal legislation making it irrelevant for current use. Construction of expert systems for consumer lending and superannuation was therefore a process of creating new models for new domains that were yet to be measured.

Note that there is no theoretical barrier to the accurate measurement of any value in any of the poorly-measured domains listed above. Given unlimited resources:

- ICI Australia could have obtained the numeric parameters required for its mathematical controller;
- All the parameters required for the construction of precise global economic models could be measured.
- Sociologists can interview sufficient people to determine the levels of unreported crime;
- Neuroendocrinologists can measure all the chemicals in all positions around the human body to a pico-mole accuracy.
- CSIRO can completely measure all the parameters relating to internal pig physiology
- We could collect all relevant data for consumers requiring loans or superannuation prior to expert system construction.

However, organisations have limited staff , time, and money. Model construction is hence a resource-bounded activity. The problems with data collection catalogued above may reflect a fundamental problem with numbers; i.e. *there exist useful numbers that we may wish to measure but lack the resources to collect.* In the absence of sufficient data for model development and testing, we must turn to qualitative (i.e. non-numeric) methods.

3.1.1. An Exceptional Domain

The one partial-exception we are aware of to our general rule of "all non toy-domains are characterised by a lack of data" is the diagnosis of electrical circuits. Electrical circuits are an artificially constructed domain and it is theoretically possible to add instrumentation to circuitry to make all values accessible.

This is a significant exception to our general rule since that area has developed an inference architecture similar to ours[64], yet does not concern itself with model validation. In the diagnosis of electrical circuits, the model can be often described with certainty since it is a product of the VLSI design feed into the CAD system that generated the chip. However, even in that domain, we can find some concern with model construction [2, 48].

[64]

3.2. Lots of Numbers Does Not Equal Lots of Meaning

This section argues that a fixation with precise measurement is somewhat misguided. Modellers may loose sight of a domain when working with numbers that confuse (rather than clarify) the important concepts. That is, qualitative modelling is an essential part of knowledge acquisition.

Precision, and in particular, numeric precision is seen by many as the best way to understand a domain. For example:

> *Through and through the world is infested with quantity. To talk sense is to talk quantities, It is no use saying the nation is large- how large? It is no use saying that radium is scarce- how scarce? You can not evade quantity. You may fly to poetry and music and quantity and number will face you in your rhythms and your octaves.* - Alfred North Whitehead

However, an over-enthusiasm for quantitative analysis can confuse, rather than clarify, a domain. Puccia & Levins comment:

> *For systems whose parts are not listed in a catalogue, which evolve together, which are difficult to measure, and which show unexpected capacity to form new connections, the results of (quantitative) simulation techniques have been less than impressive. The masses of data required make the procedure very costly; the demand for qualitative precision often forces the exclusion of many variables (e.g. stress in diabetes: poorly quantified but vitally important); the predictions apply only to the original single system from which the models were derived, and are not easily extended even to similar systems* (*[207]*, p4).

It is a mistake to confuse numbers with meaning. Numbers are often intermediary concepts which may be required to achieve some interpretation which is best expressed in some qualitative, summary form. This is the lesson of decision support systems. Numbers can overload a decision maker with too many irrelevancies. Data must be condensed to be useful for supporting decisions [3]. For example, suppose we tell Whitehead that the country's size is "ten million hectares", is that answer meaningful? Perhaps a more meaningful answer would be the qualitative statement that "this country is the same size as country X".

Many formalisms lead the expert through a process of supplying all the relevant details of their domain. As such, they are a worthy thing and should be encouraged. However an obsession with precisely defining models can confuse rather than clarify modelling issues. For example, consider applying quantitative compartmental modelling to neuroendocrinology. Quantitative methods force the expert to spend time on precisely quantifying certain details while the real task of debugging their basic models is delayed. Further, when the resulting quantitative compartmental model "works" (i.e. can predict the behaviour of the actual system), it is unclear if the correct behaviour is the result of the correct theoretical model, or a lucky guess of the parameters of an quantitative model. For example, in one text on mathematical modelling in neuroendocrinology, a chapter is devoted to generating various quantitative models of ovulatory cycles. After producing eight different models, the authors conclude:

> *The most striking feature evident from studying these models is the variety of equations which give reasonable representations of the observed experimental data... In each model these apparently appropriate equations have been derived from quite different assumptions and simplifications and use different parameters.* (*[148]*, p 224)

They go further and comment that deeper than the non-unique empirical models are the…

> *...theoretical models (sometimes called analytical or mechanistic) which embody our concept of what causes the behaviour observed.* (*[148]*, p4)

That is, the precise numeric details of a model may be less important than the underlying qualitative concepts of some deeper model.

3.3. External Testing is More Important Than Internal Testing

In this section we will argue that models can be assessed via either internal syntactic criteria or external semantic criteria. We will argue that external semantics is more important. A widely-applicable external semantic criteria is *test-suite assessment* (a re-expression of the JUSTIN process).

3.3.1. Internal Testing

Any program has a internal structure and an intended use. *Internal tests* assess the internal structures. Examples of internal tests in an object-oriented software engineering domain are modularity, encapsulation and methods per object [210]. Examples of internal tests in a KE domain are KBS verification systems surveyed by Preece, Shinghal and Batarekh (PSB) [204] (e.g. [178, 205, 249]).

PSB define a taxonomy of rule-based KBS "anomalies" (see Figure 3.1) and argue that a variety of KBS verification tools target different subsets of these anomalies (perhaps using different terminology).

```
Anomaly
 ----Redundancy
 ---- ----Unusable rules
 ---- ----Redundant rules
 ---- ---- --- Duplicate rules
 ---- ---- --- Subsumed rules
 ----Ambivalence
 ---- ----Conflicting sets of rules
 ----Circularity (inference loop)
 ----Deficiency
 ---- ----Missing rules
 ---- ----Missing values
```

Figure 3.1: *The PSB taxonomy of KBS anomalies. From* [204]. *These terms are defined below.*

Some of these anomalies require meta-knowledge about knowledge base literals:

- A literal is PSB *askable* if it represents a datum that the knowledge base can request from the outside world.
- A literal is a PSB *final hypothesis* if it is declared to be so by the KB author and only appears in a rule conclusion.

A rule is PSB *redundant* if the same final hypotheses are reachable if that rule was removed. An *unusable redundant rule* has some impossible premise. A knowledge base is PSB *deficient* if a consistent subset of askables leads to zero final hypotheses. A PSB *duplicate redundant rule* has a premise that is a subset of another rule premise. PSB define *duplicate rules* for the propositional case and *subsumed redundant rules* for first-order case (where instantiations have to be made to rule premise variables prior to testing for subsets). PSB define *ambivalence* as the case where, given a consistent subset of *askables,* a rule-base can infer final hypotheses.

PSB stress that the entries in their taxonomy of KBS anomalies may not be true errors. For example, the dependency network from a rule-base may show a *circularity* anomaly between literals. However, this may not be a true error. Such circularities occur in (e.g.) user input routines that only terminate after the user has supplied valid input[65]. For this reason, the "errors" detected by internal testing are called anomalies, not faults. Internal test anomalies are used as pointers into the system which direct the developer to areas that require a second glance.

3.3.2. External Testing

External tests are better detectors of faulty semantics. External tests relate a piece of isolated software to the environment in which it will execute; i.e. they assess the ability of a program to fulfil its function. Examples of external tests in an object-oriented software engineering domain are user-acceptability and robustness [210]. One example of external tests in a KE domain is *test suite assessment.* The inputs and outputs of the rule base are identified and a library is built of input/output pairs representing the expected output given the input. The inputs are then run against the rule base and the output compared with the expectation.

External testing is harder than internal testing since it implies making a decision about the correct behaviour of a system in a wide range of circumstances. This is an expert task and a lengthy analysis process. Further, in domains that are poorly understood (e.g. economics. ecology, neuroendocrinology), the goal of model construction may be to predict what the domain would do in certain circumstances. In such domains, information for test suite construction is limited and external testing is fundamentally difficult. External testing is therefore harder to apply and is used less frequently than

[65]

internal testing. External tests are rarely reported in the literature (exceptions: MYCIN [269], CASNET [258], PIERS [206][66], PIGE [157], QMOD [79, 80, 153]).

One interesting variant on external testing are the automatic test generation procedures offered by the dependency-network approaches of Ginsberg [95, 96] and Zlatereva [271, 272]. The dependencies between rules/conclusions are computed and divided into mutually consistent subsets. The root dependencies of these subsets represent the space of all reasonable tests. If these root dependencies are not represented as inputs within a test suite, then the test suite is incomplete. Test cases can then be automatically proposed to fill any gaps[67]. The advantage of this technique is that it can be guaranteed that test cases can be generated to exercise all branches of a knowledge base. The disadvantage of this technique is that, for each proposed new input, an expert must still decide what constitutes a valid output. This decision requires knowledge external to the model, least we introduce a circularity in the test procedure (i.e. we test the structure of M using test cases derived from the structure of M). Further, auto-test-generation focuses on incorrect features in the current model. We prefer to use test cases from a totally external source since such test cases can highlight what is absent from the current model. For these reasons, we caution against automatic test case generation.

3.3.3. Which is More Important?

This section argues that external testing is more important than internal testing.

Relying on internal tests is an uncertain process. Rajaraman & Lyu comment:

> *Experts in software engineering agree that the presence of these (internal) attributes will ensure the existence of the external attributes expected by software users…This is treated almost as an axiom. Despite the important intuitive relationships that exist between the internal structures of software products and their external attributes, there has been little scientific work to establish the precise relationships between the internal and external attributes ([210], p305).*

That is, there is no guarantee that a model that has passed internal testing will satisfy external testing. The syntactical assessment made by internal testing may be completely orthogonal to the semantic assessment made by external testing.

Consider the case where N alternative competing models are available (e.g. QMOD). Such an N model situation could occur in the case KA from multiple experts (when expert opinion is divergent, e.g. experts are feuding), group decision support systems, or the construction of expert systems by a single expert in a new domain with an unknown structure. In this N model case, what is required is some judgement about the relative merit of model X over model Y that cannot be disputed by different competitors.

[66] See section 2.3.7.2.

[67]

A judgement based on internal criteria could be disputed. Programs in routine use can fail internal tests, yet still be deemed useful. Preece & Shingal detected multiple internal test failures in fielded expert systems [205]; see Table 3.1. Yet these systems were passing their day-to-day external operational test (i.e. their behaviour was acceptable). Preece & Shingal offer several reasons why this might be so:

- A rule base's inference engine can tame a subsumption anomaly (e.g. by always picking the rule with the largest satisfied condition).
- Missing rules may reflect the "do nothing" default case [205] or may be out-of-scope for the domain (e.g. our washing rules above).
- As mentioned above, circularities may exist as part of some looping process that terminates in a special condition (e.g. prompt the user for input until they provide us with a satisfactory answer).

	Application					
	MMU	*TAPES*	*NEURON*	*DISPLAN*	*DMS1*	*Hit Rate*
Size (literals)	105	150	190	350	540	
Subsumption[68]	0	5/5	0	4/9	5/59	14/73 = 19%
Missing rules	0	16/16	0	17/59	0	33/75 = 44%
Circularities	0	0	0	20/24	0	20/24 = 83%

Table 3.1. *Some internal errors detected in fielded expert systems. Fractions represent* anomalies/faults. Anomalies *were detected automatically.* Faults *are anomalies that were assessed by the experts to be true errors. The* hit rate *is the fault detection rate. Note that the* hit rate *is much less than 100%.*

Table 3.2 enumerates all the combinations of failed/passed internal/external tests and considers for which combinations a model should definitely be rejected. Note that if our goal is judging categorically when to reject a model, then external testing is the sole determinant.

Passed internal tests?	*Passes external tests ?*	*Definitely reject model ?*
no	no	yes
no	yes	maybe
yes	no	yes
yes	yes	no

Table 3.2. *Failed external tests are always means reject a model. Failed internal tests are not always reasons to reject a model.*

3.4. KA Must Use Testing

In this section, we note that modern KA practice is an iterative refinement of approximate models. Since approximate models may be wrong, they must be tested. That is, testing is an essential part of knowledge acquisition. This section offers only a brief overview of modern and pre-modern KA[69].

[68] Preece uses the term "redundant rule pairs" for subsumption.

[69]

Gaines reports a recent change of perspective in the KA community from a "expertise-transfer" perspective to a "knowledge-modelling" perspective. Since a 1989 symposium on cognitive aspects of knowledge acquisition[70]...

> *"... the knowledge-modelling perspective has become widely adopted and terminologies reflecting an expertise-transfer perspective have been quietly dropped." [88]*

The "expertise-transfer" perspective was the dominant KA paradigm up until the late 1980s. It is best typified by Feigenbaum's characterisation of expert system construction as "mining the jewels in the expert's head" [78]. We would characterise expertise-transfer as a Platonic/Baconist view of knowledge.

Expertise transfer was developed during the pioneering days of expert systems. Subsequently, expert systems practitioners found that they could not adequately generalise or formalise expertise transfer techniques. An alternative to expertise-transfer is knowledge-modelling. "Modelling" is the generation of an artefact that is simpler to manipulate than the entity being modelled. Most modern KA theorists advocate a knowledge-modelling approach (e.g. KADS [260], generic tasks [32], components of expertise [246], model construction operators [40], SPARK/ BURN/ FIREFIGHTER (SBF) [145][71]). Knowledge modelling characterises KE as the construction of multiple partial models. In the KADS case, for example, the models are: organisational, application, task, expertise, cooperation, conceptual, and design [260].

Modern KR practitioners acknowledge the dubious nature of the models in their knowledge bases. Davis *et. al.* define KRs are *inaccurate surrogates of reality* [55]. Silverman cautions that systematic biases in expert preferences may result in incorrect or incomplete knowledge bases [235, 237]. Note that any model of some thing is different to that thing (the map is not the territory). Hence, the model may contain less information than the modelled thing and may behave differently in certain crucial situations. Thus the modelling process introduces errors into the representation. Experienced modellers caution us that:

> *A model is an intellectual construct we study instead of studying the world. Every model distorts the system under study in order to simplify it. ([207], p2)*

Summary: We argue that the rejection of expertise-transfer and the adoption of knowledge-modelling perspective requires the addition of a testing module to a KBS. We reason as follows:

- Knowledge-modelling relaxes the Platonic assumption and is a process of the construction of partial/approximate models.
- Partial/approximate models may be wrong.

[70] At the 1989 Banff Conference on Knowledge Acquisition for Knowledge-Based Systems.

[71]

- Potentially wrong models must be tested prior to their use least they produce inappropriate output in certain circumstances.
- Therefore, testing is an essential process for the products of KA as knowledge-modelling.
- We have argued above for external test suite assessment as the preferred form of testing.

3.5. KM Must Use Testing

This section argues that, ultimately, we can never be sure that a model is ever correct, even after extensive maintenance. Models must be tested as often as possible. That is testing is an essential part of knowledge maintenance (KM).

An optimist may disregard our gloomy warnings regarding the correctness of a model. "Sure", they might argue, "models are never perfect. However, we can make them pretty good and that'll do". However, when we explore experimental studies of model maintenance (be they knowledge-based models or otherwise), we find that the usual case is that they models (i) are clearly not "pretty good"; (ii) will not "do"; and hence (iii) require constant maintenance.

Our first study is a small thought experiment. The reader is invited to maintain a one-line mathematical model of exponential population growth:

$$EQ_1 : dN/dT = rN.$$

In EQ_1, r is a constant reflecting environmental conditions, T is time, and N is the population. Note that this model is wrong[72] since population growth must taper off as it approaches C the maximum carrying capacity of the environment; i.e.

$$EQ_2 : dN/dT = rN(1-(N/C)).$$

If the reader can correctly answer the following question, then we have anecdotal evidence for believing that humans can read and critique models: *is* EQ_2 *correct?* If the reader cannot find all errors in a one-line model (which they probably studied extensively in high school), then we should be suspicious of claims that the truth status of larger models can be accurately determined by people.

EQ_2 is incorrect. In the case of a hostile environment and over-population, $N > C$, $r < 0$, and our intuition is that the population will fall. However, $rN(1-(N/C)) > 0$; i.e. the maths says that population will *increase*. (example from [207]). Our experience has been that the error is not apparent to many people. Note that we have failed to ascertain the truth status of a one-line mathematical model.

[72]

Myers [172] reports controlled experiments with a 63 line model. 59 experienced data processing professionals hunted for errors in a very simple text formatter (63 line of PL/1 code). Even with unlimited time and the use of three different methods[73], the experts could only find (on average) 5 of the 15 errors in this 63 line model [172]. This result, and the thought experiment, does not inspire confidence that experts can accurately assess larger models.

As to other studies, recall the results of the Feldman & Compton study[74]: 109 of 343 (32%) of the known data points from six studied papers could not be explained using a glucose regulation modelled developed from international refereed publications [79, 80, 240]. A subsequent study corrected some modelling errors of Feldman & Compton to increase the inexplicable percentage from 32% to 45%[75]. A similar study successfully faulted another smaller published scientific theory [153][76].

Shaw reports an experiment where a group of geological experts built models for the same domain, then reviewed each other's KBs as well as their own twelve weeks later [233]. Note the two context changes: (i) different experts; (ii) a period of elapsed time between assessing a knowledge base. For the twelve week review study, it was found that an expert's understandability and agreement with their own knowledge was less than total (see table 3.3).

Expert	*Understands* *(max = 100)*	*Agrees* *(max = 100)*
E1	62.5	81.2
E2	77.8	94.4
E3	85.7	78.6

Table 3.3: *Does an expert understand and agree with a knowledge base they wrote 12 weeks previously? From [233]. Note that an expert's understanding may be quite low (e.g. expert* E1 *only understands three-fifths of her own thinking three months ago).*

For the cross-expert review, it was found that experts disagree significantly with each other [233] (see table 3.4).

[73] (i) Reading the 30 line specification, then generating test cases which were run through an executable version of the program; (ii) As before, but also reading the 63 line code listing; (iii) As with (ii), but testing was done via manual walk-throughs/inspections. Programmers only used one of (i), (ii) or (iii). Programmers using (i) and (ii) worked alone. Programmers using method (iii) worked in groups of three.

[74] See section 2.2.3.

[75] See section 6.2.

[76]

Expert pairs	*Understands (max = 100)*	*Agrees (max = 100)*
E1,E2	62.5	33.3
E2,E1	61.1	26.7
E1,E3	31.2	8.3
E3,E1	42.9	33.3
E2,E3	44.4	20.0
E3,E2	71.4	33.3

Table 3.4: *Do experts agree with each other? Three experts reviewed each other's knowledge base. Levels of understanding may be low (e.g. expert* E1 *only understands expert* E2's *knowledge base 31.2% of the time). Levels of agreement may be even lower (e.g. expert* E1 *only agrees with expert* E2's *knowledge base 8.3% of the time).*

We conclude from the Shaw study that experts may disagree with each other, and even with themselves. Consequently, the assessments of the validity of a model by an expert is not definitive. This result concurs with our small thought experiment and Myers' result.

Compton reports other studies that documented the changes made to models of biochemistry diagnosis systems. He documents one expert system (Garvin ES-1) which was developed using an iterative prototyping methodology. Rules that began as simple modular chunks of knowledge (e.g. Figure 3.2.i) evolved into very complicated and confusing knowledge (e.g. Figure 3.2.ii).

```
RULE(22310.01)
IF
   (bhthy or
    utsh_bhft4 or
    vhthy)
   and not on_t4
   and not surgery
   and (antithyroid or
        hyperthyroid)
THEN
   DIAGNOSIS("...thyrotoxicosis")
```

(i)

```
RULE(22310.01)
IF     ((((T3 is missing)
       or (T3 is low and T3_BORD is low))
       and TSH is missing
       and vhthy
       and not (query_t4 or on_t4 or
         surgery or tumour or antithyroid
         or hypothyroid or hyperthyroid))
       or ((((utsh_bhft4 or
           (Hythe and T4 is missing and
            TSH is missing))
           and (antithyroid or hyperthyroid))
       or  utsh_bhft4
       or  ((Hythe or borthy)
           and T3 is missing
           and (TSH is undetect or
                TSH is low)))
           and not on_t4 and not (tumour or
                        surgery)))
           and (TT4 isnt low or T4U isnt low)
THEN  DIAGNOSIS("...thyrotoxicosis")
```

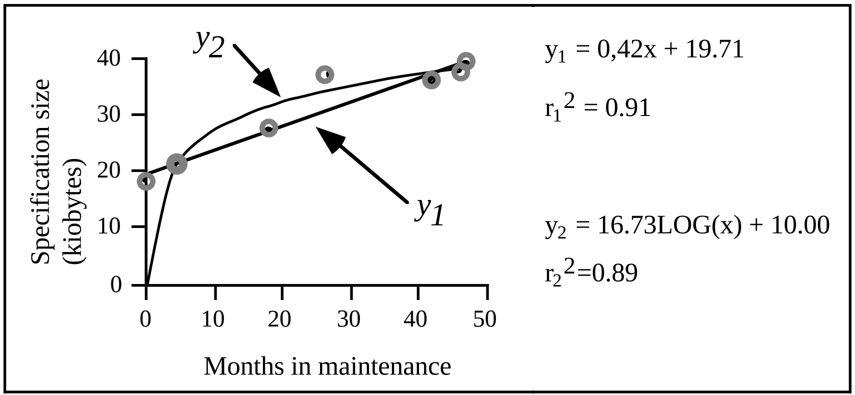
(ii)

Figure 3.2. *A Garvin ES-1 rule proposed in (i) 1984 and maintained till (ii) 1987. From [45]. Note that initially simple expressions of knowledge can grow into very complex structures.*

This particular expert system never reached a logical termination point, despite years of maintenance (see Figure 3.3). There was always one more major insight into the domain, one more major conceptual error, and one more significant addition [47].

Months	*Size*
0	18
4	21
18	27
27	37
42	36
46	38
47	39

(i)



(ii)

Figure 3.3: *Non-termination of the maintenance cycle of an expert system (from [47]). Figures for KB size (measured in kilobytes) is shown in figure (i) and graphed in figure (ii) with plausible curve fits* y_1 *and* y_2*).*

The data of Figure 3.3.i is consistent with either a linear or logarithmic growth:

- Logarithmic growth would be consistent with the Platonic belief that an objective reality exists which we can asymptote towards. However note that the asymptote is very slow (see curve y_2 in Figure 3.3.ii). We can only expect to find Platonic reality after years of effort.
- A linear growth would be consistent with the doubting Thomas position[77]; i.e. recorded human "knowledge" will always be incorrect and will always need correction.

Compton is monitoring the maintenance of PIERS [206], a much larger system (which is version 2 of the above diagnosis system[78]). A $y = x^{0.5}$ growth in KB size has been noted in that system. Significantly, the user-group sponsoring the project have created a permanent line item in their budget for maintenance. They anticipate that routinely every day, an expert will review the generated diagnoses and change some of the KB. That is, they believe that the model will never be finished/correct[79].

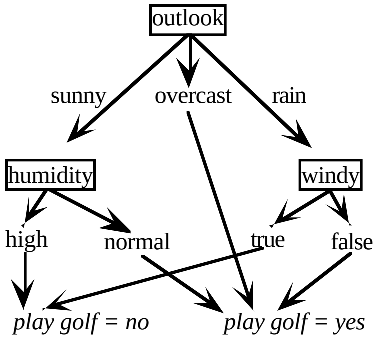
We find that experiments in machine learning endorse our belief that *any* version of a model can be improved. Machine learning programs input training data to generate a model. For example, Quinlan's ID3 program inputs the training data in Figure 3.4.i to generate the decision tree of Figure 3.4.ii.

[77] See section 1.1.3.

[78] For more on PIERS, see section 2.3.7.2.

```
% data(outlook,  temperature,humidity,windy,playGolf?)
data(sunny,      hot,        high,    false,      no)
data(sunny,      hot,        high,    true,       no)
data(overcast,   hot,        high,    false,      yes)
data(rain,       mild,       high,    false,      yes)
data(rain,       cool,       normal,  false,      yes)
data(rain,       cool,       normal,  true,       no)
data(overcast,   cool,       normal,  true,       yes)
data(sunny,      mild,       high,    false,      no)
data(sunny,      cool,       normal,  false,      yes)
data(rain,       mild,       normal,  false,      yes)
data(sunny,      mild,       normal,  true,       yes)
data(overcast,   mild,       high,    true,       yes)
data(overcast,   hot,        normal,  false,      yes)
data(rain,       mild,       high,    true,       no)
```

(i)



(ii)

Figure 3.4. *When shall we play golf? After watching expert golfers, we collect the training cases shown in Figure (i).* ID3 *uses these examples to generate the decision tree of Figure (ii) (e.g. if* outlook=sunny *and* humidity=high*, then don't* play golf*).* ID3 *generates models based on an heuristic taken from information theory (for details see [208, 209]). Note the absence of the* temperature *attribute from Figure (ii). ID3 is smart enough to see that adding* temperature *to the learnt model does not usefully increase its information content. Example from [208].*

Catlett's research explored the following area. Given a large amount of training data (e.g. thousands of training cases such as Figure 3.4.i), is it necessary to use it all? That is, after a certain number of examples, is further experience superfluous? To test this, Catlett used an ID3-variant [209] to generate 20 decision trees for eleven machine learning problems using either (i) all the training cases or (ii) half the cases (randomly selected). The results are shown in Table 3.5. In all cases, Catlett found that a more accurate model could be generated using all the training data, rather than some randomly chosen subset. For each domain 20 trees were created from the training cases, then assessed using the test cases. Note that while the theory from the *N* cases may be only marginally better than the *N/2* tree, the size of the better theory is 30% to 90% bigger; i.e. more examples prompted a significant reorganisation of the model (exception: the *demon* domain) [30]. *Conclusion:* We never know enough to create the correct model.

Summarising the Garvin ES-1, PIERS, and Catlett results, we say that any new experience can lead to significant re-organisations of a model. As knowledge bases are maintained, they will encounter such new experiences. The re-organised model will need to be tested. Therefore, testing is an on-going component of knowledge maintenance.

			% error rates				Tree size			
			A		B		A		B	
Domain	training cases	test cases	mean	sd	mean	sd	mean	sd	mean	sd
demon	5000	2000	0.193	0.089	0.368	0.201	20.800	5.022	21.300	5.440
diff	5000	2000	1.582	0.313	2.268	0.380	87.700	6.131	59.800	6.101
othello	3000	2000	16.768	1.023	20.791	1.349	352.500	22.090	208.900	21.161
heart	3000	2000	2.889	0.617	4.389	0.542	74.200	9.611	45.900	6.882
sleep	3000	2000	25.242	0.647	27.617	0.847	234.700	20.787	135.200	17.686
hyper	5000	2000	1.081	0.195	1.286	0.265	26.100	4.833	15.000	4.304
hypo	5000	2000	0.597	0.191	0.699	0.199	27.700	4.692	19.100	4.025
binding	5000	2000	2.106	0.310	2.567	0.426	46.900	8.837	30.900	7.063
replace	5000	2000	1.408	0.340	1.756	0.339	30.700	7.658	22.100	6.274
euthy	5000	2000	0.556	0.178	0.909	0.249	37.400	3.705	28.000	4.425
wave	5000	2000	24.799	1.008	25.931	0.908	625.000	41.085	326.900	23.633

Table 3.5. *When do we know enough to create a correct model? Catlett's results show that in all cases,* N *training examples produces a better model than* N/2. "A" *denotes using* N *case and* "B" *denotes using* N/2 *randomly selected cases. From [30].*

3.5.1. The Need for Continual Testing

Testing can reveal the presence of bugs, but never prove their absence. External test suite assessment must be repeated whenever new data comes to hand describing appropriate system input/output. Most models are developed in poorly-measured domains. Hence, we would expect to be using some model before all entities in the domain have been measured. For models in routine use, we would expect it to sometimes encounter new measurements from the domain. Whenever such new data arrives, the model should be re-tested. That is:

- a model's test procedure should be a constant background process.
- the test process continues throughout a model life-cycle from KA to knowledge KM.

3.6. Summary

- Contemporary KA views its goal as model construction.
- Any model is a potentially inaccurate surrogate of the thing being modelled and must be tested.
- External test suite assessment is better than internal syntactic assessment.
- Test suite assessment in poorly-measured domains requires a qualitative reasoner.
- Most KBS domains are poorly-measured. Testing must therefore be an opportunistic process that is repeated whenever new data comes to hand. Pragmatically, this means it must be repeated during KA and KM.

- QMOD/JUSTIN was a facility for testing qualitative models in poorly measured domains using external test suite assessment.
- Therefore, systems like QMOD/JUSTIN are an essential component for modern KA and KM.

I can accept the theory of relativity as little as I can accept the existence of atoms and other such dogma: Ernst Mach.

If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.: Henry David Thoreau. *Our life is frittered away by detail… simplify, simplify:* Henry David Thoreau. *For never anything can be amiss, when simpleness and duty tender it:* Theseus, A Midsummer's-Night's Dream V.I. *Everything should be as simple as possible, but not simpler:* Albert Einstein.

4. Simple

In the previous chapter, we motivated the construction of a tester of indeterminate qualitative models that did not require measurements for all entities being modelled. In this chapter and the next, we describe how to build one. This description is an attempt to abstract above the QMOD-process and so is not defined in terms of QMOD-specific terminology. The next chapter describes a customisation layer that allows the core to be used for various domains. The customisations required for QMOD (called QCM) are described there.

We base our claim on simplicity is based on the pseudo-code described in this chapter. This pseudo-code is less than 350 lines long and is a near-complete specification of our core algorithm (including optimisation techniques and low-level implementation details). This pseudo-code was generated from a reverse engineering of a type-less object-oriented implementation (in Smalltalk). It dramatically simplifies the description of our prototype (hacked together over 15 months) and contains numerous enhancements to the existing system. The pseudo-code is expressed in a Pascal-like syntax[80].

In order to understand this code, we first present a tutorial example of our process. A preliminary complexity analysis will then show that a basic chronological backtracking algorithm (i.e. no memoing) is inadequate for this domain. We need to represent the assumption space explicitly. Hence, we will use a truth maintenance system. The pseudo-code for this system will then be presented.

Caveat emptor: this pseudo-code is a paper design of the as-yet unimplemented next version of our system and may contain small errors.

[80] For specifications of non-intricate procedural processing, we prefer using Prolog. We turned to Pascal after several attempts to specify this system in pure logic, Prolog, an object-oriented notation, and an entity-relationship notation. However, in terms of describing the intricacies of our inference procedure, we found that

4.1. Example

We introduce our process with the following example. Suppose an expert has sketched up a diagram that reflects her intuitions about some domains (see Figure 4.1).

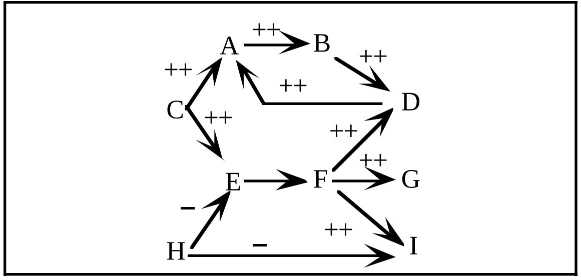


Figure 4.1: *A sketch of influence relations*

She tells a knowledge engineer that *"++"* means *"encourages"* and *"--"* means *"discourages"*. The knowledge engineer explores the expert's perception of encourage and discourages to develop the rules in Figure 4.2.

```
rule0: X ⇔ Y means (i)    X can lead to Y and
                   (ii)   Y could be explained by X.
rule1: if ++(X,Y) then X=up ⇔ Y=up and X=down ⇔ Y=down
rule2: if --(X,Y) then X=up ⇔ Y=down and X=down ⇔ Y=up
rule3: X can't be in two states at once;
       i.e. forbidden: (X=up and X=down) or (X=down and X=up)
```

Figure 4.2. *An expert's perception of the semantics of Figure 4.1*[81].

In an attempt to apply QMOD/JUSTIN style validation to the model, the knowledge engineer asks the expert for some example behaviour. In reply, the expert says that when *C* and *H* go *up*, *B, D*, and *G* goes *up* while *I* does *down*. For each output of the expert's example (i.e. *B=up, D=up, G=up, I=down*), the knowledge engineer writes down all ways the output can be linked back to the input (i.e. *C=up* , *H=up*), ignoring loops. There are seven such proofs *P* (see Figure 4.3).

```
P[1] = H=up→  E=down→ F=down→  I=down
P[2] = H=up→  I=down
P[3] = C=up→  E=up→  F=up→  G=up
P[4] = C=up→  A=up→  B=up→  D=up
P[5] = C=up→ E=up→  F=up→  D=up
P[6] = C=up→  A=up→  B=up
P[7] = C=up→  E=up→  F=up→  D=up→  A=up→  B=up
```

Figure 4.3: *Seven ways to prove {*B=up, D=up, G=up, I=down*} given {*C=up, H=up} *using Figure 4.1.*

The knowledge engineer notices that some of these proofs differ in what *assumptions* they make (i.e. the statements they make about items we have no direct knowledge; e.g. *A=up, B=up, E=up, E=down,F=up, F=down*). Some of these assumptions are mutually exclusive. For example, `P[1]` assumes *E=down* while `P[3]` assumes *E=up*. Each set of internally consistent assumptions defines a *world*. The proofs that are compatible with

[81]

that world can explain certain outputs. Our knowledge engineer writes down each world, and the outputs they can explain (see Table 4.1).

World #	*Maximal consistent subsets of the assumptions.*	*Contains*	*Explains*
1	`A=up, E=up, F= up`	`P[2], P[3], P[4], P[5], P[6], P[7]`	`I=down,G=up, D=up,B=up`
2	`A=up, E=up, F=down`	`P[2], P[4], P[6]`	`I=down,D=up,B=up`
3	`A=up, E=down, F=up`	`P[2], P[4], P[6]`	`I=down,D=up,B=up`
4	`A=up, E=down, F=down`	`P[1], P[2], P[4], P[6]`	`I=down,D=up,B=up`

Table 4.1: *The proofs of Figure 4.3 make certain assumptions. Maximal consistent subsets of those assumptions define four worlds. Each world can explain different outputs.*

Our knowledge engineer notices that one world contains everything we wanted to explain (`World[1]`). That is, there exists at least one set of assumptions in which the model can explain all known behaviour. Hence, the model of the expert's intuitions is not invalid, with respect to known behaviour.

The expert is impressed. "You've just tested an informal qualitative model without requiring much data." she says. "Normally, I would have to spent months in the labs determining the precise quantitative details of competing influences. You've given me a tool that lets me quickly build and check vague intuitions. This could save me a lot of time. Hey, I've got lots of drawings like that and lots more examples. Lets do it for all of those".

The knowledge engineer builds a depth-first search procedure in Prolog to automate the above process, then notices that the runtimes are exponential on model size. The Prolog system[82], while semantically elegant, could only handle very small models (only a few dozen vertices). Returning to Table 4.1, our knowledge engineer realises that there is some wasted computation involved in assessing combinations of assumptions that are impossible. For example, note that no path contains both the *E* and *F* assumptions of `World[2]` and `World[3]`. The reason for this is clear from the model of Figure 4.1: *F* is fully dependant on *E*. Hence, assumptions of different values for *E* and *F* will never be occur in the one proof. Ideally, the Prolog program should only have generated `World[1], World[4],` and perhaps another world containing the proofs using none of the controversial assumptions (e.g. *E=up*).

After thinking about the complexity of the program he is implementing[83], our knowledge engineer realises that he requires a fundamentally different approach using a specialised proof procedure. Accordingly, he reaches for his pseudo-Pascal compiler and

[82] Partially described in the appendix to [153]. Note that this code incorrectly handles explaining "steadies".

[83]

builds the system defined below[84]. The new system explicates structures that are only tacit in the chronological backtracking system (i.e. assumptions sets and worlds).

Note that the observed runtimes of the resulting system are still apparently exponential on model size. However, our experimental results shows that the new system does tame the runtime growth sufficiently to permit the processing of models at least as large as those seen in contemporary knowledge engineering practice[85]. Further, the new system runs 127 times faster than QMOD[86] (which used a chronological backtracking hypothesis tester) and can handle 18 times as many experimental comparisons.

4.2. Complexity

4.2.1. Search Space Size

The process described above can be summarised as *generate and test*: all worlds are generated, then assessment by some assessment operator. The operator above favoured the world(s) that includes the greatest number of outputs. In terms of complexity, this is an unfortunate choice of operator. The utility of each local inference has to be assessed by a meta-interpreter using some a global criteria "will it eventually lead to maximal coverage?". Such a global criteria cannot be applied till after all possible paths are collected; i.e. it cannot be used to cull the search space.

An inability to cull the search space is somewhat alarming since our search space can be very large. QMOD generated its input and output sets from a pair-wise comparison of a database of known experiments. For each pair of experiments, variables measured in both experiments are annotated *up*, *down*, or *steady*. These annotations become our output set. The input set is computed from the difference in the experimental context between the two experiments[87]. Outputs must be explained in terms of the inputs. Given a database of T experiments, and that a comparison of T_1 to T_2 is symmetrical to a comparison of T_2 to T_1, then there are $\alpha = (T^2-T)/2$ such comparisons[88].

Pearl & Korf describe depth-first-search (hereafter, DFS[89]) as follows:

[84] Our pseudo-code notation is described in section 1.5.

[85] See the section 6.3.

[86] See chapter two for a description of QMOD. See Figure 6.14 in section 6.2.3 for the runtime speed results.

[87] See the example of the data compiler in section 5.4.2.

[88] In the special case of non-symmetrical models, then $\alpha = (T^2-T)$. Non-symmetrical models are discussed in the section 5.4.2. For the purposes of this complexity analysis, we ignore this subtlety since it only effects a constant term.

[89]

> *Backtracking traverses the variables in a predetermined order, provisionally assigning … values to a subsequence* (X_1,…,X_i) *of variables and attempting to append it to a new instantiation such that the whole set is consistent…. If no assignment can be found for the next variable* X_{i+1}*, a dead-end situation occurs and the algorithm backtracks to one of the earlier variables and changes it assignment. [189].*

Hypothesis testing requires:

- One DFS from each *change* (i.e. an *up* or a *down* in the outputs) searching for any cause.
- Two DFS from each member of each pair of parent assignments that could combine to explain each *steady* . Why two? Steadies can be explained either by (i) their non-connection to exogenous events; or (ii) two parent nodes that want to send the *steady* node both *up* and *down* (net result being *steady*). If the average number of parents of a vertex is the branching factor *B*, then the number of possible assignment combinations is $(B^2-B)/2$. Each *steady* therefore implies $2*(B^2-B)/2 = B^2-B$ calls to DFS.

Let the number of steadies and changes be *S* and *C* respectively and $S \cup C$ be the outputs. We must call DFS β number of times where $\beta = C + (S *(B^2 -B))$.

In the language of truth-maintenance systems, the proofs we extract from a model are a *stable state* of a dependency network (i.e. contains no contradictions). A model is said to be *coherent* iff is has only one stable state [273]. In the case of knowledge acquisition in poorly measured domains where no absolute oracle exists (e.g. the neuroendocrinological domains studied by Feldman & Compton), we cannot guarantee that our models are coherent. Worse still, they may not even contain one stable state that includes all desired outputs and some known inputs[90]. Our search goal must therefore be the largest subset of the outputs which we can explain. A naive hypothesis tester would have to be repeated for all subsets of inputs and outputs; i.e. χ number of times where $\chi = 2^{C+S}$.

The outputs and inputs are vertices in a dependency graph between literals. Let the average number of parents of a node be the branching factor *B*. Let the number of vertices be *N*. Starting at $X = C + S$ outputs, there exist δ number of paths through the model where:

$$\delta = X * B^{N-B} * (B-1)!$$

In the case of a DFS from all vertices of a fully connected graph $B = N -1$, $X = N$, and this expression becomes:

[90] For example, in Table 4.1 of section 4.1, `world[2]`, `world[3]`, and `world[4]` are only able to explain a

$$\begin{aligned}\delta &= N*(N-1)^{N-(N-1)}*\big((N-1)-1\big)! \\ &= N*(N-1)^{N-N+1}*(N-1-1)! \\ &= N*(N-1)*(N-2)! \\ &= N!\end{aligned}$$

as we would expect [191][91]. The total search space is $\varepsilon = \alpha * \beta * \chi * \delta$ which is a function of $<C,S,N,B,T>$. Table 4.2 shows values for these parameters from the Smythe '89 model used in the QMOD/JUSTIN experiment[92].

Item	*Notes*	*Value*
C	Average number of changes	3.98
S	Average number of steadies	1.29
B	Branching factor (average number of parents)	1.5
N	Number of vertices in the model	80
T	Number of experimental contexts	31

(i)

α	Number of experimental comparisons	465
β	Average number of depth first searches called	4.95
χ	Average number of output subsets	38.6
δ	Number of paths per depth first search	$3.5*10^{14}$
ε	Total search space: $\alpha * \beta * \chi * \delta$	$3.11*10^{19}$

(ii)

Table 4.2: *Search space size* ε *for hypothesis testing assuming DFS. Table (i) shows figures from one model. Table (ii) shows the search space calculation.*

Table 4.2. shows that the search space ε is dishearteningly large. This worst case behaviour is not seen in practice. Nevertheless, hypothesis testing is a very slow process. The QMOD/JUSTIN study, restricted to the 24 comparisons with only one input and no steady outputs, terminated in 2 days. An analysis of the all 465 QMOD comparisons would therefore take *(2/24) * 465=38.75* days. This runtime is too slow to be practical.

Clearly, we need to tame the complexity of hypothesis testing. Our earlier prototypes (HT2, HT3) explored different techniques for taming this complexity. HT2 and HT3 were Prolog programs that applied knowledge of constraints[93] incrementally during the generation of the proofs to cull the search space prior to applying world assessment. Various additions were tried such as growing the subset of explainable outputs from smallest to largest and not growing to size *i+1* if size *i* was inexplicable[94]. HT2 and HT3 demonstrated that incremental constraint application was insufficient to tame our

[91] Previous descriptions of this expression [154] ignored the case where X <> N.

[92] See Figure 2.3 before section 2.2.2.

[93] I.E. two nodes can't be up and down in the same world.

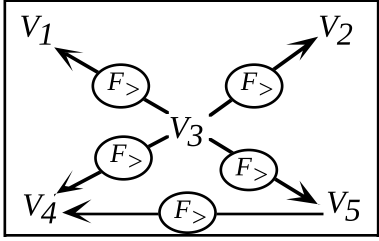
[94]

complexity. HT3, running overnight on the model of Table 4.2.i. did not terminate for even one comparison.

4.2.2. Avoiding Backtracking

4.2.2.1. The Backtracking Problem

The fundamental problem of HT2 and HT3 was the use of DFS. DeKleer [56] and Mackworth [135] caution against chronological backtracking. If the search algorithm learns some feature of the domain, it can forget it on backtracking and be doomed to re-learn that feature later (e.g. Figures 4.4 & 4.5). Such repeated effort can significantly increase runtimes.



Domain of V_1, V_2 = {a,b,c}
Domain of V_3, V_4, V_5 = {a,b}

$F_>$ is a constraint between neighbouring values that demands that upstream vertices are strictly lexicographically greater than downstream vertices; i.e. $F_>(a,b) = F_>(a,c) = F_>(b,c) = true$

Figure 4.4. *The problem: assign letters to each vertex such that parents are strictly lexicographically before their children. Note that no such assignment is possible for this model. Example from [135].*

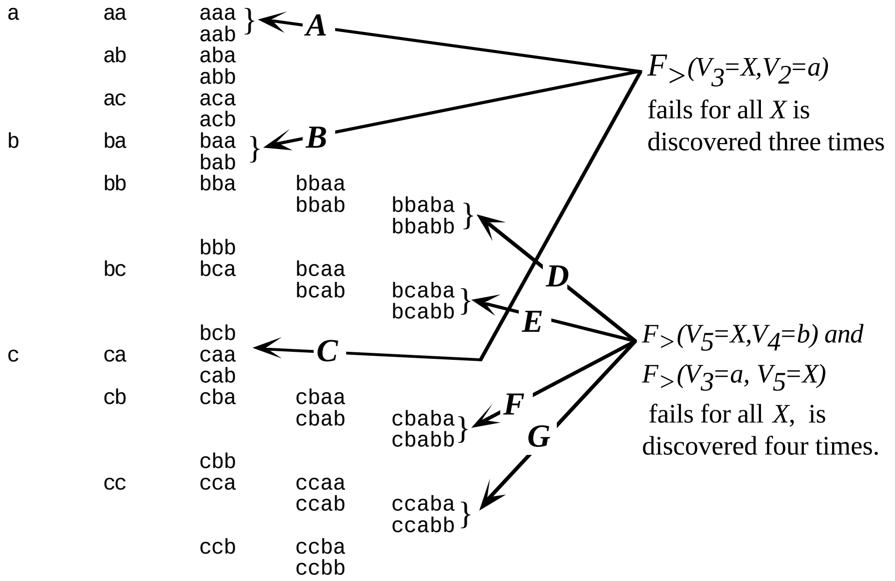


Figure 4.5: *A basic chronological backtracking algorithm (i.e. no memoing) trying to solve the Figure 4.4 problem. State assignments for* V_1, V_2, V_3, V_4, V_5 *are tried left to right.* ccabb *(for example)* = {V_1=c, V_2=c, V_3=a, V_4=b, V_5=b}. *There exists no assignment* X *such that* $V_3=X < V_2=a$. *This is discovered three times at points* ***A,B,C.*** *There exists no assignment such that* $V_3 < V_4$ & $V_3 < V_5$ & $V_4 < V_5$. *This is discovered four times at points* ***D, E, F, G.*** *Figure from [135].*

Pearl & Korf discuss *look-ahead schemes* and *look-back schemes* techniques for reducing the wasted CPU cycles from DFS [189]. Look-ahead schemes make an informed guess of what assignment to make next, based on meta-knowledge of (e.g.)

ends occur; i.e. what variables should we return to and explore alternative assignments. Simple chronological backtracking (i.e. the approach taken in Figure 4.5) returns to the last bound assignment and tries another. More intelligent look-back schemes are either *go-back to source of failure* or *constraint recording*.

4.2.2.2. PC-3

Mackworth's PC-3 algorithm is a constraint recording system [135]. PC-3 algorithm reduces the need for chronological backtracking by a pre-processor that performs a global analysis of valid assignments.

First, PC-3 discards invalid assignments to single vertices (node consistency), then to pairs of vertices joined by a single edge (arc consistency), then to triples of vertices joined by two edges (path consistency). Since all paths are sets of two adjacent edges, then once path consistency is achieved, all paths of more than two edges must also be consistent. The resulting set of assignments is a smaller search space for a chronological backtracker. In essence, PC-3 generates a search space containing all the stable states.

This approach, while achieving polynomial inference times, has several disadvantages:

- The global analysis of PC-3 precludes the generation of intuitive proof trees of the form "this assignment was made which lead to that state assignment". Systems based on this technique (e.g. the QSIM system [125][95]) find it hard to provide explanations for their behaviour.
- Let S_{av} and N be the average number of states per vertex and the number of vertices respectively. PC-3's time complexity $C = S_{av}{}^{5} * N^{3}$ [136]. For QMOD/JUSTIN model[96], $S_{av} = 3, N = 97, C = 2.22E8$. While this number is smaller that the DFS-based complexity value[97], it is still very large. Nor does it represent all our processing. Since we cannot guarantee that our models contain even one stable state, we would have to repeatedly call PC-3 on all subsets of models, adding an unwanted exponential term to the time complexity.

4.2.2.3. JTMS

Doyle's truth-maintenance system (JTMS[98]) is a *go-back to source of failure* or "dependency-directed backtracking approach" [70]. A JTMS maintains a dependency network between beliefs. When a dead-end is detected, the JTMS queries the data dependencies network and only reinstantiates decisions that are implicated in the failure.

[95] See Figure 2.3 before section 2.2.2.

[96] Described in Table 4.2, section 4.2.1.

[97] See Table 4.2.ii.

[98] Originally, Doyle's system was just *the* TMS. However, after the invention of the ATMS, an extra "J" was

All JTMS propositions P are augmented by two sets of beliefs: P_{in} and P_{out}. P is believed if $x \in P_{in}$ are believed and no $y \in P_{out}$ is believed. System premises have empty in and out lists. Conclusions reached by simple monotonic deduction have empty out lists and non-empty in lists. Default beliefs, i.e. things we can believe unless we have evidence against them, is represented by propositions with non-empty out lists.

The interesting case is contradiction: when members of P_{in} and P_{out} are both believed; i.e. we have reasons for and against believing P. In this case, the JTMS explores the dependencies of P_{in} and P_{out} looking for default beliefs we can retract to removes the contradiction. The other propositions that use the now retracted default beliefs in their P_{in} and P_{out} lists must then be checked, which may result in changes to the belief status of other default beliefs. This recursive process ripples out over the network till a stable state is reached; i.e. no belief is contradictory.

A JTMS stable state is an hypothesis testing *world*. The recursive traversal of the dependency network looking for retractable assumptions is called *world-switching.* DeKleer notes that in highly-dynamic systems where new beliefs are frequently arriving or in systems with many worlds, a JTMS can spend the majority of its time world-switching [56]. This can be an expensive process since the TMS forgets world i when it switches to world $i + 1$. If the system later needs to return to world i , then all the computation involved in generating and testing world i must be repeated.

While theoretically interesting, the JTMS does not help the hypothesis testing problem. The general case in hypothesis testing is that some subset of the model is stable. If the subset is less than the entire model then the JTMS will not terminate on a stable state. Like PC-3, if we were to use the JTMS for hypothesis testing, then we would have to call it on each subset of the model (i.e. adding an exponential term to the time complexity).

4.2.2.4. ATMS

To avoid the overhead of world-switching, DeKleer's assumption-based truth maintenance system (ATMS) builds and caches the information required to quickly determine membership of a literal in multiple stable states [56-58, 85]. Each world is defined in terms of the *minimal environments* . An *environment* is a set of assumptions which can be used to infer that environment. Environments must not violate a *nogood* set; i.e. sets of literals which if believed at the same time, would create a contradiction. DeKleer's insight into the truth maintenance process was that an environment contains a *minimal* (with respect to size) subset which uniquely determine the other assumptions in that world. Defining worlds in terms of their minimal environments saves time and space.

In DeKleer's scheme, an inference engine incrementally passes to an ATMS *justifications* for any inferences it makes. Each proposition in the ATMS is *labelled* with the set of minimal environments in which it hold. The inference engine can ask the ATMS if a proposition *p* is believed, with respect to a set of minimal environments (see Figure 4.6). This is a simple matter of checking a node's label.

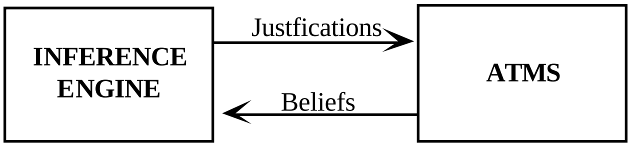


Figure 4.6: *The DeKleer ATMS architecture*

Expressed in terms of a directed dependency network:

- A justification is a node, with a list of parent nodes it was inferred from. The ATMS incrementally adds this to an evolving dependency network, recomputing the minimal environments as required.
- A minimal environment is computed from a traversal upstream from the justification's node. If an assumption is dependant on an upstream assumption, the downstream assumption is not added to the minimal environment.

Note that there is no world-switching in this architecture. When considering all possibilities (i.e. all worlds), then we need only switch between minimal environments.

For an example of the labelling process, suppose the justifications of Figure 4.7.i had been passed to an ATMS. Conceptually, these form the dependency network of 4.7.ii. The labels for the propositions of 4.7.i are shown in 4.7.iii.

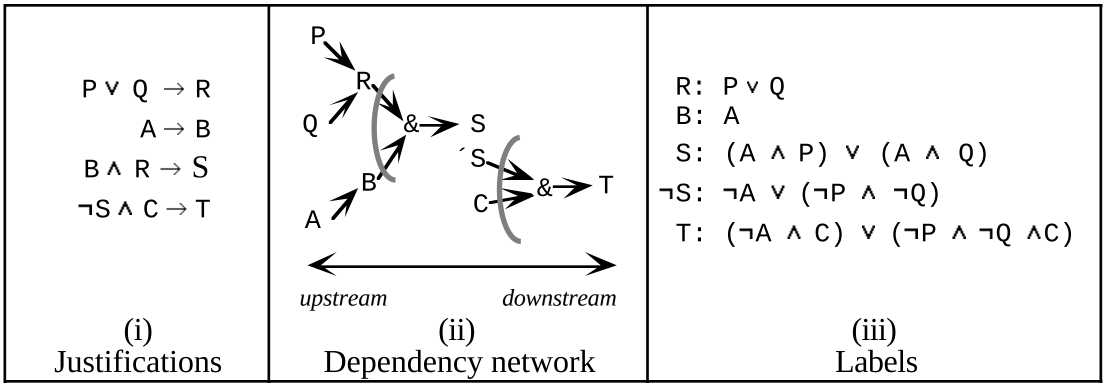


Figure 4.7. *A problem solver has passed the justifications of figure (i) to an ATMS which computes the labels of figure (iii). Figure (ii) shows the dependency network tacit in figure (i). Example adapted from [95].*

Formally, an ATMS world is an extension of Reiter's default logic [76, 216, 218]. Formulae in default logic have three parts: (i) a *prerequisite*; (ii) a *consequence*; and (iii) a special test call the *justification*. If the prerequisites are known, and the justifications are consistent (i.e. their negation can't be proved), then the consequence can be inferred. Should the justification later become inconsistent, then the consequence must be

An *extension* E of a default theory is a set of literals from the theory with consistent justifications. All formulae whose prerequisites are satisfied by E and whose justifications are consistent with E are also in E. An extension can be interpreted as an acceptable set of beliefs one may hold about the world. Like the worlds of hypothesis testing, a theory can generate multiple extensions. A model's total envisionments[99] can be computed via the generation of all possible extensions. Each extension would be one envisionment.

Despite DeKleer's early claims about the efficiency of the ATMS ([56], p153), the system proved to exhibit exponential runtimes on input size [232]. While it is true that the overheads of world-switching are avoided, the minimal environments have to be precomputed and cached. This is a fundamentally slow process. It can be shown that this process is isomorphic with known NP-hard problems [27, 232]. DeKleer himself has moved away from pure dependency-directed backtracking [64] and now uses probabilistic heuristics to tame the complexity of his technique [66].

4.2.2.5. Adapting the ATMS

Since only a subset of our models can form a stable state, hypothesis testing cannot use PC-3 or the JTMS. The ATMS seems more promising. It permits reasoning about a subset of the total model, while explicitly representing the assumption sets that were manipulated in the example at the start of this chapter. However, the ATMS does not remove the exponential runtime problem. Hence, we elect to experiment with variants to the standard ATMS approach. For the sake of efficiency:

- We restrict definitions of invariants to an arity of two. This has the advantage that, given one literal, its contradictory literals can be quickly determined. Most of the examples of invariants we see in the literature have an arity of two (typically, $invariant(X, \neg X) \vdash false$).
- We assumes a *parts-based qualitative ontology*. That is, our models comprise entities that may be in one of N *mutually exclusive* states. This will simplify the worlds search (see the discussion below on *world excluding assumptions*).
- We will use bitstrings for set manipulations. A `bitstring` is a set of integers whose binary representation denotes set membership. For example, if we have 4 vertices, then a `bitstring` denoting `vertex` 1 and 3 would be 4 bits wide with bits 1 and 3 set (i.e. 0101; note that bitstring places are counted from the right-hand-side). Sets manipulation defined over bitstrings can be executed by very direct calls to hardware. Assuming that a bitstring holds one integer, then: {} = 0; $S_1 \cup S_2 = S_1$ bit-and S_2 ; $S_1 \cap S_2 = S_1$ bit-or S_2; $S_1 - S_2 = S_1$ bit-and $\neg S_2$ (where $\neg S_2$ denotes reversing all 1's with 0's and visa versa); *subset test*: S_1 -

[99] I.E. the set of all consistent behaviours inherent in some fixed collection of objects in some configuration, see

$S_2 = 0$. If the number of required bits is greater than the bits in the local implementation of an integer, define a bitstring as a list of integers. The above operations are then defined as an iteration over that list. Certain processes (e.g. returning a list of integers for the set bits in a bitstring) can benefit from a pre-computed cache of all the possibilities.

- We restrict ourselves to propositional theories. Logical theories with variables are not supported since such variables can generating (potentially) an infinite number of terms. Such an infinite set would represent an infinite search space for an hypothesis tester searching for all possible paths.

Hypothesis testing has no other "problem" other than world creation. Hence, we do not separate the problem solver from the ATMS. Nor does hypothesis testing require incremental updates to the dependency network/ minimal environments. We can compute our minimal environments in one batch run.

We will use domain knowledge to reduce the search space as much as possible:

- A small pre-processor computes the *relevant* literals; i.e. those reachable from some input set.
- Hypothesis testing extensions can be restricted to the *relevant envisionment*;[100]. The relevant envisionment is computed by growing proofs backwards from outputs over relevant vertices. The union of the vertices downstream from the inputs (i.e. the relevant ones) and upstream from the outputs is the relevant envisionment.

For example, consider the hypothesis testing problem:

```
model:         b if a; c if a; e if (b or c);
               d if b; f if c; z if y.
contradicts:   {d,f}
inputs:        {a}
outputs:       {e}
```

The relevant literals are *{a,b,c,d,e,f}* and the irrelevant literals are*{z,y}*. Hypothesis testing would generate two proofs which could exist in the one world:

```
P[1] ← {a,b,e}; P[2] ← {a,c,e}
```

We need not label *d* or *f*, since these are not required for proofs of outputs *{e}* with respect to known inputs *{a}*. We need not label *{z,y}* since they are irrelvant to proof generation. Standard ATMS/ default logic would label all literals to generate two extensions (one with *d* and the other with *f*). Both of these extensions would contain the same proofs of *E* in terms of *A*. We view this as wasted computation. Hence our restriction to the relevant envisionment. Note, however, that if Reiterian extensions were

[100]

really required, hypothesis testing could still support it. We would merely ask the hypothesis tester to try and prove all non-input literals in the theory.

Input to the hypothesis tester are four sets of vertices supplied by the knowledge engineer: *inputs*, *outputs*, *maybes and facts*. These sets specify the search goals, and the search space. Explanatory proofs are attempted for all members of *outputs*. A *proof* is a list of adjacent vertices that connect that *output* back to any member(s) of *inputs*. The space we are allowed to search for proofs is the *maybes* set. *Facts* must not be contradicted[101]. Prior to inferencing, we can categorically rule out assumptions that contradict *facts*.

These four sets serve to restrict the general ATMS problem to the specific hypothesis testing problem at hand. This search restriction, while heuristically useful for culling the search space, precludes the use of some classic-ATMS-style pre-processor that generates all possible extensions.

Since we use relevant envisionments, our label generation tacitly assumes some set of inputs and outputs. When moving between generated worlds, we can safely assume that they were all generated using the same inputs and outputs. Our definition of a minimal environment hence ignores the roots of the dependency network. An environment of a hypothesis testing world is defined using the *base controversial assumptions* .

- An assumption is *controversial* if it belongs to the *contradicts* set of any relevant literal.
- A controversial assumption is *base* if it is not dependant on any other controversial assumption. Note that these may be vertices internal to the dependency network. Hence, we should not always "unwind" our labels right back to root nodes[102].

For example, recall the proofs generated in our introductory example[103] using *outputs* = *{I=down, G=up, D=up, B=up}* using *inputs* = *{H=up,C=up}*: If *facts* = *inputs* $\cup$*outputs*, then these proofs have made assumptions for *E=down, F=down, E=up, F=up, and A=up.* Assuming that no single variable can be in two states simultaneously, then the *E* and *F* assumptions are controversial. Of these, *E* is always upstream of *F*, so the base controversial assumptions are *E=up*, and *E=down*.

From the base controversial assumptions, we can compute the list of minimal environments: i.e. subsets of the base controversial assumptions that are consistent and maximal (with respect to size). To compute this list, we:

[101] In the QMOD case, *facts = inputs* $\cup$ *outputs.*

[102] For an example of a full unwind, see Figure 4.7.iii. in section 4.2.2.4.

[103]

1) Extract a list X of all the variables mentioned in the base controversial assumptions. Examples:

   (i) For our above proofs[104], X_1 = *{E}*;

   (ii) If the base controversial assumptions were *{day=mon,day=tues, time=am,time=pm, who=paul, who=tim},* then X_2 = *[day,time,who].*

2) Augment this list with the states of these variables in the base controversial assumptions to form *Y*. Examples:

   (i) From X_1 we get Y_1 = *[{E=up, E=down}]*;

   (ii) From X_2 we get Y_2 = *[{day=mon,day=tues}, {time=am,time=pm}, {who=tim, who=paul}],*

3) Compute all combinations of *Y[i]* with *Y[j]* for all *i,j* in *Y*. To this list, we add the empty set to represent the environment where no assumptions were required to form the minimal environments. Examples:

   (i) From Y_1 we get:

```
minimalEnvironment[1] = {}
minimalEnvironment[2] = {E=up}
minimalEnvironment[3] = {E=down}
```

   (ii) From Y_2, we get:

```
minimalEnvironment[1] = {}
minimalEnvironment[2] = {day=mon,   time=am, who=tim}
minimalEnvironment[3] = {day=mon,   time=am, who=paul}
minimalEnvironment[4] = {day=mon,   time=pm, who=tim}
minimalEnvironment[5] = {day=mon,   time=pm, who=paul}
minimalEnvironment[6] = {day=tues,  time=am, who=tim}
minimalEnvironment[7] = {day=tues,  time=am, who=paul}
minimalEnvironment[8] = {day=tues,  time=pm, who=tim}
minimalEnvironment[9] = {day=tues,  time=pm, who=paul}
```

4) Reject any minimal environment that contains contradictory assignments. In our examples, no such contradictory minimal environments exist.

The label of an output in an hypothesis tester is the list of worlds that can support its proof. More precisely, an output is in a world if its proof does not contradict the minimal environments of that world. This is consistent with the general hypothesis testing approach of reasoning generously and condoning all possible inferences.

To compute proof membership in a world, we first compute the list of *world excluding assumptions*; i.e. *worldExcluding[i]* equals the assumptions that contradict the *minimalEnvironment[i]*. A proof is in a *world[i]* if it does not overlap with *worldExcluding[i]*. In our parts-based ontologies, the *world excluding assumptions* are the states mutually exclusive with the minimal environments. Also, *worldExcluding[1]*

[104]

are the base controversial assumptions (hence, a proof in *world[1]* uses no controversial assumptions). Examples:

(i) From Y_1, the world excluding assumptions are:

```
worldExcluding[1] = {E=up, E=down}
worldExcluding[2] = {E=down}
worldExcluding[3] = {E=up}
```

(ii) The world excluding assumptions from Y_2 are:

```
worldExcluding[1] = {day=mon,day=tues,time=am,
                     time=pm,who=paul,who=tim}
worldExcluding[2] = {day=tues,    time=pm,    who=paul}
worldExcluding[3] = {day=tues,    time=pm,    who=tim}
worldExcluding[4] = {day=tues,    time=am,    who=paul}
worldExcluding[5] = {day=tues,    time=am,    who=tim}
worldExcluding[6] = {day=mon,     time=pm,    who=paul}
worldExcluding[7] = {day=mon,     time=pm,    who=tim}
worldExcluding[8] = {day=mon,     time=am,    who=paul}
worldExcluding[9] = {day=mon,     time=am,    who=tim}
```

The core of hypothesis testing is two nested `for` loops that iterate over (i) each world excluding assumptions set and (ii) each proof. If no overlap is found between *proof[i]* and *worldExcluding[j]*, then *proof[i]* is added to *world[j]*. That is:

```
function sort_into_worlds[105]
begin  exclusions ← worldExcludingAssumptions(Assumptions)
       for x in size(exclusions) do
         begin  W[x] ← ∅
                for p ∈ P do
                    if     p ∩ exclusion[x] = ∅
                    then   W[x] ← W[x] + p
end      end;
```

Completing our Y_1 example, we find that:

```
world[1] = proof[2], proof[4], proof[6]
world[2] = proof[2], proof[3], proof[4], proof[5], proof[6], proof[7]
world[3] = proof[1], proof[2], proof[4], proof[6]
```

This corresponds to three worlds we should have generated ideally in our introductory example[106]. Note that we have generated them without chronological backtracking. As our inference proceeds, it discovers and caches crucial information about the search space (i.e. the possible proofs and the base controversial assumptions).

We can calculate the base controversial assumptions (and hence the world excluding assumptions) as a side-effect of proof generation. As proofs rise from outputs to inputs, they can carry with them the set of controversial assumptions encountered along the way. On arrival upwards at a new controversial assumption, the proof's list of controversial assumptions is replaced by the newly-encountered assumption. On arrival

[105] A more-complete specification of this function may be found at the end of section 4.3.7.

[106]

at an input, the proof's controversial assumption set is guaranteed to be base. This algorithm ensures that the base controversial assumptions are restricted to only those used by proofs.

This proof/world generation process grants no special semantics to negated propositions. We saw above that the ATMS-label of a negated proposition $\neg X$ was computed from X[107]. Hypothesis testing treats literals in a uniform manner and does not automatically connect $\neg X$ and X . Vertices in the dependency network are created for each proposition or negated proposition used in the model. Knowledge that $\neg X$ and X cannot appear in the same world is supplied by the knowledge engineer who includes $\neg X$ in the contradicts set of X and visa versa. If $\neg X$ appears internal to the dependency network, and if we have to prove it, then we do so via an exploration of its upstream neighbours. If $\neg X$ appears at the roots of the dependency network, and we have to prove it, then we do so via testing for its membership of the supplied list of known *facts*.

4.3. Pseudo-Code

4.3.1. Core

Core is a function that returns the Best worlds that explain some subset of the Example of known behaviour using a Model that is a graph of dependencies between literals[108]. Best is a subjective judgement. The Best operator of our example returned the worlds that explained the largest number of outputs.

```
function core(   Model         : graph;
                 Example       : behaviour;
                 Upstream, Best: method;
                 Options       : settings
            ) : nworlds     -- Return the "best" worlds
type -- Definition of data types (see below)
var  StartTime: integer;
     ExplainedViaIsolation : bitstring;
     E: edges; V: vertices;  -- Model components

begin  cleanup; setup;
       if bad_example
       then return Options.badExampleMarker
       else begin  mark_relevant_verticies;
                   ExplainedViaIsolation ← missed_missables;
                   mark_contraversial_assumptions;
                   return  Best(sort_into_worlds(all_proofs),
                                Example);
end         end;
```

Certain details that are explained later (i.e. Options, StartTime, cleanup, setup,bad_example, badExampleMarker, ExplainedViaIsolation,

[107] For example, see the label for *T* in Figure 4.7.iii. in section 4.2.2.4.

[108]

missed_missables Upstream, mark_relevant_verticies, mark_ contraverisal_assumptions).

The inner secrets of the structure of a literal are mostly invisible to Core. The only time Core ever looks deeper than a literal is at the end of the processing to see what objects are involved (see below, the world_excluding_ assumptions procedure). For the moment, we will say that (i) a literal points to some proposition; (ii) a proposition is some test that at some time, an object had a certain value; (iii) that an object has a unit id and belongs to one of a set number of pre-determined classes; (v) and a literal points to a vertex on the dependency graph. That is:

```
any      = … -- some local implementation of polymorphism
method   = … -- a function name bound at runtime
class    = record name : string;
                  domain: list of any;
                   kind: {discrete, continuous} end;
object   = record  name: string; id: integer;
                   class: class;
                   familiarity, probeCost: integer; end;
proposition = record  object: object; value: any;
                      test:{≤,<,=, ≠,>,≥};
                      time: integer; end
literal  = record  negated: boolean;
                   p : proposition;
                   v: vertex  end;
literals = list of literals;
```

4.3.2. Models as Graphs

The Model is generated from some source statements entered by the user. For example, the rules of Figure 4.8.i form the model of Figure 4.8.ii.

```
r1 if a=up or b=2 then c=yes
r2 if not d<7 and c=down then e=yes
r3 if b=1 then e=yes
r4 if not f=yes and e=yes then not g=no
```

(i)

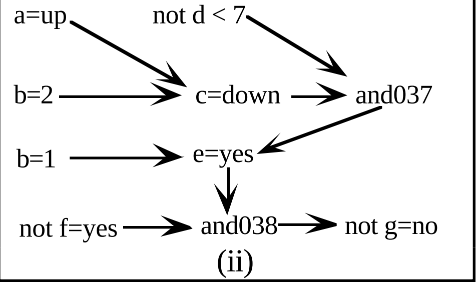


Figure 4.8. *Converting (i) the source statements from a knowledge base into a (ii) model.* And037 *and* and038 *are* and *vertices that uniquely identifiers generated for the used conjunctions. The diagrams assumes a declarative domain; i.e. the time field for each propositions is the same (hence, the time field values are omitted).*

Model comprises a set of edges E and vertices V, each with their own unique id. Each vertex V[x] is either an *and* vertex or an *or* vertex (denoted by the boolean field and. If not V[x].and, then the vertex is an *or*). Each edge E[x] has a source pointer to the statement which generated it (e.g. in Figure 4.8.i., one of *r1, r2, r3, r4*) as well as a kindOf fields denoted the edge type (e.g. *implication*, *specialisationOf*, *part-of*, *processed-by*, *causes*). Not every edge type can be used for proving literals in terms of other literals. For example, it is not usual to generate explanations of some literal via a *part-of* relationship. However, Core has no knowledge of these different edge types. Selection of edges to be used for proof generation is done by the supplied function Upstream (specified below). The price of using an edge for a proof is its cost (default = 1)[109].

Edges are directed and point from some vertex to some downstream vertex. The reverse direction is called upstream. Edges also support two counters: (i) an attempts count which records the number of times we traverse the edge upstream looking for an explanation; (ii) a successes counter which records the number of times we return from that edge after an explanation. The routines success and trying increment these counters.

```
**procedure** success(es:edges; n : integer) **begin**
  **for** e ∈ es **do** e.succeses ← e.succeses + n
**end**;

**procedure** trying(es: edges; n : integer)  **begin**
  **for** e∈ es **do** e.attempts ← e.attempts + n
**end;**
```

One vertex is created for each combination of object-value-test-time found in the source. Using meta-knowledge of test and the domain of object, negation can be removed from literals. For example, *not day = tuesday* is also *day=monday or day=wednesday or … or day=sunday*. However, this expansion adds six vertices for six days into Model while the unexpanded literal adds only one vertex. In terms of runtime speed, the fewer vertices the better.

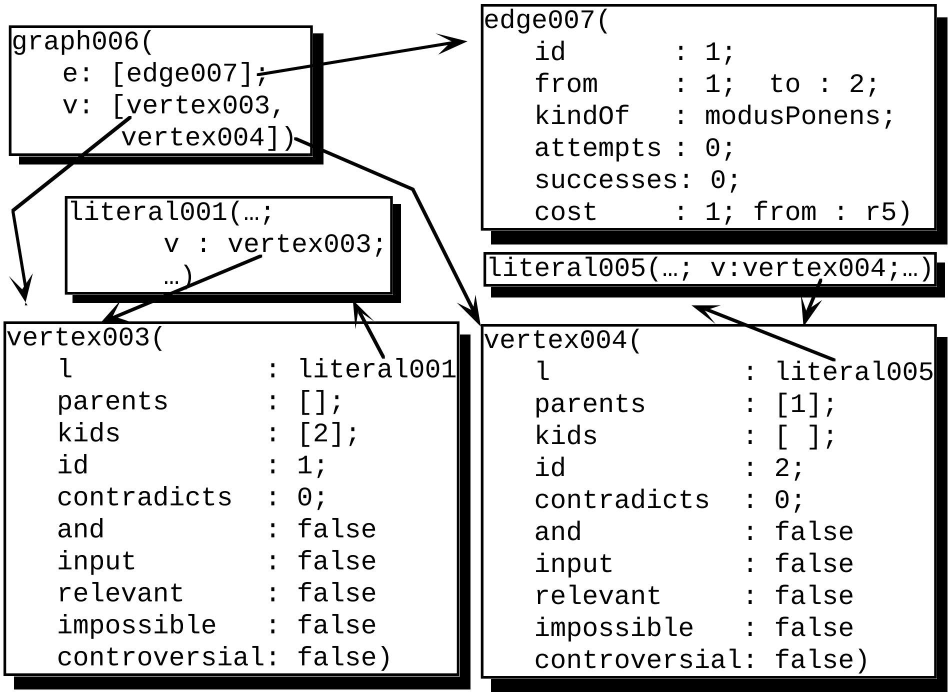
V[x].contradicts reflects the semantic invariants *I* and is the set of vertices that contradict V[x]. For example, since *b=1* contradicts *b=2* then the vertex for each such literal would contain a contradicts set that refers to the other[110]. Note that Core does not compute the contradicts set since this could require extensive meta-knowledge about incompatibilities in different objects. The Model passed to Core must have its contradicts already set.

[109] Edges may have different costs in the case of (e.g.) rules representing pre-conditions for the use of some procedure. For example, *date_of_birth=known then age=known* has the cost of calculating *age* from d*ate of birth*. This cost may be significant for complicated calculations (e.g. *person=known then life_expectancy=known* may trigger the application of a complicated maths package which computes the *life_expectency*).

[110] This implementation assumes that *I* has an arity of two (I/2). This was chosen for reasons of efficiency: given one literal, then its incompatible literals are known. I/2 is consistent with the examples we know of in the

```
source   = … -- some structure in the user view
posint   = 1 .. MaxInt;
posints  = list of posint;
vertex   = record  l : literal; parents, kids: posints;
                   id: posint; contradicts: bitstring;
                   and, input, relevant,
                   impossible, controversial: boolean;
           end;
vertices = list of vertices;
edge       =  record id : posint; from,to: posint; kindOf: thing;
                   attempts,successes,cost: integer;
                   source: source  end;
edges    = list of edge;
graph    = record e : edges ; v : vertices end;
```

For example, the rule *r5 if not person_age > 7 then toddler=true* could be compiled into a graph connecting the left-hand-side literal *not person_age* to the right-hand-side literal *toddler=true* :



```
graph006(
   e: [edge007];
   v: [vertex003,
       vertex004])

edge007(
   id       : 1;
   from     : 1;  to : 2;
   kindOf   : modusPonens;
   attempts : 0;
   successes: 0;
   cost     : 1; from : r5)

literal001(…;
     v : vertex003;
     …)

literal005(…; v:vertex004;…)

vertex003(
   l            : literal001
   parents      : [];
   kids         : [2];
   id           : 1;
   contradicts  : 0;
   and          : false
   input        : false
   relevant     : false
   impossible   : false
   controversial: false)

vertex004(
   l            : literal005
   parents      : [1];
   kids         : [ ];
   id           : 2;
   contradicts  : 0;
   and          : false
   input        : false
   relevant     : false
   impossible   : false
   controversial: false)
```

Figure 4.9. *Data structures generated from Figure 4.8.*

Each `vertex` has a unique `id` which is used in other vertices' `parent` and `kids` slots. The `contradicts` sets are empty bitstrings (0) since for all the vertices in our `graph`, there are no contradictions. With the exception of `and`, the boolean flags of each `vertex` are dependant on each `Example` (see below). Since a single Model can be processed by many examples, one duty of `Core` is to reset these flags.

```
procedure cleanup
var v: vertex
begin  for v ∈  V  do
         v.input← v.relevant← v.impossible← v.controversial← false;
end;
```

4.3.3. Low-Level Design Issues

Data modellers will note that there exists a one-to-one relationship between vertices and literals, yet we choose to separate them. Further, we seem to have made some arbitrary decisions regarding when a set is or is not a bitstring. All these design decisions are justified in terms of efficiency. Runtime inference concepts, such as the booleans reset in `cleanup`, are separated from the logical model of a literal. Sets we use a lot in the inferencing are stored as bitstrings in non-literal records.

Bitstrings are a very low-level representation. They can be processed very quickly and are very memory compact. These advantages notwithstanding, they have two drawbacks:

- Bitstrings enshrine the existence of the legal members of a set. This is a valid assumption for a universe of discourse containing a static number of entities. However, if the number of entities grows and shrinks, then bitstrings may retain set bits to now non-existent things. For simplicity's sake, `Core` assumes a fixed number of literals; i.e. it is defined for finite propositional theories.
- Bitstrings are somewhat pesky in that sometimes the program is handling (i) a bitstring set; or (ii) an integer offset into some other list; or (iii) a list of numbers generated from the bitstring.

`Core` needs the following bitstring functions defined. Note that their implementation should be as optimised as possible.

```
function bclear!(b: bitring; i: posint): bitstring
  -- clears the ith bit
function bempty(x : bitstring) : boolean
  -- true if no bits set in x
function bdifference(x,y: bitstring): bitstring
  -- bits in x but not in y
function bintersection(x,y:bitstring): bitstring
  --return bits in x and y
function bits(x: bitstring) : posints
  -- the set bits of x
function bunion(x,y: bitstring) : boolean
  -- return bits in x or y
function boverlap(x,y: bitstring): boolean
  return not bempty(bintersection(x,y))
function bset?(b: bitring; i: posint): boolean
  -- true if bit i is set
function bset!(b: bitring; i: posint): bitstring
  -- sets the ith bit
function bsubset(b1,b2: bstring) : boolean
  -- true if b1 ⊆ b2
```

For the purposes of model validation, Core should run to termination. However, we will later argue that Core's generation of consistent worlds from a model with respect to known data is a non-trivial description of knowledge-based inferencing. If Core is to be used as a general inferencing tool then, for domains that do not require the exhaustive enumeration of all possibilities, Core require heuristic abort points[111]:

- StartTime records when Core executes (see setup, below). Core aborts at time StartTime + Options.longTime.
- When the cost of a particular proof exceeds a Options.highCost, then that proof is aborted.
- Proofs from any vertex are only generated Options.enough times. For exhaustive domains, set this to MaxInt.

```
  options     =  record
                      highCost,enough: posint;
                      longTime : integer;
                      dodgeControversial: boolean;
                      badExampleMarker : any
                 end

function tooExpensive(i: integer): boolean
  return i ≥ Options.highCost

function tooSlow: boolean
  return now > StartTime + Options.longTime

function enough (successes) : boolean
  return tooSlow or successes > Options.enough
```

The field badExampleMarker is explained later.

4.3.4. Example Behaviour

Behaviour is defined as five sets of literals:

```
  behaviour   =  record inputs,outputs,
                        facts,maybes, missables: literals end;
```

Missables are literals that we can explain via their non-connection to inputs. For example, in the neuroendocrinological domain, we can explain a steady measurement via (i) two parents trying to drive it *both* up and down; or (ii) via its non-connection to exogeny (e.g. the reason why the rainfall did not change was because nothing affected it). Facts are the literals we believe in absolutely. Since we will never surrender our beliefs in facts, all vertices that contradict facts are impossible. Procedure setup initialises this impossible flag (as well as some other variables global to all functions/procedures within Core).

[111]

```
procedure setup
var l1,l2: literal; v: posint; begin
  V ← Model.v; E ← Model.e;
  StartTime ← now;
  for l1 ∈  Example.inputs do  l1.v.input ← true;
  for l1 ∈  Example.facts do
    for v ∈  bits(l1.v.contradicts) do V[v].impossible ← true end;
end;
```

Inputs and outputs denote the start and end points of an explanatory proof. Core is required to generate explanations for all literals within outputs. In our example from the start of the chapter:

```
Example.inputs   ← [C=up , H=up].
Example.outputs  ← [B=up, D=up, G=up, I=down]
Example.facts    ← inputs ∪ outputs
Example.maybes   ← [A=up,B=up,C=up,D=up,E=up,F=up,G=up,H=up,I=up,
                    A=down,B=down,C=down,D=down,E=down,F=down,G=down,
                    H=down, I=down]
Example.missables←[]
```

Behaviour can be used to control the search space. The search for explanatory proofs always avoid impossible, not relevant (defined below) vertices, as well as vertices that are not maybes.

```
function should_not_use (v: vertex) : boolean
  return not v.relevant or v.impossible or
         not bset?(Example.maybes,v.id) or
        (Options.dodgeControversial and v.controversial);
```

Core make no assumption that inputs, facts, and outputs do/ do not overlap. Such decisions are for the knowledge engineer. However, Core does assume that facts are consistent; i.e. the union of their contradicts set does not contain facts. Further outputs are assumed to be a subset of facts, and the missables must be a subset of outputs. Lastly, all the behaviour literals must be subsets of maybes (otherwise it is pointless to include them in a behaviour record). If this is not the case, then Core aborts proof generation and returns an error flag (Options.badExampleMarker)

```
function bad_example : boolean
var v : vertex
begin  with Example do
         if not bsubset(outputs facts)      then return true;
         if not bsubset(missables,outputs)  then return true;
         if not bsubset(facts,maybes)       then return true;
         if not bsubset(outputs,maybes)     then return true;
         if not bsubset(inputs,maybes)      then return true;
         if not bsubset(missable,maybes)    then return true;
       end;
       for v ∈ V do if  v.input and v.impossible then return true;
       return false;
end;
```

4.3.5. Relevant Vertices

Any vertex that is not reachable from the `Inputs` is irrelevant to proof generation. The precondition for a relevant *and*-vertex is all its parents (i.e. the vertices immediately upstream) being relevant. The precondition for a relevant *or* `vertex` is either (i) it is an `input` or (ii) one of its parents is relevant. As a heuristic to speed up proof generation, `Core` runs a small pre-processor to deduce relevancy prior to proof generation. The pre-processor is the `mark_relevant_verticies` procedure.

```
procedure mark_relevant_verticies (ls : literals)
var l:literals;
     begin for l∈ Example.inputs do visit(l.v.id);
end

procedure visit(v1: posint)
var v2: posint;
  function pre_conditions_satisfied(v1: posint) : boolean
  var v2: posint;
  begin  if V[v1].and then for v2 ∈ V[v1].parents do
           if not V[v2].relevant or V[v2].impossible
           then return false;
         else return true
  end;
begin  V[v1].relevant ← true;
       for v2 ∈ V[v1].kids do
         if not V[v2].relevant and preconditions_satisfied(v2)
         then visit(v2)
end;
```

The `preconditions_satisfied` function is called only when one of its parent has been found to be relevant. Hence, there is no need to test for a relevant parent within `preconditions_satisfied` in the case of *or* vertices.

Once we know which literals are relevant, we can determine which of the missables literals are explicable in terms of their non-connection to exogeny (i.e. their absence from the set of relevant vertices).

```
function missed_missables: bitstring
var out : bitstrings;
begin  for l∈ Example.missables do
         if not l.v.relevant then bset!(out,l.v.id);
       return out;
end;
```

4.3.6. Proofs

An explanatory `proof` is a set of connected edges that (i) are upstream of the `outputs`; (ii) downstream of the `inputs`; (iii) only include literals that are `maybes`; (iv) contain no literals that `contradict` other literals in the `proof`; (v) its roots (called the `proof`'s `inputs`) are members of `Example.inputs`; (vi) it has one leaf (called the `proof`'s `output`) which is a member of `Example.outputs`; (vii) and includes only relevant vertices.

`Proof`s contain a set of vertices used in the `proof` (called the `route`) and a set of vertices that cannot be used in the `proof` (called the `forbid` set). This `forbid` set is grown incrementally. As the `proof` grows up from `outputs` towards `inputs`, the `forbids` set includes all the `vertices` incompatible with `literals` on the `proof`. `Proof's` also store the `output` they are a `proof` for, the `inputs` where the `proof` terminate[112], the `cost` of the `proof` (cost defined above), and the proof's `environment`.

```
proof    = record output: posint; inputs: posints;
                  cost: integer;
                  route,environment,forbid: bitstring end;
nproofs  = list of proof;
```

A `proof'`s `environment` is the most-upstream controversial assumptions encountered during a search from the `output` to the `inputs`. `Core` only stores controversial assumptions in a `proof` since the multiple-world generation will only iterate through the assumptions that crucially effect what can be believed (i.e. the controversial assumptions). If a `proof'`s `environment` is empty, then there are no pre-conditions on accepting a `proof`; i.e. it is acceptable in every possible world.

For proofs to build their environments sets, the controversial assumptions must be known prior to proof generation. A vertex is controversial if (i) it is relevant; and (ii) it's *relevant contradict set* overlaps with another relevant vertex. The *relevant contradict set* of a vertex is its `contradict` set, less `irrelevant` and `impossible` vertices computed in `mark_controversial_assumptions`.

```
procedure mark_controversial_assumptions
var    temp : set of literal; l1, l2: literal;
       relevant, irrelevant, impossible, avoid : bitstrings;
begin  relevant   ← "all v.ids that satisfy V[v].relevant";
       irrelevant ← "all v.ids that satisfy not V[v].relevant";
       impossible ← "all v.ids that satisfy V[v].impossible";
       avoid      ← bunion(irrelevant,impossible);
       for l1 ∈ bits(relevant) do begin
          temp ← bdifference(V[l1].contradicts, avoid);
          if not bempty(temp) then begin
            V[l1].controversial ← true;
            for l2 ∈  bits(temp) do V[l2].controversial ← true;
end end end;
```

`Core` attempts to build proofs for every item in the `Example.outputs` set.

[112]

```
function all_proofs : nproofs
var l: literal; out : nproofs;
  function some_proofs : proof
  var p:proof begin p.output ← l.v.id; return prove(l.v.id,0,p); end;
begin  for l ∈  Example.outputs  do begin
         out ← out ∪ some_proofs(l);
         if tooSlow then return out;
       end;
       return out
end;
```

Every time a `vertex v` is added to a growing `proof`, its `V[v].contradicts` set is added to `forbid`. Before new vertices are added to the proof, `Core` (i) checks that they do not belong to the `proof`'s `forbid` set; (ii) that the `cost` of the new proof does not exceed the maximum proof cost; and (iii) that the new vertex will not introduce a loop in the proof. If these tests are passed, and if the new vertex is an input vertex, then a successful proof has just occurred. On success, a copy of the temporary proof built up on the stack of recursive calls to the `proof` procedure is added to the output list of proofs. Since we do not assume that only `Model` roots are inputs, we must then recurse upwards looking for more proofs.

```
function prove(v: posint; delta_cost: integer; p: proof) : nproofs
var  out: nproofs;
  procedure add_v_to_proof
  begin  p.forbid ← bunion( p.forbid, V[v].contradicts);
         p.route ← bset!(p.route,v);
         p.cost ←  p.cost + delta_cost
  end;
  function looping : boolean return set?(p.route,v);
  function proved : proof
  var p1: proof
    begin p1 ← copy(p); p1.inputs ← {v}; return p1
  end;

begin  if not should_not_use(v) and not looping(p,v) then begin
         add_v_to_proof(p,v);
         if  not illegal(p) and not tooExpensive(p.cost) then  begin
           if V[v].input then out ← [proved(p)];
           if V[v].and   then out ← out ∪ prove_and(v,p,ps)
                         else out ← out ∪ prove_or(v,p, ps);
       end end;
       return out;
end;

function illegal(p: proof) : boolean
  -- A proof is illegal if it overlaps with its forbid set.
  return boverlap(p.route, p.forbid);
```

If a `proof` uses no *and* nodes, then the `environment` is simply the highest controversial assumption meet during the proof generation.

```
function prove_or(v: posint; p:proof): nproofs
var    e1: edge; p1: proof; ps, out : nproofs;
       successes : integer;
begin  if V[v].controversial then p.environment ← {v};
       for e1 ∈ Upstream(v) do begin
         trying({e1},1);
         ps  ←  prove(e1.to, e1.cost, p);
         successes ← successes  + size(ps);
         success({e1}, size(ps));
         out ← out ∪ps;
         if enough (successes) then return  out;
       end;
       return out
end;
```

Note the call within `prove_or` to `Upstream` to return a sequence of edges that the proof procedure should try next. `Upstream` can be used in non-exhaustive domains to optimise the search. The proof procedure will iterate over the output, left to right. Preferences in search strategies can be implemented by ordering or ordering/truncating the edges returned by `Upstream`. At the very least, `Upstream` should just return the `relevant_parents`. For example:

```
function Upstream(v :vertex): edges return relevant_parents(v);

function relevant_parents(v :vertex): edges
     --return all edges to parents p from v with p.relevant = true
```

`Enough` must be used with care. As `prove_or` iterates over the parents of an *or*-vertex, the proofs to date are kept in a temporary variable `out` and the number of successful proofs is kept in the `successes` counter. If `enough` signals a premature termination, then `out` is returned. The test for `Enough` must therefore be at the end of each processing loop, least it leaves some variables in some half-way state.

`Prove_or` is a simple function. The situation is more complicated for proofs that use an *and*-vertex since at the *and*, the proof procedure must ascend all parents. The environment in this case is the union of the proofs of the parent environments. More precisely, if an *and* node *X* has *i* parents and each parent has *j* proofs, then *X* has one possible proof for each member of the cross-product of the *j* proofs from the *i* parents. Note that each such member will be a list of proofs (one from each parent) and must be `combine`d. Further, if the combination is `illegal`, then that `combine`d proof should be rejected.

```
function prove_and(v: posint; p: proof) : nproofs
var    temp:list of nproofs; successes, i:integer; ups:edges;
       e1: edge; proofs,out : nproofs; p1: proof;
begin  ups ← Upstream(v); trying(ups,1);
       for e1∈ ups do begin
         proofs ← prove(e1.to, e1.cost, p);
         if  size(proofs) = 0 or tooSlow then return [];
         temp[inc(i)] ← proofs;
       end;
       for ps∈ cross_product(temp) do begin
         p1← combine(ps);
         if not illegal(p1) then begin
           successes ← successes+ 1; success(ups,1); out← out ∪{p1};
           if enough (successes) then return out end
       end;
       return out
end;

function cross_product(x : list of list) : list of list
  -- e.g. cross_product([[1,2]],[[fred]],[[a,b]]) =
  --   [[1,fred,a], [1,fred,b], [2,fred,a], [2,fred,b]].
  -- If any member of x is the empty set, then the
  --   cross-product is also an empty set.

function combine(ps : nproofs) : proof
  -- Return a proof that whose route, environment, forbid,
  -- inputs sets are a union of those sets in ps. The cost of a
  -- combined proof is the maximum of the cost of the costs in ps.

function inc(var i: integer): integer begin i← i+1; return i end;
```

`Prove_and` has much the same structure as `prove_or`. Temporaries keep track of proofs and the number of proofs generated to date. If `enough` signals early termination, then the temporary list of proofs is returned. If any parent of an *and*-vertex returns no proofs, then the *and*-vertex cannot generate a proof and we need not test any other parent. Hence, the return of an empty set if `size(proofs)=0.` One heuristic for culling the search space would be to customise `Upstream` to return the upstream edges of an *and*-vertex sorted by the ones that are most likely to fail first (determined from the `attempts`/`successes` counter).

4.3.7. Worlds

Any two generated `proof`s within `Proofs` can overlap (i.e. use the same literals). Further, some `proofs` can be mutually exclusive (i.e. they use `maybes` that are `contradicts` with literals used in other `proofs`). Hence, a proof can exist in multiple worlds.

The general `Core` process is the generation of all possible `proof`s of the above form. A `world` is a maximal subset of `proofs` (maximal with respect to number of containing `proofs`) such that none of the literals in its `proofs` contradict each other. The union of all the literals in all the proofs of `W[x]` is `W[x].context.` A `world` contains a subset of the `inputs` and `outputs` from `Example`. A world's `environment` is the union of the `environment`'s of its `proofs`. A world's `forbidden` set are all the literals that contradict the `context`.

```
  world    = record  environment,inputs,outputs,
                     context, forbidden: bitstring;
                     proofs : nproofs end;
  nworlds  = list of world;
```

The `label` of a literal is the list of all the worlds in which it can be proved. We could add pointers from each literal to each world. However, in the general case, this could be a large number of pointers. An alternative is to deduce labels as required via a function:

```
function label(l : literal, ws: nworlds) : nworlds
  -- return all worlds w ∈ ws for which  bset?(w.context,l.v.id)
```

Worlds are defined in terms of the world excluding assumptions set. This is calculated from the proof environments in the function `world_excluding_assumptions`. First, we compute a list of all literals used by each object in the proofs' environments (lines 16-21) . Then, we find all combinations of object-literals (via a cross-product) (line 25). Next, we find the *excluded* literals; i.e. the literals that each combination contradicts (lines 27-31). If a combination is consistent (i.e. does not overlap with its excluded set), then the forbidden literals for `combination[i]` has index `i` in the out list. Note that `out[1]` is reserved for the base controversial assumptions (see line 23 of the following code).

```
1.   function world_excluding_assumptions(ps: nproofs):
2.                                                  list of bitstring
3.   type l_lists: list of set of literal;
4.   var  baseControversialAssumptions: bitstring;
5.        integer;  combination : literals; l : literal;
6.        used,excluded: bitstring; out: list of bitstring;
7.   function object(i:integer):integer
8.       return V[i].l.p.object.id;
9.   function compress(in : l_lists) : l_lists
10.  var out : l_lists; v,i: integer;
11.    i ← 1; for v ∈ bits(baseControversialAssumptions) do
12.              out[inc(i)] ← in[object(v)];
13.    return out;
14.  end;
15.  function literal(i: integer) :integer return V[i].l;
16.  function used_literals_for_each_object : l_lists
17.  var out: l_lists; v: integer;
18.    for v ∈ bits(baseControversialAssumptions) do
19.      out[object(v)] ← out[object(v)] +  literal(v);
20.    return out
21.  end;
22.  begin  baseControversialAssumptions ← combine(ps).environment;
23.       out[1] ← baseControversialAssumptions;
24.       i ←  1;
25.       for combination ∈ cross_product(compress(
26.                              used_literals_for_each_object)) do
27.       begin  used ← excluded ← 0;
28.              for l ∈ combination do begin
29.                used     ← bset!(used,l.v.id);
30.                excluded ← bunion( excluded,V[v].contradicts);
31.              end;
32.              if not inconsistent(used,excluded)
33.              then out[inc(i)]← excluded;
34.              if timeOut then return out;
35.       end;
36.       return out
37.  end;

function inconsistent(b1,b2: bitstring): boolean
  return boverlap(b1,b2);
```

Low-level implementation detail: the list of used literals for each object will contain gaps at index positions of unused objects. To fill in the gaps, we have to copy the used index positions across to a temporary list with one entry for each used object (lines 9-14).

`Sort_into_worlds` implements the resolve of JUSTIN[113]. It contains two nested loops: one for each world excluding assumption and an inner loop for each proof. If a proof does not contradict a world's excluding assumptions, then it is added to that world. When a new world is created, it's `context` and explained `outputs` sets are initialised to the literals that we have explained via their non-connection to exogeny. After each world is generated, we can add some of the `ExplainedViaIsolation` literals. Such

[113]

literals could belong in every world since their explanation requires no assumptions. However, just to be safe, the procedure add_missables_to_world only adds the ExplainedViaIsolation literals that do not contradict the forbidden set of that world.

```
function sort_into_worlds(ps : nproofs) : nworlds;
var    p: proof;   w: world;
       exclusions : bitstring;
       ws: nworlds; i: integer;
begin  i ← 0;
       for exclusions ∈  world_excluding_assumptions(ps) do
       begin
           inc(i);
           ws[i] ← w ← new(world);
           for p ∈ ps do
           begin     if not inconsistent(ps.route,exclusions) then
                        add_proof_to_world(w,p);
                     if timeOut then  return  ws;
           end;
           add_missables_to_world(w);
       end;
       return ws
end;

procedure add_proof_to_world(var w : world; p : proof)
begin  w.environment ← bunion(w.environment,p.environment);
       w.outputs     ← bset!( w.outputs, p.output);
       w.inputs      ← bunion(w.inputs, p.inputs);
       w.forbidden   ←  bunion(w.forbidden,p.forbid);
       w.context     ← bunion(w.context, p.route);
end;

procedure add_missables_to_world(var w : world)
var    addable : bitstring;
begin  addable     ← bdifference(ExplainedViaIsolation,w.forbidden);
       w.context   ← bset!( w.context, addable);
       w.outputs   ← bset!( w.outputs, addable);
end;
```

4.3.8. Best(s)

Best returns the preferred worlds. Its general form is:

```
function best(ws : nworlds; example : behaviour): nworlds
```

Best is a domain-specific computation. We have previously described various best operators[114], some of which are specified below. Note that the customisation of the world selection process is just a few lines of code.

Best1 returns the worlds that use the least number of assumptions; i.e. the members of its context that are not known facts.

[114]

```
function best1 (ws : nworlds; example: behaviour) : nworlds
var  w: integer; x : bitstring; l : literal;
     score: list of posints;
begin   for w in 1 to size(ws) do begin
         x ← ws[w].context;
         for l ∈ example.facts do  bclear!(x, l.v.id);
         score[w] ← size(bits(x));
       end;
       return least_score(score,ws);
end;

function least_score(score : list of number; ws: nworlds) : nworlds
var  min : number; out: nworlds, w: integer;
  min ←  MaxInt;
  for w in 1 to size(ws) do  min ←minimum(min,score[w]);
  for w in 1 to size(ws) do  if score[w]=goal then out← out+ ws[w];
  return out;
end;
```

Best2 returns the worlds that use the fewest number of inputs.

```
function best2 (ws : nworlds; example: behaviour) : nworlds
var  w: integer; x : bitstring;
    score: list of posints;
begin  for w in 1 to size(ws) do score[w] ← size(bits(ws[w].inputs));
       return least_score(score, ws);
end;
```

Best3 returns the worlds which, on average, use the smallest number of proofs per thing explained.

```
function best3 (ws : nworlds; example: behaviour) : nworlds
var  w: integer;  score: list of real;
begin  for w in 1 to size(ws) do
         score[w] ← size(bits(ws[w].context))*100/
                     size(bits(ws[w].outputs));
       return least_score(score, ws);
end;
```

Best4 returns the worlds with maximum cover; i.e. explain the most number of effects.

```
function best4 (ws : nworlds; example: behaviour) : nworlds
var  w: integer;  score: list of real;
begin  for w in 1 to size(ws) do score[w] ← cover(w);
       return best_score(score, ws);
end;

function cover(w: world) : real
  return (size(bits(w.outputs)) * 100)/ Example.outputs;

function best_score(score : list of number; ws: nworlds) : nworlds
var  max : number; out: nworlds, w: integer;
  max ← -MaxInt;
  for w in 1 to size(ws) do  max ← maximum(max,score[w]);
  for w in 1 to size(ws) do  if score[w]=goal then out← out+ ws[w];
  return out;
end;
```

`Best6` returns the worlds with the smallest percentage base controversial assumptions; i.e. a world's `environment`.

```
function best6 (ws : nworlds; example: behaviour) : nworlds
var  w: integer;  score: list of real;
begin  for w in 1 to size(ws) do
         score[w] ←  size(bits(ws[w].environment))* 100/
                     size(bits(ws[w].context));
       return least_score(score, ws);
end;
```

4.4. Notes

In this section, we make some notes about *lumping* and *caching*, two unsuccessful attempts at optimising `Core`.

Cache was a simple idea inspired by the *memoing* of Warren's OLDT resolution [257]. Memoing is the processing of remembering solutions when they are generated. When problem solving, before looking for a new solution, the inference engine checks its knowledge of old solutions. Rather than repeat a prior computation, if an old solution exists, it is simply returned. Warren claims that memoing can reduces exponential time complexity to polynomial time.

Caching in `Core` is apparently simple. Before `prove` exits a vertex, it caches the proofs generated from that vertex. When `prove` arrives a vertex, if a cache exists for that vertex, then it returns the cache rather than repeating the computation.

Cache was tested by running `Core` with and without caching. A four-fold speed-up was noted in the execution time of a large model ($|V|$ = 554). However, 8% of the generated proofs were different suggesting that *Cache* had introduced some incompleteness into the inferencing. After some investigation, it was realised that the proofs found by `prove` at a vertex were dependant on the route taken from the output being explored to that vertex. Recall that proof generation is constrained by the `forbid` set built incrementally as a candidate proof grows. Proofs generated along different routes have different `forbid` sets. If `prove` arrives at a vertex from two different routes, then (potentially) different upstream vertices are usable by that proof (i.e. those that do not conflict with the `forbid` set).

To use memoing in `Core`, all possibly proofs from a vertex must be generated, unrestrained by proof-specific invariant violations; i.e. move the illegal test of `prove` into `proved`. If the proved proof was illegal (i.e. it's `route` overlapped with it's `forbid` set), then the proof is not added to the `out` set. This version of the program ran out of memory as every visited vertex cached every proof it could generate from itself up to any member of `inputs`. *Cache* was hence abandoned.

Lump was a small program that ran just after `mark_contraversial_assumptions`. If a relevant vertex had only one relevant parent, then they were both combined into one lump. All other vertices were assigned their own lump containing only themselves. Proof generation jumped lump to lump. Experiments with sample models showed that around 40% of vertices were lump-able. It was hoped that *Lump* would shorten the average proof length *L* and hence decrease the proof generation time.

Lump proved to be tricky that anticipated. In the case where an input or controversial assumption occurred within a lump and not at the root of a lump (i.e. it is *lump-internal*), lumping became rather convoluted. It was realised that lump generation has to be constrained to those lumps which do not violate the invariants. Note that this is a proof-dependant computation. Consider the difficult case where (i) an input *IN* is lump-internal of some lump *L ;* (ii) a proof *P* reaches *L;* (iii) some vertex *V* within *L* lump is inconsistent with the proof; and (iv) *V* is upstream of *IN* , i.e. the proof could terminate using *L* contents downstream from *IN*. If a proof treated all the vertices in a lump as one entity, then the proof would incorrectly ignore *IN* since when it accessed the members of the lump, it found an contradiction *V*.

The solution would be to allow proof generation to use lump portions separately. In the special case of non-cyclic theories, some partial ordering could be maintained of lump contents such that the inference engine could efficiently access portions of a lump upstream/downstream of some vertex. In the general case of possibly-cyclic theories, this is not possible.

Dividing a lump in a proof-specific manner negates the utility of lumping. *Lump* was hence abandoned.

4.5. Complexity (Again)

In this section, we use the pseudo-code to discuss the complexity of backtracking-less hypothesis testing without lumping or caching.

The above code can be summaries as follows. After a *forward sweep* pre-processor finds all the literals reachable from the inputs, a *backward sweep* executes from every output looking for members of the input. This search is restricted to the space found by the forward sweep. The backward sweep outputs a list of proofs and, as a side-effect, the minimal environments that describe the assumptions space. This knowledge is passed to a *worlds sweep* that assigns proofs to worlds via two nested for-loops: (i) an iteration through the minimal descriptions of the assumption space; and (ii) the proofs. These worlds are then filtered via the *best* operator.

We denote the total `Core` time complexity of the forward sweep, backward sweep, worlds sweep and best operators as $\varphi = \lambda+\phi+\gamma+\eta$ where λ,ϕ,γ and η denotes the time complexity of the best operator, forward sweep, backward sweep, and worlds sweep respectively.

The forwards sweep is implemented by `mark_relevant_verticies` and `visit`. `Visit` leaves a trial as it executes (the set relevant flags). If it arrives a previously visited vertex, it ignores its children. Worst case performance for `visit` arises in the case of a fully-connected graph of size N containing only *and*-vertices; i.e. each vertex has $N\text{-}1$ parents and $N\text{-}1$ children. N vertices will be visited. At each visit, `preconditions_satisfied` must test $N\text{-}1$ parents. `Visit` is then called recursively $N\text{-}1$ times on its children. This recursive call terminates if it arrives on a previously visited vertex. Total worst-case complexity for `mark_relevant_verticies` is therefore $\phi = N*N\text{-}1*N\text{-}1 = O(N^3)$. Usually, `Model` is sparse. Hence the average case performance is usually better than $O(N^3)$. It is not recommended trying to optimise `Core` via optimising `mark_relevant_verticies` since this is not the rate-determining step (the backward and worlds sweeps are much slower).

For the backward sweep, every pathway over a directed graph from a set of *start* vertices to a set of *end* vertices must be traced out. If these proofs: (i) are generated from X outputs; (ii) have an average length of L; (iii) and are generated over a dependency network with an average fanout of B; then there are $X*B^L$ such proofs. Therefore $\gamma = O(B^L)$.

For the worlds sweep, `Core` must iterate over all $X*B^L$ proofs for every member of the worlds excluding assumptions set. The number of worlds is equal to the number of consistent members of the world excluding assumptions. These are generated from all combinations of state assignments to the A objects references in the base controversial assumptions. If these base assumptions use, on average, S states from their A objects, then the upper limit on the number of worlds is S^A. Therefore $\eta = O(B^L * S^A)$.

An inspection of the specification of the `best` operators defined in the previous section shows λ is linearly proportional to the number of worlds. Therefore $\lambda = O(S^A)$.

Total complexity for `Core` $\varphi = \phi+\gamma+\eta$ is therefore a function of $<B,L,S,A,N>$. N and B are static parameters while A ,L, and S are runtime parameters that are dependant on model topology. Clearly, `Core` is impractical for models with long proofs (i.e. large L) and many base controversial assumptions for many objects (i.e. large A). The chapter *Practical* explores this limit experimentally and finds that the current implementation can handle $N \leq 800$ vertices (generated from 140 objects). Based on the available data on dependency networks of current expert systems, it will be argued this is adequate for the type of models we see in contemporary practice.

Heaven and earth were created all together in the same instant on October 23rd, 4004 B.C. at nine a.m. in the morning: Dr. John Lightfoot, vice-chancellor of Cambridge University, just before the publication of the Origin of the Species.

What really matters is the name you succeed in imposing on facts, not the facts themselves: Cohen's Law. *Every dream has a name and names tell your story:* David Byrne.

5. Customisable

In the previous chapter, we described the inner core of an hypothesis tester. In terms of building a usable system, `Core` is like the machine code of a computer. In this section, we describe the layer we added on top of `Core` to make it useful for modelling purposes. The claim that hypothesis testing is *customisable* is based on the observation that this added layer can be fully specified/ modified by editing a few small tables/ procedures.

The key to customisation is the identification of the modification points of some basic structure which apply to numerous domains. Our proposed basic structure is that of a *vague causal diagram* (VCD)[115]. VCDs can be processed using `Core` via a customisable *model compiler*[116] and *data compiler*[117]. We claim that the customisations required to reproduce Feldman & Compton's QMOD/JUSTIN work [79, 80] are general to parts-based ontologies[118]. We therefore give those customisations a special name: *QCM*, short for *Q*ualitative *C*ompartmental *M*odelling.

5.1. Vague Causal Diagrams

We view all symbolic knowledge bases[119] as *vague causal diagrams* (VCDs):

- We call them *vague* to emphasis their hypothetical[120] and possibly indeterminate/non-monotonic[121] nature.
- We call them *diagrams* since `Core` executes over a direct graph whose vertices contains literals from the expert's domain of discourse and whose edges represent the space of possible proof trees.

[115] See section 5.1.

[116] See section 5.2.

[117] See section 5.3.

[118] See section 5.4.

[119] Non-symbolic knowledge bases include neural networks and knowledge compiled into tables of mathematical probabilities.

[120] See chapter 1.

[121]

- We call them *causal* to emphasis their role in explaining known behaviour. Our general theme is that a knowledge base that cannot offer explanations of known behaviour is definitely invalid. Testing for validity is hence a process of generating possible explanations. Explanation and causality are intertwined.

Purists could argue that a purely propositional system or a system of equations has no causal reading since it contains statements of acausal logical implication or mathematical inter-relationships. This may be so, but a *usable* knowledge base also contains an inference engine that operationalises the knowledge. As far as an expert is concerned. a working expert system is the knowledge *plus* the inference. States that led to other states within the inference engine are definitely causal.   Our belief that all knowledge bases have a causal interpretation are supported by our informal observations of experts. When trying to understand the inner workings of a system, they often make causal connections between knowledge base entities[122].

Normally, VCDs are viewed as precursors to other modelling techniques which necessitates further knowledge acquisition. In quantitative fields, a VCD such as the Smythe '89 model[123] can be translated into a numeric compartmental model[124].   In equation-based qualitative fields, VCDs are generated by replacing continuous variables with the sign of their value or first-derivative, then deducing the dependency graph from the equations (e.g. causal ordering[125]).

Our approach is to explore what semantics can be granted to VCDs, without having to request more information from the expert(s) or the domain. That is, we would like a system that can *understand* (e.g.) hastily scribbled whiteboard sketches. We say we can a VCD is *understood* iff we can extract for it a deductive theory can explains some of our known behaviour without also entailing inconsistencies. We can also *understand* that VCD_x is better than VCD_y iff the deductive theory extracted from VCD_x can explain more known behaviours that the theory extracted from VCD_y. .

The imprecision of VCDs typically makes them indeterminate. VCD inferencing must assume multiple possibilities and manage mutually exclusive possibilities in separate worlds (i.e. the `Core` inference process). When generating possibilities, only a subset of the indeterminate VCD may be consistent; i.e. VCD inference is a search for subsets of the edges of the VCD which (i) are consistent and (ii) mention literals that we want to include in our reasoning (e.g. observations & effects).

[122] E.g. "Lets' see... that was retracted which lead to this being asserted which, once the conflict with the other thing was detected, blocked the path to the thing I wanted." or "That went up which is linearly proportional to this so this went up as well."

[123] See Figure 2.3 before section 2.2.2.

[124] See section 2.2.1.

[125]

Our claim is that the processing of VCDs is applicable to many knowledge bases. We support this claim with examples (see next section).

5.1.1. VCDs: Examples

Informal vague causal diagrams are a common technique for illustrating and sharing expert intuitions. Such diagrams consist of nodes connected by arcs labelled (e.g.) "inhibits", "+", "promotes", "-", or "blocks". Our neuroendocrinological expert could find five such graphs in as many minutes from the first two textbooks he took from bookshelf (e.g. figure 5.1).

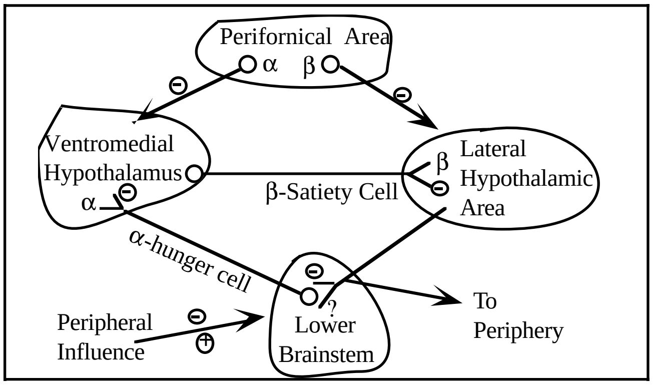


Figure 5.1. *Reciprocal circuits proposed for the roles of α- and β–adrenoceptors in the control of feeding of rats. The inhibitory and excusatory receptors are denoted by negative and positive signs respectively. From [10].*

Our expert expressed much of his physical intuitions regarding neuroendocrinology is such a graphical form (e.g. Figure 5.2).

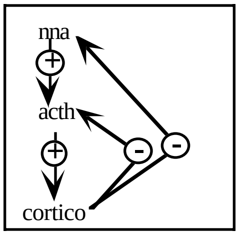


Figure 5.2: *VCD for connections between serum adrenocorticotropin (*acth*), serum corticosterone (*cortico*), and neuro-noradrenergic activity (*nna *- measured as the ratio of noradrenaline to its post-cursor, 3,4-dihydroxphenyl-ethethyleneglycol). VCD drawn by the author of* [239][126]

Once the eye is sensitised to VCDs, they can be spotted frequently. Spohrer & Riesbeck's models of economics, designed for language comprehension programs, are clearly VCDs (see Figure 5.3).

Clark & Matwin used a VCD to constrain the search space of a machine learning program learning the rules of economics (see Figure 5.4). Their term for a VCD was *RSpace*: the space of rules from which ideal domain rules can be learnt. *RSpace* represents…

> *…background knowledge (which) may be over-general (for the performance task), ambiguous, and contain more inconsistencies. The learning task is thus partly one of knowledge extraction (from the background knowledge). [41]*

[126]

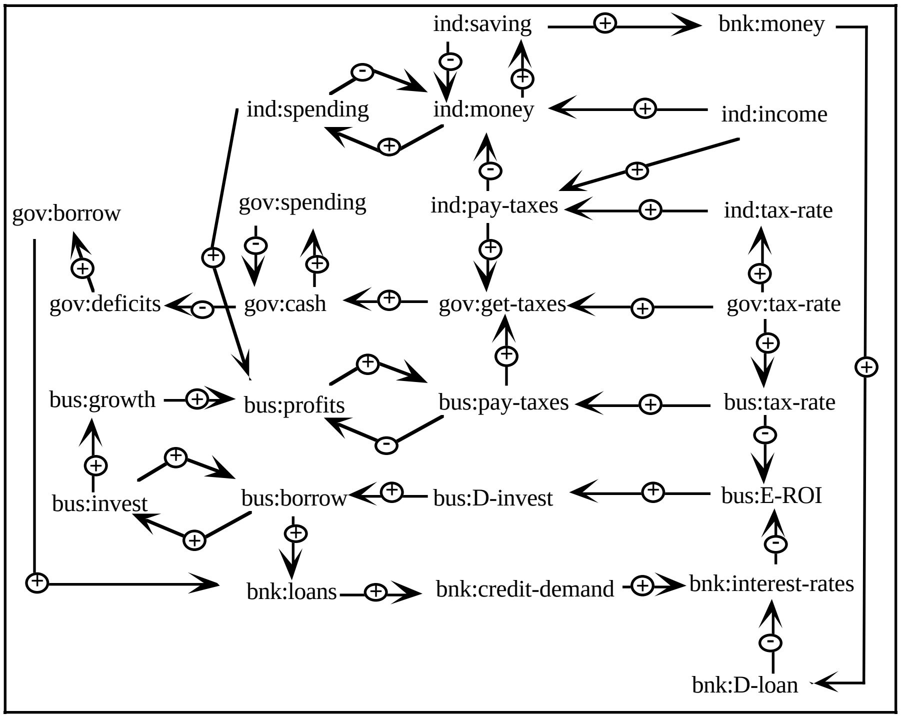


Figure 5.3 *Spohrer & Riesbeck's natural language model of the economy. From [244].*

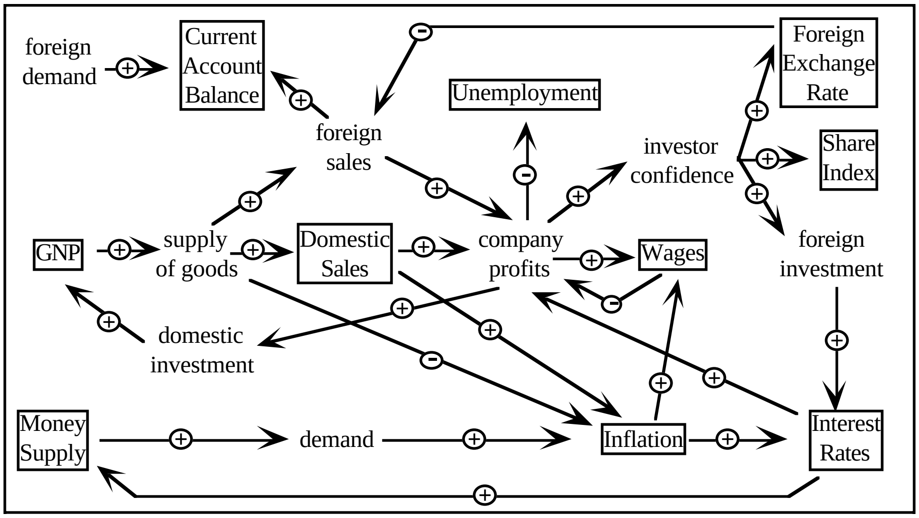
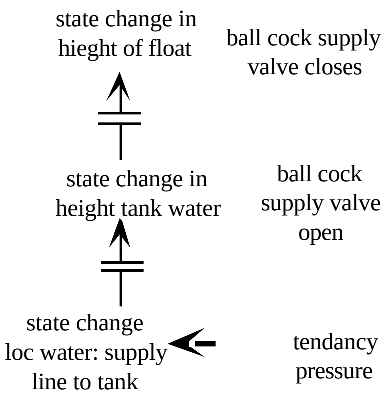


Figure 5.4: *Clark & Matwin's model of economics. From* [41]. *Note that their representation is based on Spopher & Riesbeck's*

The internal graphs of Reiger & Grinberg's MECHANISM LAB [220] are not-so-vague VCDs. MECHANISM LAB augments its causal diagrams with (i) quantitative information; (ii) conditional links; (iii) invariant knowledge (for example see figure 5.5).

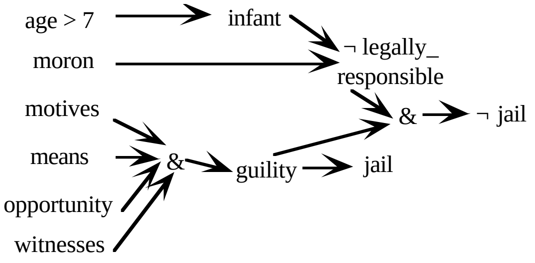
Reggia argues that a *problem-oriented attribute hierarchy* (read VCD) is a "general conceptual framework for representing information about domain-specific problems whose solutions are desired" [214]. Sample VCDs from Reggia are shown in figure 5.6.


state change in
hieght of float
ball cock supply
valve closes
state change in
height tank water
ball cock
supply valve
open
state change
loc water: supply
line to tank
tendancy
pressure

satisfied rules for actual firing. VCDs from propositional systems contain the dependency knowledge, but lack the conflict resolution knowledge. Hence, it may be non-monotonic. Note that Figure 5.7.ii can infer both *jail* and its negation. The processing of such a VCD must fork multiple worlds and apply an assessment operator to chose between the worlds (i.e. the `Core` process).

```
if      infant or moron
then    ¬ legally_responsible.

if      age > 7
then    infant.

if      ¬legally_responsible and
        guilty
then  ¬ jail.

if      motive and means and
        opportunity and witnesses
then    guilty.

if      guilty
then    jail.
```

(i)



(ii)

Figure 5.7: *Figure (ii) shows a VCD deduced from Figure (i).*

The VCDs listed explicitly represent all the vertices and edges in the model; i.e. they are *explicit*. Newell & Simon's problem space [126] are like an *implicit* VCD. In a problem space, an agent searches for a *goal state* from a *current state*. Each state has a set of operators which can transform the current state to a new state. New states can be generated at runtime. If we pre-compute and cache all possible states, and the and-or graph that connects them, the implicit search space of the problem space becomes an explicit VCD:

- States become a set of literals combined by an *and*-vertex.
- The operators have been compiled away into the and-or graph.
- The goal state becomes the only member of the `outputs` set.
- The initial current state becomes the `inputs` set.

Similarly, VCDs also exist implicitly in first-order theories. We can converting the implicit search space in a first-order theory into a propositional form with an explicit search space via *partial evaluation*. In partial evaluation, terms are *unfolded*; i.e. replaced with the body of their defining clauses. Variables bound during the unfolding can cascade across to other terms [107].

Lest we overstate our case, note that not all implicit search spaces can be represented explicitly. Infinite first-order theories[127] implies infinite unfolding in which case, the generation of ground instances will never terminate. Also, non-infinite theories/ implicit

[127] E.G. the space of all variables that are either 1, or twice X where X is some number already in the theory. In

problem-spaces that have a very large explicit representation that may be impractical to process.

VCDs can also represent equations. For example Iwasaki & Simon's causal ordering algorithm generates causal diagrams from equations [112, 114][128] is both vaguer and more precise than QMOD VCDs:

- *Vaguer:* Causal ordering as defined by Iwasaki and Simon does not augment its links with "encourages" or "promotes". However, this is a small extension to the basic algorithm.
- *More precise:* From the space of all possible dependencies between terms in equations, causal ordering selects a minimal subsets that is a simple tree. Hence, the route from exogeny to any variable is determinate. Exception: in the case where simultaneous equation solving is required, then there exists some indeterminacy in the order that the variables in the simultaneously solved equations were solved.

The algorithm was proposed as a general approach to causality in equational systems [238]. It was somewhat controversial and has been extensively debated (see [62, 112-114]). We make no comment here except to say that in the approach described below, the program that generates a VCD from a knowledge base (e.g. a set of equations) is a customisable *model-compiler* operator. The use of meta-knowledge of the inner workings of an equation solver to guide a VCD creation for equations is a reasonable heuristic. Causal ordering could hence be used as part of the front-end to `Core`. For other VCD-from-equation model-compilers, see [109, 131, 207].

5.1.2. Components of a VCD

Abstracting from the above examples of VCDs and the QMOD work, we say that a VCD is a directed, possibly cyclic, and-or graph whose edges represent the superset of explanations acceptable to the VCD author. A VCD's vertices are literals from some knowledge base. Certain sets of literals may be mutually exclusive. VCD processing is the extraction of a subset of the edges that are relevant to the task at hand that do not contain mutually exclusive literals.

Any general framework has limitation. For the sake of efficiency, `Core` makes certain assumptions about its VCDs:

- Literals are assumed to refer to objects that take one of finite, small, and pre-determined number of mutually exclusive values. That is, the VCD is explicitly represented before inference starts (i.e. not the implicit representation of Newell & Simon's problem space). The emphasis on a finite number of literals restricts

[128]

the processing of objects with continuous domains[129]. Hence, Core cannot process all the VCD examples described in the last section. For example, Core could not process the quantitative links of MECHANISM LAB.

- Knowledge of mutually-exclusive literals is restricted to pairs of literals. That is, the invariant rules have an arity of two.

Our own experimental investigation of the limits to our technique shows that our current implementation fails at $N > 850$ vertices and models with an average number of parents an average fanout > 7[130]. This places some limitation on the processing of domains that use time-based simulations. Our system must create one literal for each legal state of each attribute at every time tick with measurements. For example, if our model contains 40 attributes which can take one of 3 states, then we can process 850/(40*3) = 7 time ticks.

Within these limits, our experience has been that a customisable VCD processor can be specified by:

<*sources, objects, OVCs, literals, model-compiler*>

where *model-compiler* is a program specified by:

<*classes, invariants, local-rules, links*>.

The VCD can be tested via:

<*data-compiler,* Model, Best, Core>.

where Model is a VCD and *data-compiler* is a program specified by:

<*data, prep, ok2fail*>

Model, Core and Best were described in the previous chapter. In summary, Core extracted all consistent explanations relating to some example data from the space of possible explanations suggested by Model. This set was then filtered by Best to generate the preferred explanations.

As to the other components:

- A knowledge base is a set of assertions, each called a *source* statement. The propositions of a literal of a source statement is a OVC.
- An object-value-comparisons (OVC) is a test that at some time, an object's value passes some comparison test; e.g. The OVC *cortisol=up* uses the object *cortisol*, the comparison "=" and the value *up*.
- Each object must be one of pre-defined number of *classes*. Each class has a pre-defined domain.

[129] See perhaps not completely restricts; see section 5.2.5

[130]

- *Invariants* return true if two literals are incompatible. In parts-based ontologies, objects values of a certain class are incompatible with some other value from the domain of that object. However the invariants may also include statements about other objects.
- The *links* relation defines the valid connections between values in the domain of the known classes of objects. For example, the *direct* link models proportionality. If measurement M_1 is *directly linked* to measurement M_2, then M_2=*up* can connect to M_2=*up* and M_2=*down* can connect to M_1=*down*.
- A *data-compiler* that maps *data* into sets of vertices in the VCD. *Prep* assigns any required extra significance to special vertices (e.g. the vertices that are valid end-points for an explanation). *Prep* computes the `inputs`,`outputs`, `facts`, and `missables` sets passed to `Core` in the `Example` behaviour.
- A *model-compiler* that inputs source statements in the knowledge base and outputs it's associated VCD. This compilation feature can contain domain-specific processing described in the *local-rules*.

The output of the model-compiler is a VCD with two sets of components: the obvious vertices/edges and the tacit vertices/edges. Obviously, VCDs contain:

- One *or*-vertex for each literal in the source. If we restrict vertex generation to only the literals mentioned in the *source*, then the `Model` size is reduced. Each vertex also stores a list of contradictory vertices. This is compiled from the *invariants* knowledge.
- One edge for each dependency between vertices condoned by the knowledge base and approved by the definitions of legal inter-actions (i.e. the *links* relation). For example, if a source statement was *direct(cortisol, dexamethasone),* then the edges *{edge1(cortisol=up,dexamethasone=up), edge2(cortisol=down, dexamethasone=down)}* exist as shown in Figure 5.8.

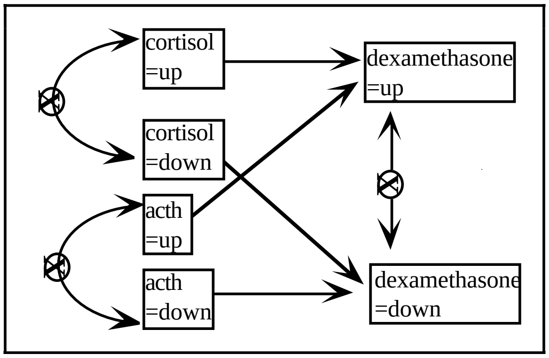


Figure 5.8: *Obvious VCD edges from the source statement* direct(cortisol, dexamethasone), *assuming that* cortisol *and* dexamethasone *can take the values* up *or* down. *As with MECHANISM LAB,* X ←⊗→ Y *denotes an incompatibility (i.e.* Y *is in* X*'s contradicts' set and visa versa)*[131].

Less obviously, VCDs also contain *tacit* components:

[131]

- *And*-vertices that represent combinations of literals which give rise to some net effect.
- Edges for certain causal connections that are tacit in the specification.

For more details on the tacit components, see the section *Model-Compiler*.

5.2. Model Compiler

An hypothesis tester *model-compiler* inputs the source statements and outputs its associated VCD `Model`. This VCD must include all the tacit components. Our current implementation caches the generated VCD. In the case of passing multiple examples to `Core`, using the cached VCD saves model-compilation time.

A customisable model compiler for parts-based ontologies can be specified via:

<*classes, invariants, local-rules, links* >

These terms are described below.

Classes was described above (see the `type` definitions for `class`[132]).

Invariants is a function that accepts two literals and returns true if they are incompatible. The `contradicts` set for each `Core literal` is computed by passing all pairs of literals to invariants. An invariant hard-wired into `Core` is attributes are single-valued; i.e. any value of an object contradicts any other value for that object.

Links is a table with five attributes that describes the legal links between objects:

```
links(name,class1,value1,class2, value2)
```

For example, our direct link could be specified as:

```
links(direct,measure,up,  measure,up).
links(direct,measure,down,measure,down).
```

Links can be used to control an interactive specification environment. Queries to the link table can provide control information for such an environment:

- User selection of vertices can be scoped by known object types (i.e. the unique members of the union of columns 2 and 4).
- A user may click on a graphical representation of *object1* and try to attach it to *object2*. The specification environment could constrain the set of the legal *object2*s to those object of a type that can connect to the type of *object1*.
- Source statements could be checked in batch mode. If the edges they propose violate *links* , then a warning message could be printed.

Local-rules are a set of domain-specific functions that add the non-obvious components:

- A VCD from an equational system could be generated using Iwasaki & Simon's causal ordering algorithm.

[132]

- Two commonly used non-obvious components in QCM are *ablers* and *steadies*. These are explained in the section *Local-Rules*[133].

The experience with the current implementation is that while writing *local-rules* may be a tedious task, the resulting rules are very short. For example, the QCM *local-rules* (described below) are 89 lines long of Smalltalk code. That is, *local-rules* are simply customised. The rest of this section of model compilers describes the local-rules for various domains.

5.2.1. Propositional Systems

In propositional domains, we would represent *modus ponens*[134] with edges *from* premise literals *to* consequence literals (with appropriate *and*-vertices added). We would then add additional edges *from* the negation of the conclusion *to* the negation of the premise (i.e. the *modus tollens*[135] edges).

When deducing a VCD from a rule-base, we caution against expressing conclusions that are negative literals as the negation of their pre-conditions. If applied recursively, such a re-expression is exactly the same process as unwinding a label to the roots of a dependency graph[136]. We noted in the previous chapter that such enthusiastic label generation can discard knowledge of incompatible proofs if the incompatibility lies in one of the literals expanded away by over-zealous labelling.

Note that if the VCD also contains the negation of the negative literal, then an invariant should be added that X and $\neg X$ cannot occur simultaneously.

5.2.2. First-Order Systems

For non-infinite first-order domains, we can partially evaluate the first-order theory to generate a ground instance of the same program. The resulting ground theory can be represented as a VCD.

Naively, we could create one object with domain *{yes,no}* for each ground term in the unfolded theory. However, we can be smarter. Given domain knowledge, a more economical representation may be possible. For example:

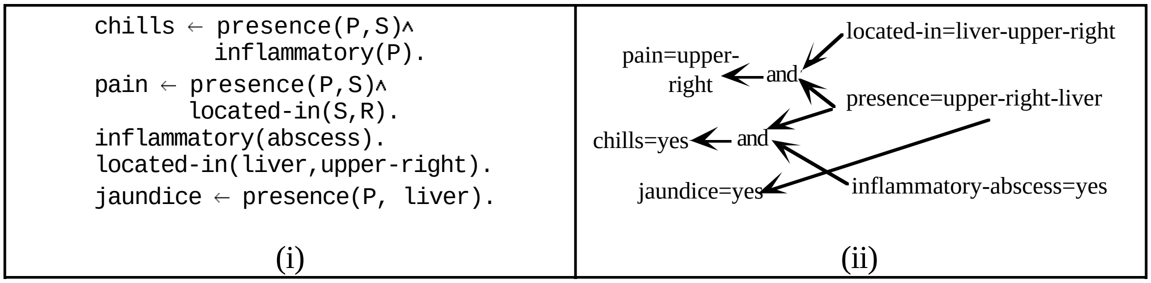
(i) If the ground theory contained the terms *{day(mon), day(tues) day(wed)}*, then it could be represented as a *day* object from the class *date* with the domain *{mon, tues, wed}*;

(ii) With meta-knowledge of terms, we could unfold Figure 5.9.i into Figure 5.9.ii.

[133] See sections 5.2.4.2 and 5.2.4.3.

[134] *Modus ponens*: {B, A if B} ⊢ A

[135] *Modus tollens*: {¬ A, A if B} ⊢ ¬ B



```
chills ← presence(P,S)^
          inflammatory(P).
pain ← presence(P,S)^
        located-in(S,R).
inflammatory(abscess).
located-in(liver,upper-right).
jaundice ← presence(P, liver).
```

(i)

Figure 5.9: *A smart unfolding of Figure (i) yields Figure (ii).*

5.2.3. Frame-Based Systems

To create a VCD from a frame system [16, 211, 213] we use Hayes' isomorphism between frames and logic [101]. Hayes' noted that much of the declaractive aspects of Minsky's frame representation [167] can be re-expressed as a set of batched assertions referring to one object. A VCD generated from a frame system would create one literal for every slot and frame, then edges *from* slots *to* frames and *from* subclasses *to* superclasses. Child frames would imply parent frames (e.g. duck → bird).

In domains where frames can only be activated by a full-match to slot contents, the slots would connect to an *and*-vertex. This *and*-vertex would then connect to (e.g.) *personFrame=active*. In domains where frames can be activated by a partial-match to slots contents, then the slots would connect directly to the *personFrame=active* vertex; i.e. evidence for any slot is evidence for its frame being active.

For example, frame-based reasoning, as explored by Reggia [213, 214] is a partial-match algorithm. Known inputs trigger the frames containing the slots that match the input. The matched frames are assessed according to various criteria. For example, Reggia's SYSTEM-D sought the minimum number of frames which could explain the known input. In the case of different explanations, Reggia used a heuristic 4-valued symbolic probability system was used to rank alternatives: <*a*> denotes a symptom that is always present; <*n*> denotes a symptom that is never present; and <*h*> or <*l*> denotes a symptom that is usually present or sometimes present respectively. A sample SYSTEM-D frame is shown in Figure 5.10.

```
dizziness-secondary-to-barbiturate-or-other-sedatives
     [description:
        dizziness [course  = chronic <h>, re$t <l>];
               current medication = barbiturates or other sedatives <a>;
                      diplopia <l>;
               nystagmus [occurrence = spontaneous;
                          type = horizontal <h>;  vertical <m>]].
```

Figure 5.10: *A sample* SYSTEM-D *frame* [212] *that lists various reasons for believing that a patient's dizziness is a result of the intake of certain drugs.* re$t <l> *denotes that all* non-chronic dizziness ' courses *are sometimes present for this frame.*

This frame uses data points defined in the attributes section of a System-D knowledge base (see Figure 5.11)

```
dizziness (val) [elaboration:
       type(sgl):    vertigo, syncopal, giddiness, imbalance;
       course (sgl):  acute and persistent,
              episodic [elaboration:episode duration(sgl):minutes,days, hours;
                        occurrence (sgl):orthostatic, positional,…,
                            only while urinating, non-specific];
              chronic]].

current-medications(mlt): …, l-dopa, insulin, barbiturates or other sedatives, … .

;NEURO-OTOLOGICAL SYMPTOMS
neursx (mlt):  diplopia  [elaboration: duration (sgl):
                            transient during dizziness, persistent],… .

;NEUROLOGICAL SYMPTOMS
neurex (mlt):  optic atrophy, …,
              nystagmus [elaborations: occurrence(mlt): positional, spontaneous;
                         type(mlt): horizontal, vertical, rotatory;
                         duration(sgl):transient during dizziness,persistent],
       ….
```

Figure 5.11: *Definition of* SYSTEM-D *attributes relevant to Figure 5.10. Attributes that can take multiple values are denoted* (mlt). *Attributes that can take only one of a set of values are denoted* (sgl). *... denotes details irrelevant to this example. Nested attributes are denoted with the keyword elaborations. For example, one observation made during a neurological examination could be* neurex.nystagmus.duration *is* persistent *or* transient during dizziness.

A model-compiler for frames could input Figure 5.10 and 5.11 to generate the and-or graph of Figure 5.12. Frames expressed in this form can be run using generalised test. In such a format, they offer a clean semantics for the SYSTEM-D symbolic probabilities. <*a*> and <*n*> are positive or negative literals that must be *and*-ed to explain the frame. <*h*> and <*l*> are used by *BEST* to assess different worlds. $BEST_{SYSTEM\text{-}D}$ would return the worlds with fewest "causes" (number of literals denoting frames i.e. has a value *active*) and with more <*h*> literals than <*l*> literals.

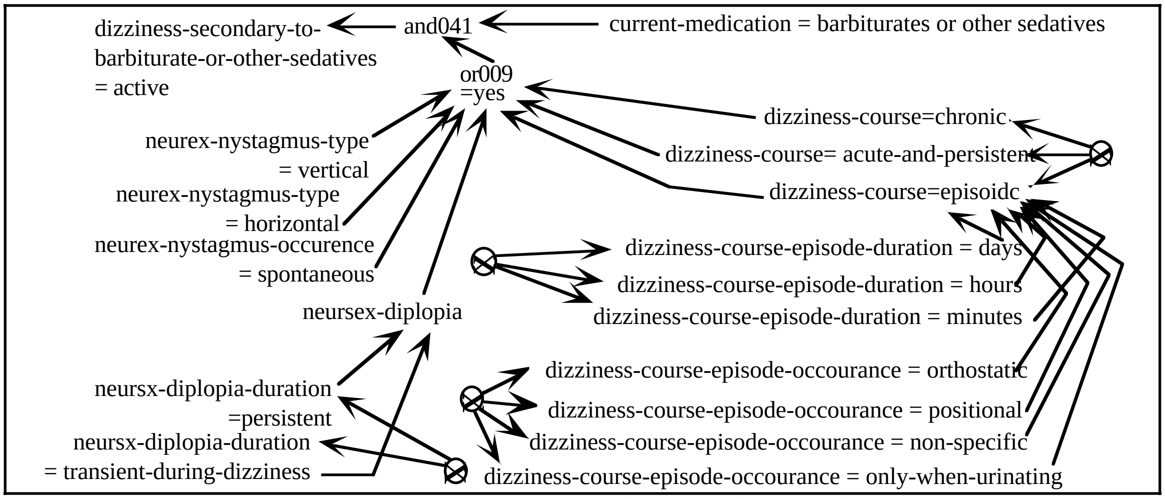

dizziness-secondary-to-
barbiturate-or-other-sedatives
= active
and041
current-medication = barbiturates or other sedatives
or009
=yes
dizziness-course=chronic
neurex-nystagmus-type
= vertical
dizziness-course= acute-and-persistent
neurex-nystagmus-type
= horizontal
dizziness-course=episoidc
neurex-nystagmus-occurence
= spontaneous
dizziness-course-episode-duration = days
dizziness-course-episode-duration = hours
dizziness-course-episode-duration = minutes
neursex-diplopia
dizziness-course-episode-occourance = orthostatic
neursx-diplopia-duration
=persistent
dizziness-course-episode-occourance = positional
neursx-diplopia-duration
= transient-during-dizziness
dizziness-course-episode-occourance = non-specific
dizziness-course-episode-occourance = only-when-urinating

Figure 5.12: *An and-or graph generated from the frame of Figure 5.10 and attributes of Figure 5.11.* Sgl *values are mutually exclusive; denoted by* ⊗ . *Nested attributes are evidence for the outer-level attribute; hence (e.g.)* neursx-diplopia-duration=persistent → neursex-diplopia. <a> *attributes denote essential slot contents; hence the* and041 *vertex. All other slots can be partially matched, hence the* or099 *vertex.*

An important variant on the frame-approach is the CLASSIC family of systems pioneered by KL-ONE system [16]. As these systems evolved, Brachman & Levesque came to realise that certain invariants they wished to apply to a particular frame required information that was non-local to that frame. As they generalised their system, they evolved a twin representation system. The sentence such as "all elephants are gray" defined its terms (e.g. elephant, gray) in terms of an inheritance system called the T-Box while the sentence its self was a global assertion in a first-order system called the A-Box [15].

HT4/`Core` would view A-Box/T-Box systems as input to a model compiler. T-Box statements would be compiled as above. A-Box statements would be partially evaluated to generate a propositional theory that used a set of literals found in the T-Box statements. One advantage of viewing A-Box/T-Box systems in terms of HT4/`Core` is that multiple-worlds reasoning comes for free.

5.2.4. Qualitative Modelling Systems

5.2.4.1. Obvious Components

Model compilation for the obvious components of a qualitative model can be controlled by a simple table look-up. Statements of the form *direct(a,b)* could be compiled into the graph $G = <V,E>$ where $V = \{a=up, a=down, b=up, d=down\}$ and $E = \{edge(a=up, b=up), edge(a=down, b=down)\}$.

More generally, we can say that given (i) a statement of the form $l_i(object_j, object_k)$); (ii) knowledge of the domain of each object (which can be deduced from the class of that object), and (iii) the links relation, which is of the form:

$$links(name,class_{1,} value_1,class_2, value_2)$$

then we can create one edge from $object_j=value_1$ to $object_k=value_2$ for each entry in the links relation where l_i equals *name* and $class_1$ is the class of $object_i$ and $class_2$ is the class of $object_2$.

Further, if we create two vertices from the same domain, then we can add each to the other's `contradicts` set; i.e. in a qualitative domain, variables can take only one value at the same time.

The less-obvious components are concerned with the conditional edges and the steady vertices (see below).

5.2.4.2. Conditional Edges

Suppose we have a knowledge base assertion that the link X,Y is conditional on some other literal Z. The MECHANISM LABS research arrived at a simple device for implementation such conditional edges: a new *and*-vertex is added between X and Y whose parents are X and Z. Explanations across this new *and*-vertex are conditional on the conjunction of X and Z.

around and short out the wiring. The link between flicking the light switch and increasing the illumination in a room is conditional on the absence of the rats; i.e. *if not rats then direct(power, lights)* . If *power* and *lights* are both measurements and *rats* is of the class *event* with domain *{present,absent,arrived, left},* then we have the dependency graph shown in Figure 5.13.

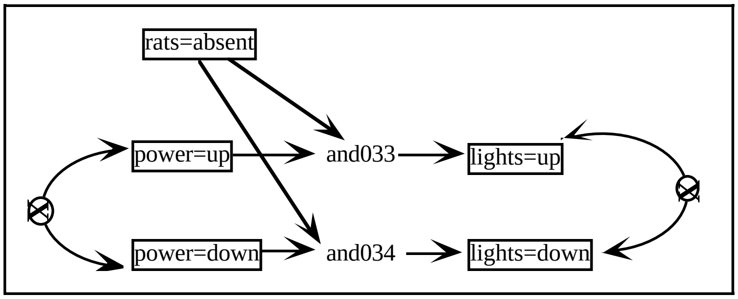


Figure 5.13. *Tacit edges from the source statement* if not rats then direct(power,lights). *This diagram is evolved further in Figure 5.14.*

Note that the domain of an *event* represents not only their current state*{present absent}* but also some comparison with a former state *{arrived, left)*. The reason for this is explained shortly.

Literals such as *rats=present* are called *ablers*. *rats=present* is a *disabler* since its presence disables some link. The opposite *enabler* literal enables some link if it is present. For example in the case of *if transportAvailable then direct(schools, literacy)*, *transportAvailable=present* is an enabler.

Ablers not only permit explanations in terms of other literals, but can be the roots of explanations. Recall our statement *if not rats then direct(power, lights)*. In the case of *power* not rising (but on) and the *rats* being *present*, the *lights* are dark. Now consider the same situation, but the *rats* suddenly disappearing. The *lights* going up can now be explained in terms of a change in the rat population. More generally, changes to an object's value downstream of an abler link can be explained in terms of changes to the abler. Mahidadia argues cautiously that any change in the downstream vertex can be explained in terms of any change to the abler [138][137]. We view this as an overly-generous approach and refine it below. However, it is consistent with our general goal of faulting models after making all possible consistent assumptions to generate the largest cover. Assuming for the moment that we adopt the Mahidadia rule, then *if not rats then direct(power,lights)* expands as per Figure 5.14.

[137] Recall that Mahidadia works on a parallel project. While we focus on faulting theories, Mahidadia explores correcting faulted neuroendocrinological theories in an inductive-logic programming paradigm [137, 140].

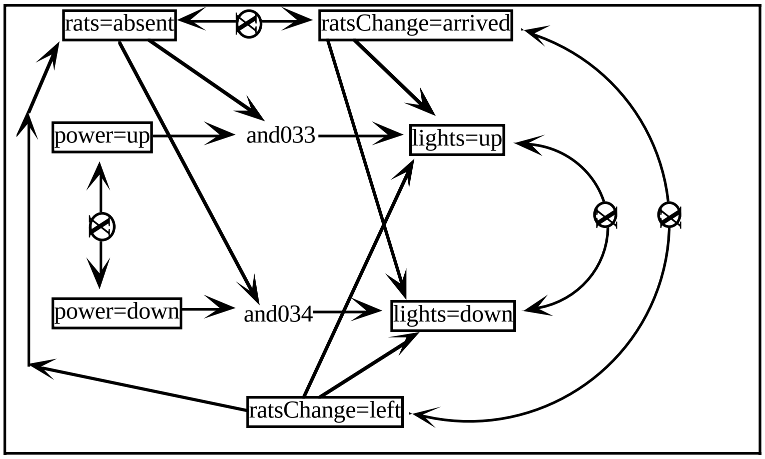


Figure 5.14. *Assuming Mahidadia's rule, we can add certain tacit edges and vertices to Figure 5.13. This diagram is evolved further in Figure 5.16.*

Note the creation of the *ratsChange* vertex. `Core` works best when no one object can be in two states at once. Hence, we cannot model the departure of the rats with the two vertices *rats=present* and *rats=left*. We therefore create a new object called *ratsChange*. Note that:

- *ratsChange* can be used to deduce the present/absent status of the rats; i.e the edge *ratsChange=left* to *rats=absent* shown in Figure 5.14 as well as the edge *ratsChange=arrived* to *rats=present*;
- *ratsChange* has certain incompatibilities with *rats* ; i.e. *rats=absent* ←⊗→ *ratsChange=arrived* and *rats=present* ←⊗→ *ratsChange=left*.
- *ratsChange=steady* is absent from Figure 5.14. A *steady ratsChange* is not a change and so it is not required by Mahidadia's rules.

5.2.4.3. Steady Vertices

Measurements of "no change" in an object (i.e. *steady*) can be explained in one of two ways:

- *Non-connection to exogeny:* If a steady vertex is not downstream from some perturbation to a model, then a plausible explanation for the steady is that nothing effected it.
- *Competing upstream influences:* In the case of connection to exogeny, if two parents of an object want to sent it both up and down, the net results could be a cancellation of the exogenous effect; i.e. *up + down = steady.*

For example, suppose we have *direct(cortisol, dexamethasone),* and *direct(acth dexamethasone)* (where *acth* is of class *measurement*). Two *and*-vertices exist above *dexamethasone=steady* connecting to the pairs of parents that can combine to produce a net steady effect on *dexamethasone*; i.e. *{cortisol=up, acth=down}* or *{cortisol=down, acth=up}*. These two *direct* links therefore give rise to Figure 5.15 .

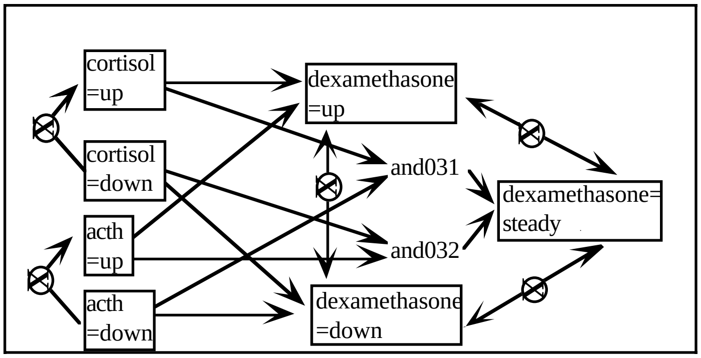


Figure 5.15: direct(cortisol, dexamethasone) *and* direct(acth dexamethasone) *implies that measurements of no changes in* dexamethasone *can be explained by a conjunction of opposing influences from its parents.* And031 *and* and032 *are specially-created* and-*vertices.*

The possibilities of steady explanations implies that we have not finished with our rats. Observe that the *lights* vertices of Figure 5.14 have more than one parent. Therefore, we can explain measurements of no changes in *lights* via combinations of competing upstream influences (see Figure 5.15).

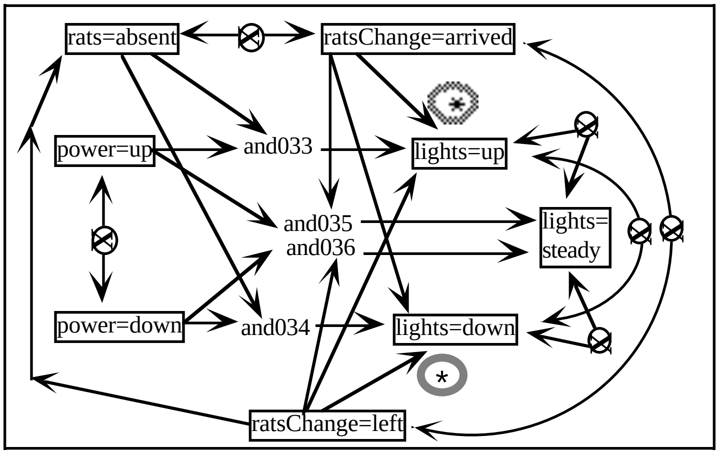


Figure 5.16: *All the tacit components of* if not rats then direct(power,lights). *Given the possibilities to steadies, we must add* and-*vertices to figure 5.14 .*

The edges marked with ⊛ *are arguably unnecessary (see below).*

Recall that Figure 5.16 was based on Mahidadia's argument that any change in the downstream vertex can be explained in terms of any change to the abler. Creating edges from all downstream vertex states to all abler states increases the number of behaviours the model can explain. If we cannot explain a behaviour given this generous assumption, then the model must indeed be faulty. While we cannot conclusively disprove Mahidadia ’s argument, we have never found an case in which the following two *restrictive edge-conditional expansion* rules did not suffice:

1) An enabler C influencing a link $\overline{AB}$ is linked $\overline{CB}$ in the same manner as $\overline{AB}$. For example, *if transport then direct(education, literacy)* implies the tacit link *direct(transport,literacy).*

2) Disablers are linked in the *opposite* manner. For example, direct links model qualitative proportionality. *Inverse* links model qualitative inverse proportionality. The model *inverse(a,b)* expands to E_1*(a=up,b=down)* and E_2*(a=down,b=up)*. The model *if not rats then direct(power,lights)* implies the tacit link *inverse(rats,lights)*.

The effect of these two rules is to reduce the causal connections between entities; i.e. according to these two rules, assuming restrictive edge-conditional expansion, then (i) Figure 5.16 has been over-generous in its causal assignments; and (ii) the edges marked with ⊛ are not required.

5.2.5. Continuous Systems

The above discussion tacitly assumes discrete domains. Objects representing numeric values are non-discrete and (in the general case) can take one of an infinite number of numeric values.

In order to process continuous domains in `Core`, we use a partial evaluation trick. We create one literal for each range mentioned in the source. For example, if (i) *age* is continuous; (ii) has the range 0 to 120; (iii) is referenced only in the source statement *if age > 16 then adult*; then we would create two literals for *age ≤ 16* and *age > 16*.

5.3. Data Compiler

Once the source statements are compiled into an and-or graph, the database of known observations must be mapped into that graph. This mapping process is accomplished by the data compiler. The data compiler is specified by:

<data, prep, ok2faill>

The *data-compiler* inputs the *data* to produce a set of example input and output pairs. *Prep* takes each example in turn, generates a `behaviour` record. In general `facts` = `inputs` ∪ `outputs` and `maybes` = `V` (i.e. the whole model may be searched).

Most data-compilers are simple. For example, for propositional domains, lists of input and output literals map directly into `behaviour` records. However, even propositional problems have their subtleties. If a domain requires default reasoning[138] then the knowledge engineer might define a *prep* that adds literals that are all root vertices (i.e. have no in edges) to `inputs` and `maybes`. If these roots are also in `facts`, then they represent definite observations about the world outside the model. If these roots are not in `facts`, then they represent literals we can assume, since we have no evidence to the contrary.

One place were the generality of our formalism has been tainted by domain-specific details is the `missables` field of the behaviour record. One method for explaining steadies is to prove their non-connection to `inputs`. Hence, within `Core`, steady vertices must be handled slightly differently (see the references to `ExplainedViaIsolation` in `Core`). `Missables` is an attempt to abstract away

[138] E.g. we can believe *not day=tuesday* if we have no evidence to the contrary without (i) having to specify the domain of *day*; and (ii) expanding the negation into a belief of its complement (e.g. *day=monday* or

from this domain-specific processing from the domain of qualitative reasoning. QCM's data-compiler sets `missables` to the union of the *id* of the steady *vertices*.

With the exception of `missables`, `Core` is shielded from the inner details of *data* by the *data-compiler*. Hence, we have no need to it specify its details here, except to say that *data-compiler* is dependant on the structure of *data* (i.e. they should be implemented as a pair). For an example *data-compiler*, see the definition of QCM (below).

5.4. QCM: An Example of Customisation

This section gives an example of a data and a model compiler for qualitative causal models (QCM). This definition uses the *restrictive edge-conditional expansion* rules[139].

Our definition of QCM has a different status to the rest of this report. QCM reflects our intuitions about qualitative modelling for QMOD-style domains. If the reader disagrees with any other portion of this research, we can offer a reasoned defence. However if the reader disagrees with the intuitions in this section, then our bottom-line defence is this: if some aspect of QCM does not reflect your domain intuitions, then change the customisation tables.

5.4.1. Model Compilation

Figure 5.17 will be our example model that we will pass to the QCM model-compiler.

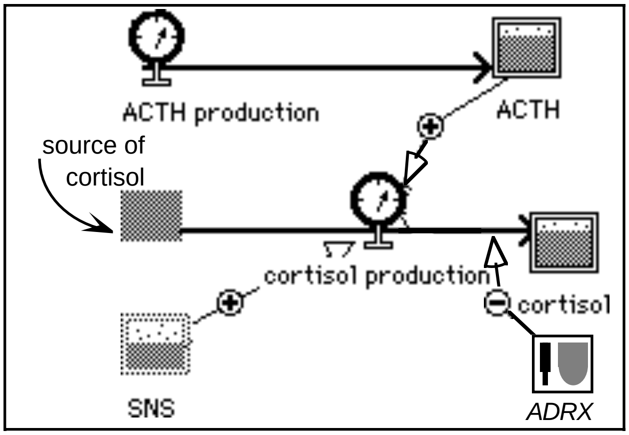


Figure 5.17 *A portion of the Smythe '89 model [240]*[140]*. The QCM model compiler inputs this diagram and outputs the edges of Figure 5.18.*

The *links* relation for QCM (see below) uses two classes, *events* and *measures*. *Events* model experimental interventions (such as surgical removal of the adrenal gland; e.g. *ADRX* from Figure 5.17). *Measures* model continuous variables that take one of three states: *up, down,* or *steady*.

Six edge types are known in QCM: *direct, inverse, creator*, *destroyers* , *inverse creators*, and *inverse destroyers* denoted ++, --, +-+, +-- , ---+ and --- respectively:

[139] Note that it would be a small change to use the Mahidadia expansion rules instead.

- *Direct* and *inverse* edges model qualitative proportionality and inverse proportionality respectively.
- *Creator* and *destroyer* edges model the flows of compartmental modelling. In-flows to a compartment can only increase the amount of stuff in the compartment while out-flows can only decrease the amount of stuff. Hence, while an increase in a in-flow/out-flow can raise/lower (respectively) the amount of stuff in a compartment, a decrease in a in-flow/out-flow does not effect the compartment. In Figure 5.17, *cortisol production* is a creator of *cortisol* and no destroyers are shown. Creators and destroyers are the only example we know of in the literature of asymmetric causal relations.
- The restrictive edge-conditional expansion rules require knowledge of the reverse of edge types. *Direct* and *inverse* are reverse. We define a *inverse creator* to be the "reverse" of a *creator* and a *inverse destroyer* to the "reverse" of a *destroyer*. By "reverse" we mean "hold the output influence of the edge constant whilst flipping the input influence". This will become clearer when we defined the QCM *links* relation.

The definitions of these edges is stored in the *links* relation[141]. As we explore the definition of *links*, note that we will use a lot of words to explain a very small relation; i.e. intricate domain semantics can be captured by a small/ customisable *links* relation.

A vertex of a certain class C_1 assigned a certain value V_1 can explain another vertex of another class C_2 having a certain value V_2 via some *Link* if (i) these two classes can effect each other; (ii) V_1 and V_2 are in the domain of C_1 and C_2; and (iii) that Link supports that explanation; i.e.:

```
links(Link,C1,V1,C2,V2) :- can_effect(C1,C2),
             domain(C1,V1), domain(C2,V2),
             links1(Link,C1/V1,C2/V2).
```

It is reasonable to assume that the physics that influenced one vertex could apply to another of the same class and that a change in one such vertex can influence the other.

```
can_effect(C,C).
```

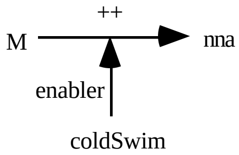
Should QCM permit propagation between vertices of different classes? Consider propagation between measures of different types (e.g. measures to events). Events are binary state devices and measures are continuos real-numbers. In the general case, influences between vertices of different class are nonsensical. Exception: domain-specific semantics may permit propagation between nodes of different types. For an example, QCM permits events to influence measures, but not visa versa; i.e.:

```
can_effect(event,measure).
```

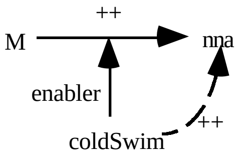
[141] For the sake of brevity, we express that relation here in Prolog (even though our current implementation is in

In the specific case of neuroendocrinology, we use domain-specific modelling semantics to argue that an *event-measure* link is shorthand for a somewhat arcane construct which can be simplified to event-measure links. Consider the modelling intention behind the *event-measure* link *direct(coldSwim ,nna).* We justify the validity of the event-measure link as follows.

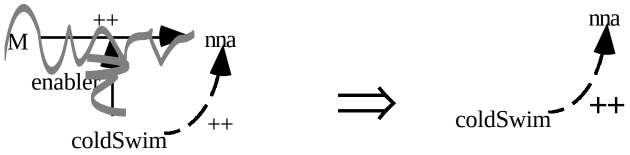
- *ColdSwim* is an event representing a two minute swim in ice water. *Nna* is a measure of brain stress. The model *direct(coldSwim,nna)* has the following modelling intent: "long swims in cold water increases levels of stress".
- If *coldSwim=absent,* then assuming all other influences being unchanged, the value of *nna* is unchanged.
- If *coldSwim=present* becomes true then it is as if some new mythical measure *M* has been suddenly connected to *nna* and the level of *nna* has increased.
- If *coldSwim=absent* then becomes false, then the connection between *nna* and this new measure *M* disappears.
- In terms of our ablers, it seems that the *coldSwim* enables a direct link between a mythical measure *M* and *nna*; i.e.:



- Applying restrictive edge-conditional expansion rules, we add the tacit edge *direct(coldSwim,nna)*:



- Note that we have no measurements for *M*; i.e. it will never appear in an `inputs` set. . Hence, no proof will ever terminate on *M*. Consequently, $\overline{Mnna}$ will never be used and so the *coldSwim* enable effect on $\overline{Mnna}$ will never be tested. These links are redundant and therefore can be removed.



- The final graph is a simple event-measure connection.

`Links1/3` describes the explanations supported by various edge types. Class-specific terms are classified as "positive" or "negative" and supported explanations are specified in terms of mappings between positive and negative terms.

```
    links1(++, X1,X2) :- pos(X1), pos(X2).  % direct
    links1(++, X1,X2) :- neg(X1), neg(X2).  % direct
    links1(-- ,X1,X2) :- pos(X1), neg(X2).  % inverse
    links1(-- ,X1,X2) :- neg(X1), pos(X2).  % inverse
    links1(+-+,X1,X2) :- pos(X1), pos(X2).  % creator
    links1(--+,X1,X2) :- neg(X1), pos(X2).  % inverse creator
    links1(+--,X1,X2) :- pos(X1), neg(X2).  % destroyer
    links1(---,X1,X2) :- neg(X1), neg(X2).  % inverse destroyer
```

The 2nd, 3rd, and 4th arguments of `class/4` define the "positive", "negative", or "neutral" value for various classes

```
    %class(name,  positive,           neutral,  negative).
    class(event,  [arrived, present], [],       [left, absent]).
    class(measure,[up],               [steady], [down]).
```

A value is "positive" or "negative" depending on where it is defined in its class.

```
    pos(C/V)     :- class(C,Positive, _,_), member(V,Positive).
    neg(C/V)     :- class(C,_,_, Negative), member(V,Negative).
    neutral(C/V) :- class(C,_,Neutral, _),  member(V,Neutral).
    domain(C,V)  :- pos(C/V) |
                    neg(C/V) |
                    neutral(C/V).
```

We can represent our example VCD[142] as the following assertions:

```
1) creator(acthProduction,acth).
2) direct(acth, cortisolProduction).
3) if not adrx then creator(cortisolProduction, cortisol).
4) direct(sns, cortisolProduction).
```

With our links knowledge, we can expand these assertions into a set of edges.

```
% edge(id, source, from, to).
edge(1,[1], acthProduction=up,       acth=up).
edge(2,[2], acth=up,                 cortisolProduction=up).
edge(3,[2], acth=down,               cortisolProduction=down).
edge(4,[3], cortisolProduction=up,   cortisol=up).
edge(5,[4], sns=up,                  cortisolProduction=up).
edge(6,[4], sns=down,                cortisolProduction=down).
```

Our local rules for steadies tells us that, in certain circumstances, we can explain a *cortisolProduction=steady*. These leads to the following edges.

```
edge(7, [2,4], sns=up,    and052).
edge(8, [2,4], acth=down, and052).
edge(9, [2,4], sns=down,  and053).
edge(10,[2,4], acth=up,   and053).
edge(11,[2,4], and052,    cortisolProduction=steady).
edge(12,[2,4], and053,    cortisolProduction=steady).
```

Further, our local rules for conditional edges tell us to insert into edge #4 an *and*-vertex to represent the disabler.

```
edge(4', [3], cortisolProduction=up,  and054).
edge(4'',[3], and054,                 cortisol=up).
edge(13, [3], adrx=absent,            and054).
```

[142]

Finally, the local rules also tell us that we can explain an increase in cortisol by the departure of the *ADRX* disabler.

```
edge(14, [3], change(adrx)=left, cortisol=up).
```

The final set of edges are shown in Figure 5.18.

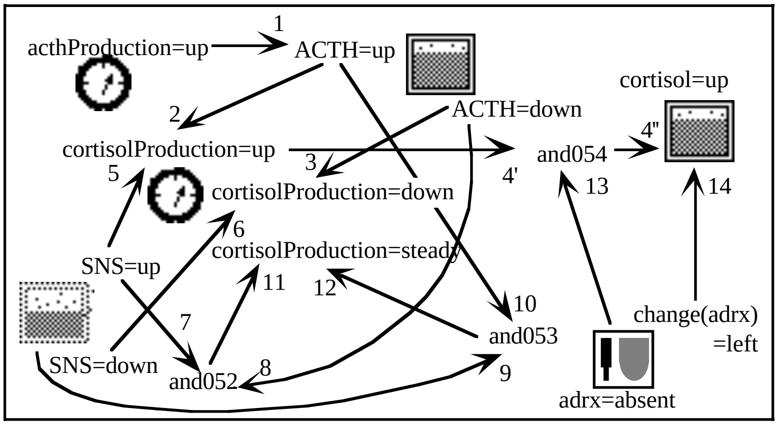


Figure 5.18: *The edges generated from Figure 5.17. The edge numbers refer to the edges described in the text.*

Note the absence in our edges of the vertex in Figure 5.18 marked "source of cortisol". Standard compartmental modelling demands that a flow must have an input and a output compartment. Pseudo-compartments called "sources" and "sinks" are therefore used to model infinite input and output streams. Such sources and sinks represent entities whose inner-details are not required for the current model. QCM could enforce that convention. However, recalling the above discussion regarding event-measure links, such sources and sinks would have the same status as the mythical *M* compartment; i.e. present in the model but never used in a proof. Therefore, QCM just ignores them and models of flows that connect to sources and sinks are terminated at the flow[143].

5.4.2. Data Compilation

Having creating a model, the data compiler must map known observations into the vertices of that model. We will use the example from [153] to illustrate the data compiler since (i) it is a small example; (ii) it is an example of a neuroendocrinological theory published in an internationally refereed journal [239] that can be faulted with `Core`[144].

Table 5.1 repeats some of the Symthe '87 results which we saw earlier[145].

			Context	
Value	**{} = control**	**{dex}**	**{coldSwim}**	**{dex , coldSwim}**
nna	0.122	0.105	0.210	0.246
serum cortico	129.0	11.3	1232.0	32.8
serum acth	89.0	0.0	240.0	0.0

Table 5.1: *Sample experimental data (from* [239]*).*

[143] For another example of the QCM model-compiler, see Figure 5.19, section 5.4.2 and Figure 7.8, section 7.3.

[144] See section 6.1.

[145]

The rows of Table 5.1 record the values of various measures taken during the different experiments. The column of this table represent different experiments (conjunctions of events). For example: (i) the *control* experiment is the case where *dex* and *coldSwim* where both absent; (ii) the *{dex, coldSwim}* experiment is the simultaneous events of a *coldSwim* and an injection of dexamethasone (*dex*). Note the closed-world assumption. If a event is not mentioned in a column header, it is assumed to be absent; e.g. the event *{dex}* is really *dex & ¬ coldSwim*.

The data of Table 5.1 refers to measurements of the entity being modelled in Figure 5.19.i.

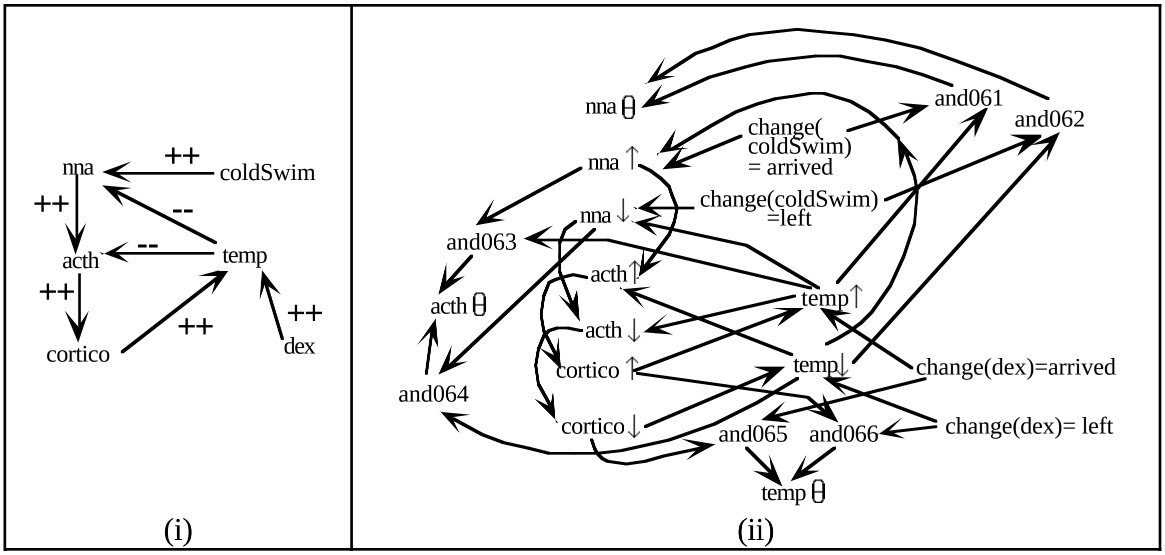


Figure 5.19. (i) *Connections between serum adrenocorticotropin (*acth*), serum corticosterone (*cortico*), and neuro-noradrenergic activity (*nna*). The* temp *vertex models the intuition that* dex *acts in the same manner as* cortico. *Drawn by the author of* [239]. *(ii) The edges generated by the QCM model compiler from Figure (i) assuming that* coldSwim *and* dex *are* events *while* nna, acth, cortico *and* temp *are* measures. *Isolated vertices (e.g.* cortico θ*) are not shown.*

The QCM data-compiler has to convert the values in Table 5.1 into values from the domains of the classes of the entities being modelled. Recall that *measures* have the domain *{up, down, steady}*, *events* have the domain *{present, absent}* and *eventsChange* have the domain *{arrived, left}*. For the purposes of explanation, we represent Table 5.1 as the following Prolog `data/1` fact.

```
data([experiments    - [   [],    [dex],  [coldSwim], [coldSwim,dex]],
                       %------------------------------------------------
      results/nna    - [   0.122,  0.105,     0.210,    0.246],
      results/cortico- [ 129,     11.3,    1232,       32.8],
      results/acth   - [  89,      0,       240,        0]]).
```

We define certain accessors to query `data/1`. `Event/1` finds all valid events (i.e. all the experimental interventions mentioned in the experiments definition line of `data/1`. For example, `event(dex)` is true.

```
event(X)       :- events(All), member(X,All).
events(Events) :- setof(Event, events1(Event), Events).
events1(E)     :- experiments(Exps), member(Exp,Exps), member(E,Exp).
experiments(X) :- data(Data), member(experiments - X,Data).
```

M/3 finds all the value X of a measurement M in an certain Experiment (e.g. m(nna,[dex],0.105)).

```
m(M,Experiment,X)          :- experiments(Exps), results(M,Resutls),
                               m1(Experiment,Exps,Results, X).
results(M,Results)         :- data(Data),
                              member(results/M - Results, Data).
m1(Exp,[Exp|_], [R|_], R) :- !.
m1(Exp,[_|Exps],[_|Rs],R) :- m1(Exp,Exps,Rs,R).
```

Low-level details taken care of, we can now turn to the top-level driver for the QCM data compiler. For all *valid experimental comparisons* <E_1, E_2> (defined below), QcmDataCompiler/6 computes the inputs, outputs, facts, missables, and maybes sets for the behaviour record. Inputs/outputs are a report of the comparative state of the events between E_1 and E_2. in the events/measures respectively. Facts is a set of all known observations; i.e. the union of inputs and outputs. QCM has no heuristics for culling the search space. Hence, the search space maybes is simply all the vertices from the model.

```
qcmDataCompiler(E1, E2, Inputs, Outputs, Facts, Maybes, Missables) :-
  twoExps(E1,E2),               % for two experiments E1 and  E2
  values(event,E1,E2,Inputs),   % inputs  = values for events in E1, E2
  values(measure,E1,E2,Outputs),% outputs = values for changes in E1,E2
  union(Inputs,Outputs,Facts),  % facts   = inputs union outputs
  allModelVertices(Maybes),     % maybes  = all vertices in the model
  allSteadyVertices(Missables). % missables = all steady vertices.
```

Union/3 is a set union predicate. AllModelVertices/1 is a hook into the model data structure which returns all the vertices of the model. AllSteadyVertices/1 returns all the X=steady vertices from the model. The other sub-goals of QcmDataCompiler/6 are described below.

Given N experiments, the QCM data compiler generates one behaviour record for all each valid experimental comparisons.

```
% get a valid experimental comparison
twoExps(E1,E2) :-
 experiments(N),
 member(E1,N),           % E1 is any experiment
 member(E2,N),           % E2 is any experiment
 (not(model(symmetrical)) -> not(E1=E2) | E1 @> E2).
```

Given N experiments, there exist N^2-N, such pairs (i.e. we ignore the trivial case of comparing two identical experiments). In the case where the comparison <E_1,E_2> is the same as <E_2,E_1>, then the comparisons are *fully symmetrical* and we can ignore half the possible pairs (e.g. process <E_1,E_2> and ignore <E_2,E_1>). A model is fully symmetrical when all its links are symmetrical. For example, consider the model *direct(coldSwim, nna)* and the data:

```
data([experiments    - [   [],    [coldSwim]],
      results/nna    - [   0 ,     1]]).
```

Recall that *direct(coldSwim,nna)* expands to two edges:

edge(1,coldSwim=arrived, nna=up).
edge(2,coldSwim=left,nna=down).

For the comparison $<E_1,E_2>$ = *<Control, [coldSwim]>*, then:

<Inputs, Outputs> = *<{coldSwimChange=arrived}, {nna=up}>*.

For the reverse comparison $<E_2,E_1>$:

<Inputs, Outputs> = *<{coldSwimChange=left}, {nna=down}>*.

Note that the explanation for $<E_1,E_2>$ *is*

{coldSwim=arrived, nna=up}

which is symmetric to the explanation for $<E_2,E_1>$*;* i.e.

{coldSwim=left, nna=down} .

That is, everything that is explicable for $<E_1,E_2>$ is explicable for $<E_2,E_1>$. This model is fully symmetrical and we can save CPU time by ignoring inverse comparisons (i.e. $<E_2,E_1>$).

Now consider the same data, but the asymmetric model *creator(coldSwim, nna)*. Recall that creator links can only increase the downstream value, never decrease it. *Creator(coldSwim,nna)* expands to one edge:

edge(1,coldSwimChange=arrived, nna=up)

For the $<E_1,E_2>$ and $<E_2,E_1>$ comparisons, the *<Inputs, Outputs>* sets are as above. However, while we can explain *nna=up* in $<E_1,E_2>$, we can't explain *nna=down* in $<E_2,E_1>$. This model is not fully symmetrical and we cannot save time by ignoring the inverse comparisons.

The knowledge engineer configuring the QCM data compiler must define the `model(symmetrical)` rule used within `twoExps/2`. A QCM model is asymmetrical if it contains creators, destroyers, inverse creators, or inverse destroyers. Ablers also make a model asymmetric since the permitted explanations when the abler permits the link can be very different to the permitted explanations when the abler forbids the link. For example, while the Smythe '87 model is symmetrical, the Smythe '89 model[146] is not.

For each valid experimental comparison $<E_1,E_2>$, the QCM data compiler compares the *events* and *measures* in E_1 and E_2 to deduce their values. The simplest case is the computation for *measures*. A measure is assigned the value `up`, `down`, or `steady` according to the change in their numeric value.

[146]

```
% get all values for all objects of a certain type
values(Type,E1,E2,Out)   :- bagof(X,E1^E2^value(Type,E1,E2,X), Out).
value(measure,E1,E2,X=V) :- m(X,E1,M1), m(X,E2,M2), delta(M1,M2,V).

delta(N , N , steady).
delta(N1, N2, up    ) :- N1 < N2.
delta(N1, N2, down  ) :- N1 > N2.
```

Strictly speaking, `delta/3` should compute `up/down/steady` for measures after applying a t-test to a significant number of measurements to compare the means. Here, we show the simplest delta. Note that it would be a simple matter to change the QCM data compiler to handle t-test delta comparisons.

Note that `value/4` for *measures* fails if it can't find a value for `X` in both E_1 and E_2. In poorly measured domains, such failures would be common.

Computing the value of the *change(event)* is analogous to computing the deltas for the measurements. If an event was in E_1 and isn't in E_2, then it has `left`. In the converse case, it has `arrived`.

```
value(event,  E1,E2, change(X)= arrived):- event(X), not(member(X,E1)),
                                           member(X,E2).
value(event,  E1,E2, change(X)= left)   :- event(X), member(X,E1),
                                           not(member(X,E2)).
```

We assign the `present/absent` values to events by checking for their presence in E_2.

```
value(event, _, E2, X=present) :- event(X), member(X,E2).
value(event, _, E2, X=absent)  :- event(X), not(member(X,E2)).
```

Running the compiler on Table 5.1 generates the following output:

```
?- qcmDataCompiler(E1, E2, Inputs, Outputs, Facts, Maybes, Missables).
...
E1 =     [dex]
E2 =     [coldSwim]
Inputs = [coldSwim=present, dex=absent,
          change(coldSwim)=arrived, change(dex)=left]
Outputs= [nna=up, cortico=up, acth=up]
Facts =  [coldSwim=present, dex=absent, change(coldSwim)=arrived,
          change(dex)=left, nna=up, cortico=up, acth=up]
Maybes = [acth=down, acth=steady, acth=up, coldSwim=absent,
          coldSwim=present, cortico=down, cortico=steady, cortico=up,
          dex=absent, dex=present, nna=down, nna=steady, nna=up,
          temp=down, temp=steady, temp=up, change(coldSwim)=arrived,
          change(coldSwim)=left, change(dex)=arrived, change(dex)=left]
Missables=[acth=steady, cortico=steady, nna=steady, temp=steady]
...
```

5.5. Conclusion

Generalised test is a customisable process. On top of the `Core` inference procedure, we can build domain-specific data and model compilers. Such compilers may reflect intricate domains semantics, whilst remaining quite small/customisable.

Flight by machines heavier than air is unpractical and insignificant, if not utterly impossible: Simon Newcomb, 18 months before the Wright brothers took off at Kitty Hawk

Not to know certain things is a great part of wisdom: Hugo Gratius. *We have many sayings, but no doings:* Anonymous *Certum quod factum (i.e. one is certain of only what one builds):* Giovanni Battista Vico 1668-1744. *Avagoyamug:* Generic Australian. *I come from the frenetic "build-a-model-then-rip-it-apart" persuasion within Artificial Intelligence, because it has been my experience that , no matter how clever one is, he never uncovers the real problems by gedankens-experiments. Rather, he thinks a while, builds a model, runs it, watches it fail, thinks some more, revises it, runs it again, and so on:* Chuck Reiger.

6. Practical

Previously, we have (i) motivated the need for testing, (ii) defined a generalised test engine for propositional domains, and (iii) described domain-specific customisation layers which allow us to use the test engine for models from different domains. In this chapter, we demonstrate the practicality of the system defined above. This demonstration be done in three stages:

- *The Smythe '87 study.* In this study, a paper from an internationally refereed neuroendocrinological journal [239] is faulted [153] using HT4, the prototype system that we reversed-engineered to generate the `Core` specification. This will demonstrate that `Core/HT4` can find faults that are invisible to other techniques.
- *The Smythe '89 study.* We describe our corrections to the Feldman & Compton study which increased the percentage of inexplicable results from 32% to 45%. This study demonstrates that `Core`/HT4 scales to medium-sized models.
- *The mutation study.* Hundreds of models were artificially generated using the Smythe '89 models as an initial reference point. These models were then analysed using HT4 to identify the limits to our process. These limits will be found to of the same order as the models developed by current knowledge engineering practice.

6.1. The Smythe '87 Study

6.1.1. Description

We have previously describe the data set[147] and model[148] used in the Smythe '87 study [153, 239]. The section is an annotated trace of the execution of our Smalltalk implementation of `Core` running over the Smythe '87 model/data. Recall that Smythe '87 had 4 experiments: *control= {}, {dex}, {coldSwim}, and {dex, coldSwim}.* Smythe '87 is fully-symmetric so we only need compare $(4^2-4)/2 = 6$ experiments.

6.1.2. Results

Our first comparison is $<E_1, E_2> = <\{\},\{dex\}>$.

```
1.   ======================================================
2.   name..............[014. Smythe 87]
3.   date..............[(Sep 20, 1994 14:50:48)

4.   ----| comparison 1 |--------------------------------
5.   experiments.........[Control to (dex)]
6.   inputs..............[((change(dex) up))]
7.   outputs.............[((acth down) (cortico down) (nna down))]

8.   Conclusion: cant do.[()]
9.   % cant do...........[0]

10.  PROOFS(S):
11.  (acth down).......[(change(dex) arrived) (temp up) (nna down)
                         (acth down)]
12.  (acth down).......[(change(dex) arrived) (temp up) (acth down)]
13.  (cortico down)....[(change(dex) arrived) (temp up) (nna down)
                         (acth down) (cortico down)]
14.  (cortico down)....[(change(dex) arrived) (temp up)
                         (acth down) (cortico down) ]
15.  (nna down).. ......[(change(dex) arrived) (temp up) (nna down) ]
```

Figure 6.1 *Trace of the comparison between* control *and* dex.

`Core` reports that all the outputs can be explained (i.e. the `cantdo` set is empty, see lines 9-10). It then offers (i) two explanations for the changes in `acth` (lines 11 and 12); (ii) two explanations for the `cortico` change (lines 13 and 14); and (iii) one explanation for the change in `nna` (line 15). The two `cortico=down` explanations are shown in Figure 6.2.

[147] See Table 5.1, section 5.4.2.

[148]

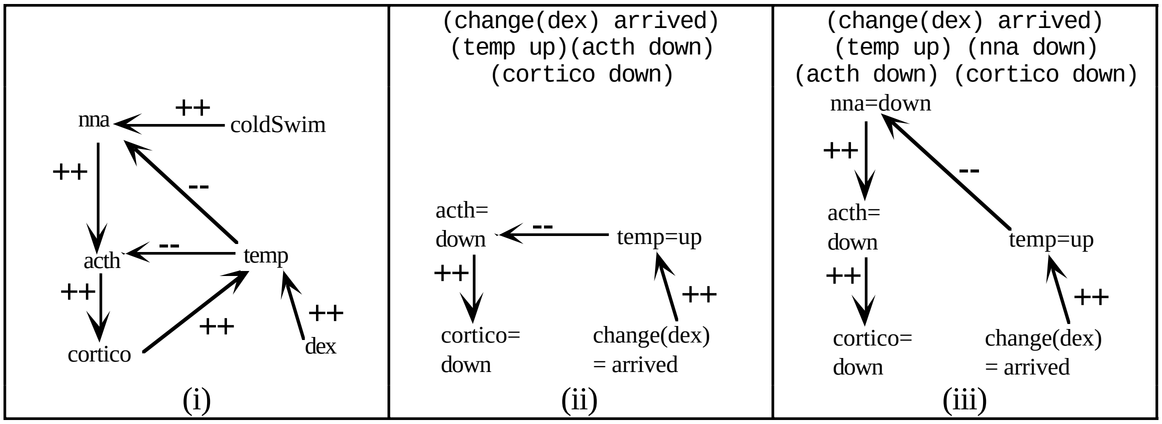


Figure 6.2. *Two proof from (ii) line 14 of Figure 6.1 and (iii) line 13 of Figure 6.1. Note that proofs are subsets of the edges in Figure (i).*

Like comparison 1, the next three comparisons can explain all the outputs in a single world.

```
16.  ----| comparison 2 |--------------------------------
17.  experiment........[Control to (coldSwim)]
18.  inputs............[((change(coldSwim) arrived))]
19.  outputs...........[((acth up) (cortico up) (nna up))]

20.  Conclusion: cant do.[()]
21.  % cant do.........[0]

22.  PROOFS(S):
23.  (acth up).........[(change(coldSwim) arrived) (nna up) (acth up) ]
24.  (cortico up). .....[(change(coldSwim) arrived) (nna up) (acth up)
                         (cortico up)   ]
25.  (nna up)..........[(change(coldSwim) arrived) (nna up)]
```

Figure 6.3. *Trace of the comparison between* control *and* coldSwim.

```
26.  ----| comparison 3 |--------------------------------
27.  experiments.......[Control to (dex coldSwim)]
28.  inputs............[((change(coldSwim) arrived)
                         (change(dex) arrived))]
29.  outputs...........[((acth down) (cortico down) (nna up))]

30.  Conclusion: cant do.[()]
31.  % cant do.........[0]

32.  PROOF(S):
33.  (acth down).......[(temp up) (acth down) (change(dex) arrived)]
34.  (cortico down)....[(change(dex) arrived) (temp up) (acth down)
                         (cortico down)   ]
35.  (nna up)..........[(change(coldSwim) arrived) (nna up)]
```

Figure 6.4. *Trace of comparison between* control *and* {dex, coldSwim}.

Comparison 4 has one interesting feature. Note the numerous proofs (lines 43-50 in Figure 6.5).

```
36.  ----| comparison 4 |--------------------------------
37.  experiments .......[(dex) to (coldSwim)]
38.  inputs ............[((change(coldSwim) arrived)
                          (change(dex) left))]
39.  outputs ...........[((acth up) (cortico up) (nna up))]

40.  Conclusion: cant do.[()]
41.  % cant do .........[0]

42.  PROOFS(S):
43.  (acth up) .........[(change(coldSwim) arrived) (nna up) (acth up)]
44.  (acth up) .........[(change(dex) left) (temp down) (acth up) ]
45.  (acth up) .........[(change(dex) left) (temp down) (nna up)
                         (acth up) ]
46.  (cortico up) ......[(change(dex) left) (temp down) (nna up)
                         (acth up) (cortico up)  ]
47.  (cortico up) ......[(change(coldSwim) arrived) (nna up) (acth up)
                         (cortico up) ]
48.  (cortico up) ......[(change(dex) left) (temp down) (acth up)
                         (cortico up)  ]
49.  (nna up) ..........[(change(dex) left) (temp down) (nna up) ]
50.  (nna up) ..........[(change(coldSwim) arrived) (nna up)]
```

Figure 6.5. *Trace of comparison between* dex *and* {coldSwim}.

Comparison 5 reports that two-thirds of the outputs are inexplicable (see line 56). Only `nna=up` can be explained.

```
51.  ----| comparison 5 |--------------------------------
52.  experiments .......[(dex) to (dex coldSwim)]
53.  inputs ............[((change(coldSwim) arrived))]
54.  outputs ...........[((acth steady) (cortico up) (nna up))]

55.  Conclusion: cant do.[((acth steady) (cortico up))]
56.  % cant do .........[67]

57.  PROOF(S):
58.  (nna up) ..........[(change(coldSwim) arrived) (nna up) ]
```

Figure 6.6. *Trace of comparison between* dex *and* {dex, coldSwim}.

The cause of the inexplicable outputs is two-fold. Firstly, the only input to `cortico` is from `acth` (see Figure 6.2.i). To explain `cortico=up`, we require `acth=up` but this is forbidden by the observation that `acth=steady` (see line 54, Figure 6.6). Secondly, since `acth` is downstream of the exogeny `{coldSwimChange = arrived}`, then we can only explain its steady value by a conjunction of competing upstream influences; i.e. either (i) `temp=up` & `nna=up` or (ii) `temp=down` & `nna=down`. Explanation (ii) is incompatible with the observation that `nna=up` (see line 54, Figure 6.6) so the only candidate is (i). However, to explain `temp=up`, we need some input from one of its upstream vertices. There are only two: `cortico=up`, which can't be explained for the above reasons ; and `change(dex)=arrived`, which is not possible since it is not found in the inputs (see line 53, Figure 6.6). Hence, we cannot explain `acth=steady` or `cortico=up`.

As an aside, we note that the addition of one extra link in Smythe '87 would permit explanations of all the outputs of *{dex} vs {dex coldSwim}* (see Figure 6.7).

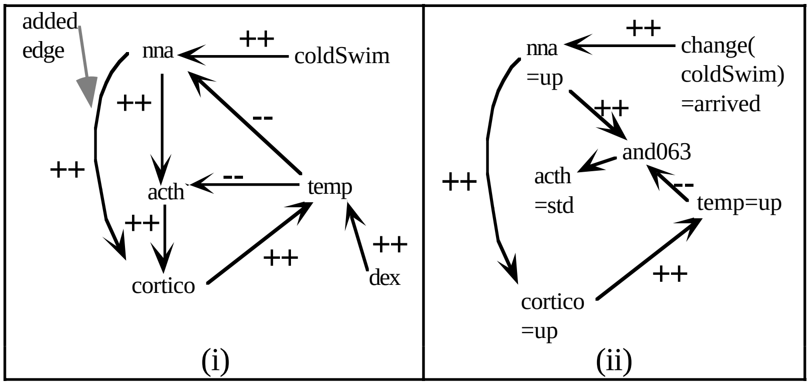


Figure 6.7: *The addition of the one edge shown in Figure (i) permits the explanations shown in Figure (ii) for* acth=std, nna=up *and* cortico=up.

Returning to the original Smythe '87 model, comparison six says that we can't explain `nna=up`.

```
59.  ----| comparison 6 |--------------------------------
60.  experiments.......[(coldSwim) to (dex coldSwim)]
61.  inputs............[((change(dex) arrived)
62.  outputs...........[((acth down) (cortico down) (nna up))]

63.  Conclusion: cant do.[((nna up))]
64.  % cant do.........[33]

65.  PROOF(S):
66.  (acth down).......[(change(dex) arrived) (temp up) (acth down) ]
67.  (cortico down)....[(change(dex) arrived) (temp up) (acth down)
                         (cortico down)]
```

Figure 6.8. *Trace of comparison between* coldSwim *and* {dex coldSwim}.

The reasons why *nna=up* failed are instructive. Possible proofs for *nna=up* are given in Figure 6.9

```
1. nna=up
2.   if (change(coldSwim)=arrived)
3.   or if (temp=down
4.                    if (change(dex)=left)
5.                    or if (cortico=down
6.                                 if (acth=down
7.                                       if (nna=down)
8.                                       or if temp=up )))
```

Figure 6.9. *Possible proofs for* nna=up[149].

Note that line 2 and line 4 of the proof fails since the required events are not in the inputs (see line 61, Figure 6.8). The alternative proof offered in lines 5 to 7 fails since `nna=down` (line 7) is incompatible with the outputs (line 62, Figure 6.8). The final alternative proof in line 8 fails since `temp=up` is incompatible with another assumptions along the proof; i.e. `temp=down`.

6.1.3. Discussion

The 4 experiments studied here from the Smythe '87 paper contained 16 measurements. 3 of these are inexplicable with respect to the model proposed by its author

[149]

(*{acth=steady, cortico=up}* in comparison 5; *{nna=up}* in comparison 6). Significantly, the author was not aware of these errors until we showed them to him [241]. Further, these errors escaped international peer review.

Even small models, such as Smythe '87, can generate large numbers of proofs (see lines 43-50). This observation leads to two comments:

- Without an automatic tool to handle the low-level mechanics of proof generation, human readers can find it tedious/unmanageable to explore all these proofs. This problem with the computational overhead associated with the simple technique is one of the reasons why generalised test can discover previously undetected errors.
- A pre-experimental intuition is that since a large number of behaviours are possible then, for complicated models, any behaviour at all could be reproduced. If this was the usual case, then generalised test would not be a useful tool since its critiquing power would be zero. This is the Pendrith objection to generalised test[150]. In the case of Smythe '87, we saw that this was not so in 3 of the 16 measurements. However, this one case does not give us confidence that, in the general case, the Pendrith objection does not hold. Consequently, we will check for the Pendrith limit in all our subsequent studies (see below).

6.1.4. Conclusion

For small models such as Smythe '87, generalised test can detect errors that are invisible to other review techniques such as visual inspection and international peer review. The Smythe '89 study (see below) checks if this result is true for larger models.

6.2. The Smythe '89 Study

6.2.1. Description

The Smythe '89 study was an attempt to repeat the results of Feldman & Compton. The Smythe '89 model of glucose regulation [240][151] and data sets used in QMOD/JUSTIN were converted into QCM and processed with HT4.

Smythe '89 is much larger than Smythe '87. It contained the events and measures of Figure 6.10.

[150] Named after the PhD student who first succinctly articulated this problem.

[151]

```
((e acutEdex)             (e adrx)                 (e chroniCdex)
 (e chroniCdiaz)          (e chroniCglucose)       (e chroniCinsulin)
 (e chroniCtolbut)        (e dex)                  (e diaz)
 (e etherstr)             (e gentle)               (e guan)
 (e hghInj)               (e hypox)                (e insulin10)
 (e insulin30)            (e insulinBolis)         (e msg)
 (e parg)                 (e ptu)                  (e stress)
 (e swimstr)              (e tolbut10)             (e tolbut20)
 (e tolbut30)             (e twoDg)                (e yoh)

 (m acth)
 (m acthProduction)       (m aluminium)            (m brainGlucose)
 (m brainGlucoseUptake)   (m catechole)            (m catecholeDisp)
 (m catecholeProd)        (m corticoidProduction)  (m cortisol)
 (m cortisolProduction)   (m crf)                  (m da)
 (m da2Hva)               (m daProduction)         (m dhpg)
 (m fiveHIAA)             (m fromGut)              (m fromLiver)
 (m fromPancreas)         (m ghProduction)         (m ghrh)
 (m glucagon)             (m glucagonDis)          (m glucagonProd)
 (m glucocorticoid)       (m glucose)              (m hgh)
 (m hva)                  (m insulin)              (m ne)
 (m ne2dhpg)              (m ne2Epin)              (m neControl)
 (m neProduction)         (m pHgh)                 (m pns)
 (m pPrl)                 (m prl)                  (m prlRelease)
 (m sateity)              (m serotonin)
 (m serotoninProduction)
 (m serotoninTOfiveHIAA)  (m sns)                  (m srif)
 (m t4)                   (m temp1)                (m temp2)
 (m temp3)                (m toKidneys)            (m toTissue)
 (m vagus))
```

Figure 6.10. *The 27 Events and 53 measures of QCM's Symthe '89.* (e X) *denotes that* X *is an event and* (m Y) *denotes that* Y *is a measure.*

QCM's Smythe '89 had the edges of Figure 6.11. These edges were generated by a manual translation of the QMOD diagrams into the QCM syntax. The QMOD diagrams were generated by the knowledge acquisition work of Feldman & Compton [79, 80] who interviewed Smythe. [240] to translate that loose causal diagram into the qualitative compartmental models of QMOD.

```
( crf  '++'  acthProduction )
( acthProduction  '+-+'  acth )
( hypox  '--'  acthProduction )
( acth  '++'  cortisolProduction )
( if guan then not
   (sns  '++'  cortisolProduction ))
( if adrx then not
  (cortisolProduction  '+-+'  cortisol ))
( corticoidProduction  '+--'  cortisol )
( glucocorticoid  '--'  acthProduction )
( corticoidProduction
   '+-+'  glucocorticoid )
( dex  '++'  glucocorticoid )
( acutEdex  '++'  glucocorticoid )
( chroniCdex  '++'  glucocorticoid )
( if adrx then not
   (catecholeProd  '+-+'  catechole ))
( catecholeDisp  '+--'  catechole )
( if guan then not(sns '++' catecholeProd))
( daProduction  '+-+'  da )
( da2Hva  '+--'  da )
( prl  '++'  da )
( aluminium  '--'  daProduction )
( if msg then not ( da2Hva  '+-+'  hva ))
( parg  '--'  da2Hva  )
( glucagonProd  '+-+'  glucagon  )
( glucagonDis  '+--'  glucagon  )
( if guan then not (sns '++' glucagonProd))
( glucose  '--'  glucagonProd )
( insulin  '--'  glucagonProd )
( chroniCglucose  '++'  glucose )
( fromGut  '+-+'  glucose )
( fromLiver  '+-+'  glucose )
( brainGlucoseUptake  '+--'  glucose )
( glucose  '++'  brainGlucoseUptake )
( toTissue  '+--'  glucose )
( brainGlucoseUptake  '+-+'  brainGlucose )
( temp1  '++'  toTissue )
( glucose  '++'  temp1 )
( insulin  '++'  temp1 )
( temp2  '++'  fromLiver )
( insulin  '--'  temp2 )
( glucocorticoid  '++'  temp2 )
( pns  '--'  temp2 )
( catechole  '++'  temp2 )
( if guan then not ( sns  '--'  temp2 ))
( glucagon  '++'  temp2 )
( twoDg  '--'  brainGlucoseUptake )
( fromPancreas  '+-+'  insulin )
( toKidneys  '+--'  insulin )
( insulin  '++'  toKidneys )
( if guan then ( sns  '--'  temp3 ))
( catechole  '--'  temp3 )
( glucagon  '++'  temp3 )
( glucose  '++'  temp3 )
( pns  '++'  temp3 )
( temp3  '++'  fromPancreas )
( insulinBolis  '++'  insulin )
( insulin10  '++'  insulin )
( insulin30  '++'  insulin )
( chroniCinsulin  '++'  insulin )
( tolbut10  '++'  fromPancreas )
( tolbut20  '++'  fromPancreas )
```

```
( tolbut30  '++'  fromPancreas )
( chroniCtolbut  '++'  fromPancreas )
( neProduction  '+--'  da )
( if msg then not (neProduction '+-+' ne))
( ne2dhpg  '+--'  ne )
( ne2Epin  '+--'  ne )
( if msg then not (ne2dhpg  '+-+'  dhpg ))
( dhpg  '++'  crf )
( dhpg  '++'  sns )
( stress  '++'  neControl )
( glucocorticoid  '--'  neControl )
( brainGlucose  '--'  neControl )
( neControl  '++'  neProduction )
( neControl  '++'  ne2dhpg )
( ne  '++'  ne2dhpg )
( aluminium  '--'  ne2dhpg )
( ne  '+-+'  ne2Epin )
( hgh  '++'  neProduction )
( insulin  '--'  neProduction )
( swimstr  '++'  stress )
( etherstr  '++'  stress )
( yoh  '++'  neProduction )
( parg  '--'  ne2dhpg )
( gentle  '++'  stress )
( diaz  '--'  neControl )
( chroniCdiaz  '--'  neControl )
( pns  '++'  vagus )
( insulin  '++'  pns )
( fiveHIAA  '++'  pns )
( sns  '--'  pns )
( da  '--'  prlRelease )
( da  '--'  pPrl )
( prlRelease  '+--'  pPrl )
( if hypox then not(prlRelease '+-+' prl))
( fiveHIAA  '++'  sateity )
( brainGlucose  '--'  sateity  )
( if msg then not (serotoninProduction
                          '+-+' serotonin ))
( serotoninTOfiveHIAA  '+--'  serotonin )
( serotoninTOfiveHIAA  '+-+'  fiveHIAA )
( hgh  '--'  serotoninProduction )
( t4  '--'  serotoninProduction )
( t4  '++'  serotoninTOfiveHIAA )
( serotonin  '++'  serotoninTOfiveHIAA )
( brainGlucose  '++'  serotoninProduction)
( insulin  '++'  serotoninProduction )
( pns  '++'  serotoninProduction )
( pns  '++'  serotoninTOfiveHIAA )
( parg  '--'  serotoninTOfiveHIAA )
( msg  '--'  serotoninProduction )
( pns  '--'  sns )
( ghProduction  '+--'  pHgh )
( if hypox then (ghProduction '+-+' hgh ))
( hghInj  '++'  hgh )
( fiveHIAA  '++'  ghrh )
( ghrh  '--'  pHgh )
( ghrh  '++'  ghProduction )
( glucose  '++'  ghProduction )
( glucose  '++'  pHgh )
( srif  '--'  pHgh )
( srif  '--'  ghProduction )
( crf  '++'  srif ))
```

Figure 6.11 *Edges of QCM's Smythe '89 model. See Table 6.1 for an explanation of symbols.*

Term	*Meaning*	*Term*	*Meaning*
X ++ Y	direct(X,Y)	X -- Y	inverse(X,Y)
X +-+ Y	creator(X,Y)	X +-- Y	destroyer (X,Y)
if X then Y	X enables Y	if X then not Y	X disables Y

Table 6.1. *Explanation of terms in Figure 6.11.*

Table 6.2 summaries the adjacency matrix formed by the edges shown in Figure 6.11. When calculating the adjacency matrix, we grouped together the 27 events and the 53 measures of Smythe '89. Hence the adjacency matrix had four quadrants: (i) from events to events; (ii) from measures to events; (iii) from events to measures; (iv) from measures to measures.

Quadrant	*From*	*To*	*Number of edges*	*% connectivity*
e-e	27 events	27 events	3	3/(27*27) = 0.41%
m-e	53 measures	27 events	0	0.00%
e-m	27 events	53 measures	23	23/(27*53) = 1.60%
m-m	53 measures	53 measures	94	94(/53*53) = 3.35%

Table 6.2: *Summary of the adjacency matrix formed by Figure 6.11.*

Note that no edges occurred in the measures-to-events (m-e) quadrant since the QCM semantics (defined in the *links* relation) forbids the connection of measures to events.

The edges in each quadrant where one of nine types (see Table 6.3).

		% edge type in each quadrant			
Edge name	*Example*	*e-e*	*m-e*	*e-m*	*m-m*
enabled creator	(if hypox then (ghProduction '+-+' hgh))	0	0	0	1
enabled inverse	(if guan then (sns '--' temp3))	0	0	0	1
disabled inverse	(if guan then not (sns '--' temp2))	0	0	0	1
disabled direct	(if guan then not (sns '++' catecholeProd))	0	0	0	3
disabled creator	(if adrx then not (catecholeProd '+-+' catechole))	0	0	0	7
creator	(corticoidProduction '+-+' glucocorticoid)	0	0	0	11
destroyer	(corticoidProduction '+--' cortisol)	0	0	0	15
inverse	(hypox '--' acthProduction)	0	0	35	22
direct	(crf '++' acthProduction)	100	0	65	39
	Total	100	0	100	100

Table 6.3: *Summary of the adjacency matrix formed by Figure 6.11. This table says, for example, that 39% of the edges in the* m-m *quadrant were of type* direct.

The model-compiler converted Figure 6.11 into a directed graph described in Table 6.4.

Source	*N=|V|*	*|E|*	*fan out = |E|/|V|*
Figure 6.11.	events: 27 measures: 53 total: 80	120	120/80 = 1.5
Model-compiler output from Figure 6.11	and-vertices: 294 other vertices: 260 total: 554	1246	1246/554 = 2.25

Table 6.4. *Comparison of the adjacency matrices of Figure 6.11 and the graph generated by the QCM model compiler from Figure 6.11. Recall that* and-vertices *are generated when domain-specific inferencing is compiled into an and-or graph. In this case, the* and-vertices *represent the* steady *semantics and conditional edge semantics discussed in the last chapter.*

The experimental data used for the Smythe '89 study came from the 1989 Feldman & Compton study (see Table 6.5). This data was taken from the papers used by Smythe to write his overview paper of glucose regulation [240].

	Measurement											
Experiment	*da*	*ne*	*hgh*	*hva*	*acth*	*dhpg*	*glucose*	*insulin*	*cortisol*	*fiveHIAA*	*glucagon*	*serotonin*
Control	10	10	10	10	10	10	10	10	10	10	10	10
msg	5	10		10		10				10		15
diaz	10	5		10	20	12			90	5		20
guan		7				30	5	5	50			
parg	20	20		2		2				2		20
hypox	10	10		10		10				20		10
twoDg	10	8		20		20	20	15	50	10	15	12
acutEdex		10			10	15			10			
gentle		15			8	10			8	10		10
chroniCdex		15			1	10			5			
swimstr	10	10		12	20	20			100	9		12
etherstr	15	8		12	20	23			100	10		12
ptu yoh		3			20	30			20	9		11
tolbut10	9	9		11		11	5	50	50	10	10	10
tolbut20	10	10		10		10	5	20	40	10	10	10
insulin10		11	50			9	5		8	20	10	10
insulin30		10	5			20	3		9		50	
msg parg	20	20		2		2				2		20
chroniCtolbut	10	10		10		10	7	10	10	15	10	10
chroniCglucose	7	10		10		10	12	10	10	7	10	10
chroniCinsulin	10	9		10		11	5	20	25	10		10
gentle yoh		5			30	15			30	9		11
guan twoDg		7				21	9	10	50			
ptu swimstr	10	9		18	20	15			90	18		12
ptu etherstr	10	10		20	20	23			90	18		12
diaz chroniCdiaz	10	10		10	10	10			45	5		20
hypox hghInj	10	10		10		10				10		10
acutEdex swimstr		10			45	20			50			
chroniCdex swimstr		10			1	21			6			
chroniCglucose chroniCtolbut	10	10		10		10	8	10	10	15	10	10

Table 6.5. *Experimental results used for Smythe '89. Taken from Feldman & Compton '89. Column 1 shows the events present for each experiment. Note that the Feldman & Compton study used a Prolog that only supported integers. Hence, they normalised and rounded their data. This is why all the values presented here are simple integers.*

Table 6.5 demonstrates the poorly-measured nature of neuroendocrinology. The 30 experiments listed here measure 12 variables (*da, ne, hgh,... serotonin*). However, note the empty cells of Table 6.5. Not all the variables were measured in all the experiments. Recall that for a variable to appear in the outputs set of a behaviour record generated by the QCM data-compiler, it must be measured in two experiments. While there exist 12 possible comparisons, the average number of outputs that are comparable for each comparison is only 5.19.

Figure 6.12 shows the distribution of the number of inputs and outputs that the QCM data-compiler generated from Table 6.5.

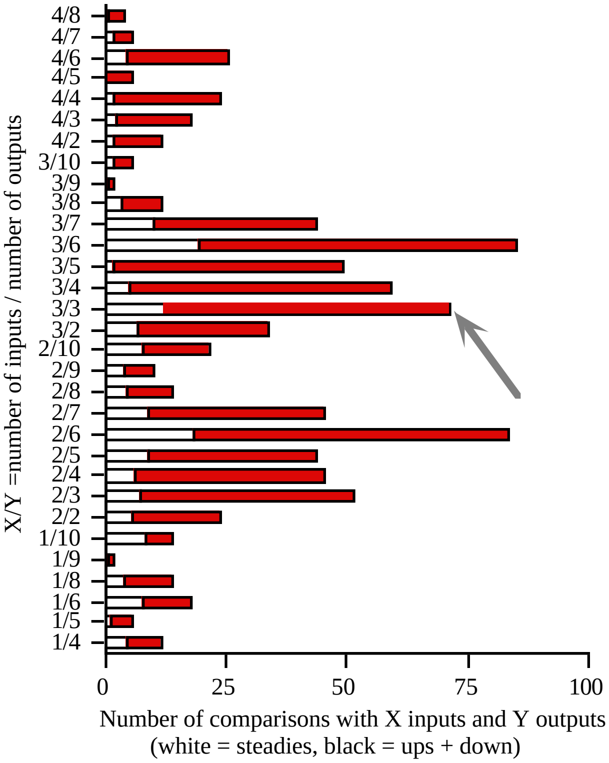


Figure 6.12. *Number of comparisons with* X *inputs and* Y *outputs from Table 6.5. Colours denote different types of outputs: white* = steadies *and black* = ups *and* downs. *For example, the line marked with an arrow says that 72 comparisons have 3 inputs and 3 outputs. Of those outputs, 20% were* steadies *(marked by the white region near the Y-axis) while the majority were* ups *or* downs *(numbers of ups and downs marked by the black region).*

6.2.2. Results

HT4 executed the 870 comparisons in 12343 seconds (≈ 206 minutes) on a Macintosh Powerbook 170 under System 7.0.1. HT4 was implemented in Smalltalk/V Mac version 1.0 running with 6MB of RAM.

On average, 45% of the outputs were found to be inexplicable (see Figure 6.13).

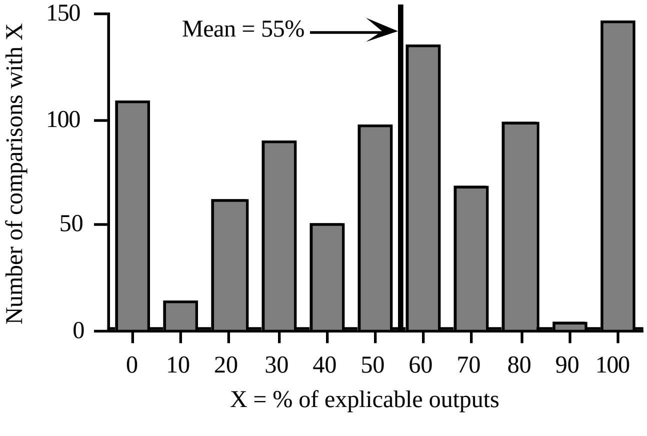


Figure 6.13: *Number of comparisons with X% inexplicable outputs. For example, 50 comparisons had 40% inexplicable outputs. The average number of explicable outputs was 55%. Note the right-most bar: only (150/870 = 17.2%) comparisons could explain 100% of their outputs.*

6.2.3. Discussion

Our average explicable rate of 55% is lower than the 68% reported by Feldman & Compton. We offer four explanations for the observed difference in behaviour between QMOD/JUSTIN and HT4:

- HT4 ran over a different set of comparisons than QMOD/JUSTIN. QMOD/JUSTIN had no semantics for steadies or multiple inputs. Hence, it ran only on the 24 comparisons with one input and no steadies in the output. HT4 could execute over all 870 comparisons.
- Feldman & Compton did not use creator and destroyer links to model flows in and out of compartments. Recall that creators and destroyers are asymmetrical causal connections. While they can explain a change in one direction, they are silent regarding changes in the opposite direction. Creators and destroyers are better representations of the in-flow and out-flow semantics of quantitative compartmental models. In-flows and out-flows were modelled as direct and inverse links in the QMOD/JUSTIN study. The introduction of these asymmetric causal edges reduced the number of possible explanations.
- Compton, while checking our models, found several errors in the representation of neuroendocrinological knowledge. HT4 used Compton's revised model.
- HT4 required the user to pre-specify all their vertex names prior to specifying the edges. Edges that terminated on an unknown vertex generated an error message. Using this simple error checking tool, several typographical errors were found in the QMOD/JUSTIN models. HT4 used the corrected models.

Precise runtime figures on Feldman & Compton's runtimes are not available, but Compton's believes that QMOD/JUSTIN took two days to run on a Macintosh SE (a machine four times slower than the Powerbook used for HT4). Comparatively, therefore, HT4 executed Smythe '89 two orders of magnitude faster than QMOD (see Figure 6.14).

Number of comparisons handled by HT4

QMOD runtimes (seconds)

$$\text{Speed-up} = \left(\frac{870}{24}\right) * \left(\frac{24 * 2 * 3600}{12343}\right) \Big/ \; 4 \quad = 126.8$$

Number of comparisons handled by QMOD

HT4 runtimes (seconds)

hardware factor

Figure 6.14. *Speed-up of HT4 over QMOD assuming a 2 day runtime for QMOD. Note: this figure of 2 days is only approximate.*

Given all the work we had done on optimising the testing process, we were disappointed by the mere two-orders of magnitude speed-up. The original goal of QMOD/JUSTIN was an interactive environment that could check theories as fast as experts could propose them. We now doubt the practicality of that goal: generalised-test seems more suitable to an over-night batch run than an interactive environment.

6.2.4. Conclusion

Smythe '87 demonstrated that utility of generalised test for small models. Smythe '89 demonstrated the utility of generalised test for medium-sized models. The level of critique is surprisingly high. Figure 6.13 tells us that only (150/870= 17.2%) comparisons could explain all their outputs. Further, on average, nearly half the data (100-55=45%) published to support Smythe '89 is not explainable with respect to that model. This is both a disturbing and exciting finding. It is disturbing in the sense that if the very first large-scale medical theory analysed by generalised-test contains significant numbers of errors, it raises doubts as to the accuracy of theories in general. This result is exciting in the sense that the level of critique is so high. Generalised test promises to be a powerful tool for assessing vague theories.

The question of scalability remains. Smythe '87 and Smythe '89 are interesting studies, but are too small a sample size to make general claims. The mutation study (see below) widens that sample size in an attempt to find the limits to generalised test.

6.3. The Mutation Study

6.3.1. Description

The mutation study began with the following observation. While model-compilers and data-compilers can vary from domain to domain, the structures they generate are remarkably uniform and simple. At its lowest level, generalised test executes over a directed graph between sets of vertices marked as inputs or outputs. Such graphs contain vertices V and edges E and can be partially characterised by their size ($N= |V|$) and average fanout ($B = |E|/|V|$)[152]. Generalised test is a process of finding explanations for O outputs in terms of I inputs ($I \subseteq V, O \subseteq V$). Recall that the and-or graph of Smythe '89 had $N = 554$ and $B = 2.25$[153] which was processed by $1 \le |I| \le 4$ and $1 \le |O| \le 10$[154].

The mutation study had four sub-studies, each of which held three of $<N,B,I,O>$ constant while varying the fourth. Table 6.6 shows the number of models and inferences made during the mutation study.

Study	*Models*	*# of runs*	*|V|*	*|E|/|V|*	*|Inputs|*	*|Outputs|*
Changing N	94	1991	150…1250	2…2.25	1…4	1…10
Changing B	161	2588	480…554	2…10	1…4	1…10
Changing 0	1	872	554	2.25	1…4	1…53
Changing I	1	519	554	2.25	1…27	1…10
Total	*257*	*5970*				

Table 6.6: *The mutation study called* `Core` *5970 times over 257 models of varying sizes* (|V|) *and fanout* (|E|/|V|). *Number of inputs and outputs were also varied.*

[152] This computation for average fanout also yields the average fan-in; i.e. average fan-out equals average fan-in. This must be so since average fanout and fan-in measure the average number of edges incident on a vertex. Since an edge must (i) come out of a vertex and (ii) go into a vertex, then the mean number of input edges (average fan-in) must equal the mean number of output edges (average fanout).

[153] See Table 6.4, section 6.2.1.

In all, 257 models were used for 6070 runs of HT4[155].

6.3.1.1. The Changing N Study.

The changing N study varied N whilst keeping $B,I,$ and O constant; i.e. $150 \leq N \leq 1250$, $B = 2.2.5$, $1 \leq |I| \leq 4$, $1 \leq |O| \leq 10$.

A set of L links of the form of Figure 6.11 were generated as follows. X number of vertices X_v were randomly created and set to be either events or measures in the ratio 27:53 events: measures (the ratio of events to measures in the Smythe '89 model[156]). Until L links had been created, the model-generator picked two members of X_v at random and assigned them an edge consistent with known distributions from the Smythe '89 model[157]. For example, no edges were assigned to measure-event pairs while 3.35% of measure-measure pairs were given an edge. The nature of the edge was controlled by with known distributions from the Smythe '89 model[158]. For example, all *event-event* edges were *direct* while 15% of the *measure-measure* links were *destroyers*. If a link was enabled or disabled, a third event vertex was chosen at random to act as the abler. These L links were then passed to the QCM model-compiler to generate an and-or graph.

Smythe '89 has $|L_0|$= 120 links. 94 models were generated using $|L|$ = 80, 100, 120, 140, 160, 180. This resulted in and-or graphs that varied in size from 150 to 1250 with a $2.0 \leq B \leq 2.25$. At runtime, between 1 and 4 events were chosen at random to be inputs and between 1 and 10 measures were chosen at random to be outputs. In this way, the 94 models were run 1991 times. Figure 6.15 shows the number of comparisons made at each and-or graph size.

When executed, it was found then that the N >1000 models did not terminate, even on very long runs. Memory errors or an exponential runtime curve were suspected. A "give-up" time of five minutes was built into HT4.

6.3.1.2. The Changing B Study

The changing B study varied B whilst keeping $N,I,$ and O constant; i.e. N=554, $2.25 \leq B \leq 10$, $1 \leq |I| \leq 4$, $1 \leq |O| \leq 10$.

The changing-B graphs were generated as follows. Starting with the and-or graph from Smythe '89, edges were added at random. After each addition, the average fanout of the and-or graph was checked against the desired fanout.

[155] Many of the sample sizes used in this study are not simple whole numbers. For example, 5895 is not 5000, 6000, or 10000. The results reported here were generated from thousands of runs of HT4, some of which crashed for random reasons. When we discarded the data from the crashed runs, we were left with the sample sizes described below.

[156] See Figure 6.10, section 6.2.1.

[157] See Table 6.2, section 6.2.1.

[158]

When executed, the runtimes were so low that it prompted further studies where (i) the fanout around and-vertices only was changed; (ii) the fanout around the or-vertices only was changed. At runtime, inputs and outputs were chosen as per the changing *N* study.

Preliminary results with the Smythe '89 model suggested that model criticability decreased with increased fanout. This effect seemed important enough to warrant further study. The *Changing B (other models) study* used the *Changing N study* model generator to produce six new models. These new models were then mutated between a fanout of $2 \leq B \leq 10$.

Table 6.7 shows the number of graphs and data sets used in the changing *B* study.

	Generated graphs	Number of runs
Basic changing B study.	76	1597
Changing B, ands-only	42	483
Changing B, ors-only	37	433
Changing B, other models	6	75
Totals	161	2588

Table 6.7: *Graphs and runs in the changing* B *study.*

6.3.1.3. The Changing I Study

The changing *I* study varied *I* whilst keeping *N,B,* and *O* constant at the Smythe '89 levels; i.e. N=554, B= 2.25 $1 \leq |I| \leq 27$, $1 \leq |O| \leq 10$.

To perform the changing *I* study, events were chosen at random from the Smythe '89 and-or graph to be members of the inputs set. The upper limit of *I*=27 came from the Smythe '89 model: it only had 27 events. Outputs were chosen as per the changing *N* study.

In all, 519 runs of increasing *I* size were executed across the Smythe '89 model.

6.3.1.4. The Changing O Study

The changing *O* study varied *O* whilst keeping *N,B,* and *I* constant at the Smythe '89 levels; i.e. N=554, B= 2.25 $1 \leq |I| \leq 4$, $1 \leq |O| \leq 53$.

To perform the changing *O* study, measures were chosen at random from the Smythe '89 and-or graph to be members of the outputs set. The upper limit of *O=53* came from the Smythe '89 model: it only had 53 measures. Inputs were chosen as per the changing *N* study.

In all, 872 runs of increasing *O* size were executed across the Smythe '89 model.

6.3.2. Results

6.3.2.1. The Changing N Study.

Figure 6.15 shows the number of comparisons in the *Changing N study*. As with all results from the changing *N* study, the results are averaged in "buckets" of size $|V| = 50$.

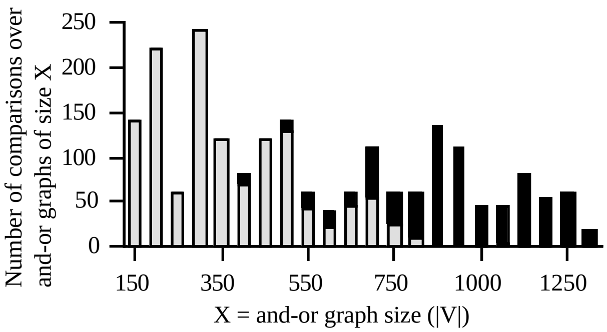


Figure 6.15. *Number of comparisons run on and-or graphs of different sizes (|V|) in the changing N study. Black denotes runs that did not terminate before the five minute give-up time. For example: 36 comparisons were run over and-or graphs of size 600.; and half of these failed to terminate. Note that none of the runs terminated after |V| =850.*

Figure 6.16 shows how the runtimes varied with model size. The plateau after |*V*|=800 was due to the give-up time of five minutes (300 seconds).

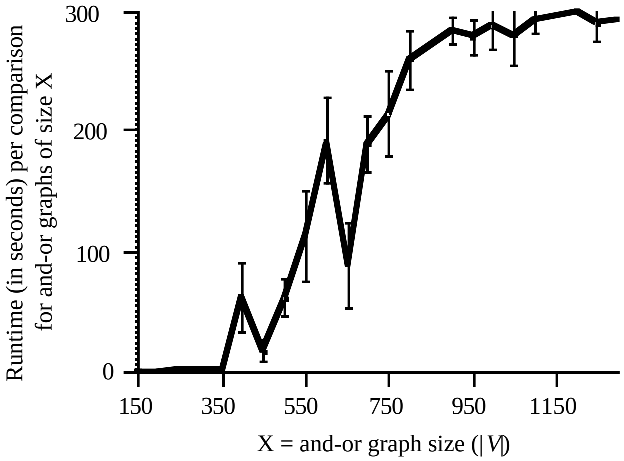


Figure 6.16: *Runtimes vs model size for the changing N study. The displayed runtimes are the means of all the runtimes divided into buckets of size |V|=50. Runtimes that exceeded the give-up time of 300 seconds were assigned a runtime of 300. Error bars denote the standard deviation σ of that mean, normalised by the sample size of each bucket; (sample sizes for each bucket shown in Figure 6.15) i.e.*

$$\mathrm{I} = \frac{\pm 2\sigma}{\sqrt{n-1}}$$

Figure 6.16 is inconclusive regarding the shape of the runtime curve. Any number of curves (e.g. linear, exponential, low-order polynomial) could be fitted to it. We tried to prove that the curve was a manageable polynomial (i.e. O(N^3) or less) with the following experiment. Models with 800 ≤ N ≤ 950 were tested with a give-up time of 14 minutes. At $N \geq 850$, the majority of the runs did not terminate; i.e. the average runtimes above N = 850 is greater than 14 minutes. Figure 6.17 contrasts the observed HT4 average runtimes with those of a cubic (*O(N^3)*), quartic (*O(N^4)*), and exponential curve.

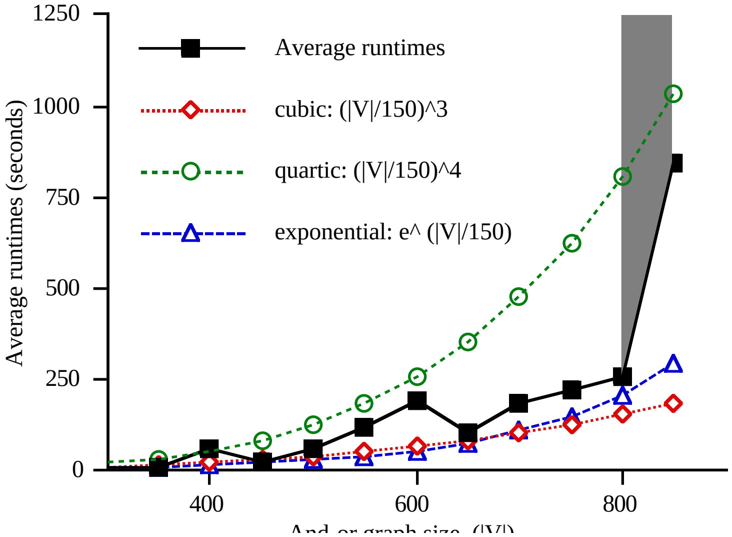


Figure 6.17: *Is HT4* O(N^3), O(N^4), *or* O(e^N)*? The curve marked with the black squares denotes the observed runtimes. Other curve denote cubic, quartic, or exponential curves. At* N *= 850, all the runs did not terminate; i.e. the runtime curve grows somewhere into the gray region shown on the right.*

Figure 6.17 suggests that HT4 has a runtime complexity that is worse than cubic on model size. Our intuition is that most curves fitted through the observed runtimes plot and the gray area of Figure 6.17 would be exponential. Further, we suspect HT4 must be exponential on model size due to the exhaustive nature of its inferencing. In subsequent chapters, we will develop a theoretical justification for believing that HT4/`Core` is exponential[159].

Figure 6.18 explores the Pendrith limit for the *Changing N study*. Recall that if the usual number of explainable outputs was 100%, then the indeterminacy of the vague models processed by generalised test would render HT4/`Core` useless. Fortunately, the observed levels of critique in the *Changing N study* (see Figure 6.18) are non-trivial level of critique. In the range where most runs successfully terminated (|*V*| < 850), between 45% and 75% of the outputs were inexplicable.

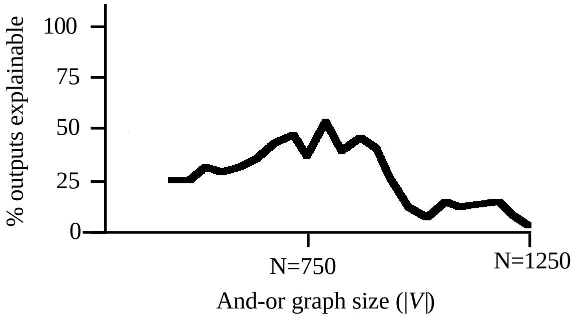


Figure 6.18: *Average percentages of inexplicable outputs observed in the changing N study.*

6.3.2.2. The Changing B Study

Figure 6.19 shows the number of comparisons in the *Changing B Study*. As with all results from the changing *B* study, the results are collected in "buckets" of size |*B*| = 1.

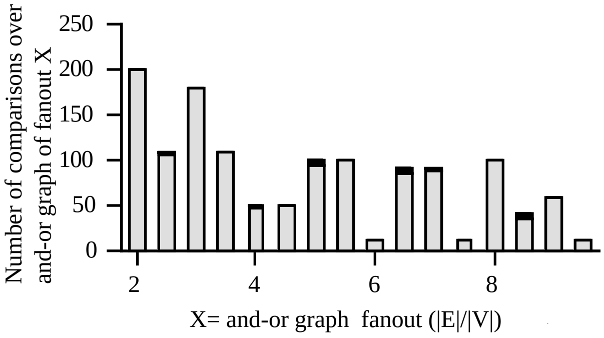


Figure 6.19. *Number of comparisons run on and-or graphs of different fanouts* (|E|/|V|) *in the changing B study. Black denotes runs that did not terminate before the five minute give-up time. For example: 50 comparisons were run over and-or graphs of fanout 5.; and none of the runs at fanout = 5 failed to terminate.*

Figure 6.20 shows how the runtimes varied with model size.

[159] In summary: HT4 is an *abductive* inference procedure and abduction is NP-hard. For more details, see section

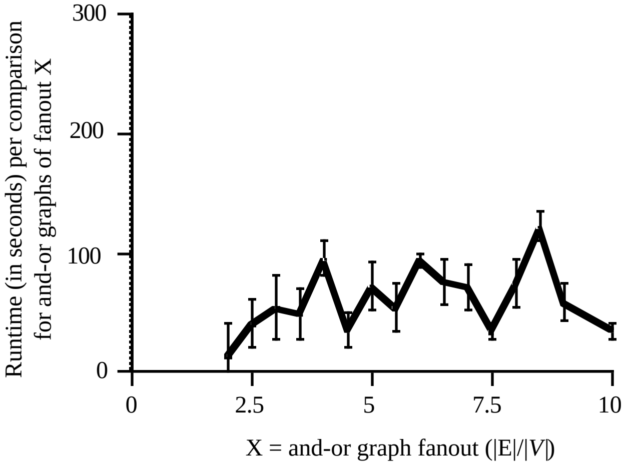
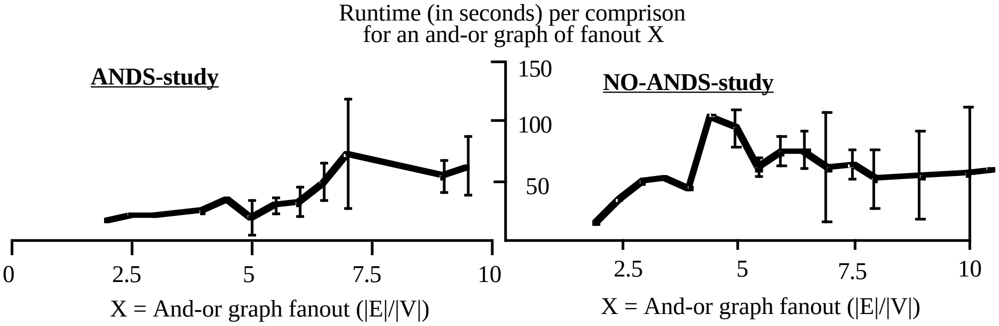


Figure 6.20: *Runtimes vs model size for the changing B study. The displayed runtimes are the means of the runtimes divided into buckets of size |B|=1. Runtimes that exceeded the give-up time of 300 seconds were assigned a runtime of 300. Error bars denote the standard deviation σ of that mean, normalised by the sample size of each bucket; (sample sizes for each bucket shown in Figure 6.19) i.e.*

$$\text{I} = \frac{\pm 2\sigma}{\sqrt{n-1}}$$

The pre-experimental intuition was that runtimes would be exponential on fanout since increasing fanout in a graph containing two nodes *X* and *Y* increases exponentially the number of paths between *X* and *Y*. The observed *B* increases were therefore surprisingly small. Figure 6.19 shows that very few of the runs failed to terminate while Figure 6.20 shows an apparent non-exponential growth. Several factors could inhibit increasing runtimes:

- Frequent incompatibilities of nodes on possible paths would cull the total number of paths generated (i.e. violations of invariants cull the search space).
- Adding links around an and-node increases the pre-conditions to propagation of the search over that and-node. That is, sometimes adding links also adds extra constraints which restricts the number of possible paths. In order to demonstrate this factor, two further changing *B* studies were performed. In the *ands study*, edges were added only upstream of and-vertices. In the *no-ands study*, edges were added only upstream of non-and-vertices. The results are shown in Figure 6.21. Note that the runtimes increase slower when the edges are added upstream of and-vertices (Figure 6.21.i).



(i) *(ii)*

Figure 6.21. *The effects of selective edge addition on runtimes. Figure (i) shows the effects of adding edges only upstream of and-vertices. Figure (ii) shows the effects of adding edges only upstream of non-and-vertices. Runtimes were generated by processing 1 ≤ |I| ≤ 4 and 1 ≤ |O| ≤ 10. Errors bars denote normalised standard deviation (as per Figure 6.16).*

Figure 6.22 records the level of critique found in the *Changing B study* for seven model. Initially, the Smythe '89 graph was mutated to form 76 graphs of fanout B of $2.25 \leq B \leq 10$. These graphs were executed 1597 times using $1 \leq |I| \leq 4$ and $4 \leq |O| \leq |10|$. After a fanout of 4, all the behaviours were explicable. This threshold represents the upper limits to the HT4 model validation process; hence, it was explored further. Six graphs were generated using the *changing N study* model generator with $|L| = 80$; i.e. clones of Smythe '89. These graphs, were of size $480 \leq |V| \leq 535$. Before increasing the fanout for each studied graph, five sets of inputs/outputs were executed ($1 \leq |I| \leq 2, 2 \leq |O| \leq 10|$). The results (see Figure 6.22) show that after $B = 6.8$, nearly all the studied models could explain all known behaviour. The graphs are logarithmic: 89% of the presented behaviours could be explained after fanouts of $B = 4.4$.

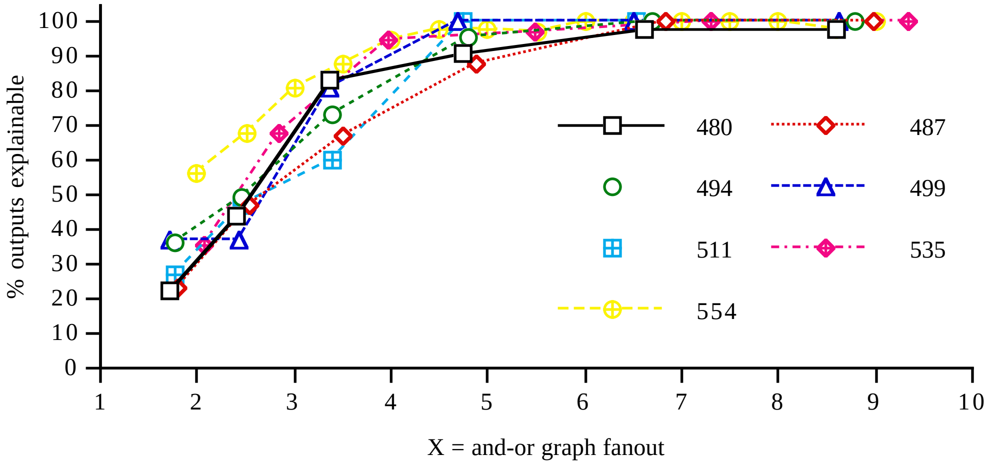


Figure 6.22: *Average percentage of inexplicable outputs observed in the* Changing B study. *Lines are labelled with the |V| of the mutated models. The model labelled 554 is Smythe '89. The other models were generated using the* changing N study *model generator. Note that after a fanout of 4.4 most behaviour is explicable and after a fanout of 6.8, nearly all behaviours are explicable.*

Based on this experiment, we conclude that (i) the Pendrith limit to generalised test is $B = 7$ and (ii) the utility of generalised test is low after $B = 4.5$. The identification of these limits is the major experimental result of this research.

6.3.2.3. The Changing I/O Studies

A surprising result seen in both the changing I and changing O studies was a much-less-than-exponential growth in runtimes with increasing I/O sizes. The pre-experimental intuition was that increasing the size of I and O would increase the runtimes. This was reasoned as follows:

- As I increases, the size of the zone sweeped out downstream of the inputs also increases. Since this zone is the space searched during proof generation, then the number of possible proofs would increase exponentially as more pathways between two vertices became possible.

- As *O* increases, the number of starting points for proofs would increase, this increasing the time required for proof generation.

Neither of these expectations were realised. 100% of the changing *O* runs and 88% of the changing *I* studies terminated within the give-up time.   Figure 6.23 shows the observed runtimes. Bucket size for changing *I*/ changing *O* were 1 and 5 respectively .

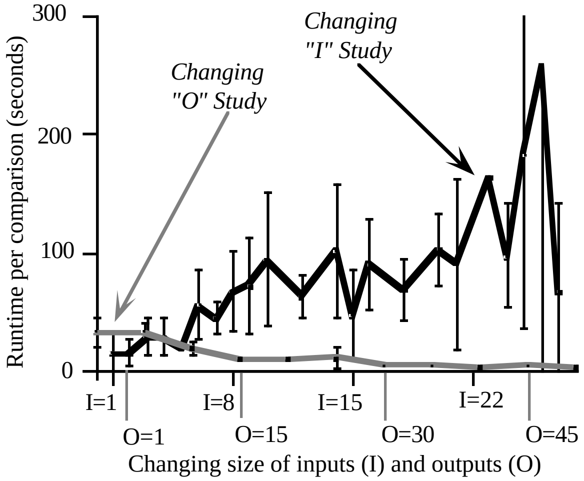


Figure 6.23: *Runtimes vs model size for the* changing I *and O study. The displayed runtimes are the means of all the runtimes divided into buckets of size* |I|*=1 (*changing I study*) and* |O| *= 5 (*changing O study*). Runtimes that exceeded the give-up time of 300 seconds were assigned a runtime of 300. Error bars denote the normalised as per Figure 6.16.*

6.3.3. Discussion

One way to summarise the result from the changing *O* study is as follows. In vague domains, testing is complicated by having to manage assumptions. However, as more data becomes available, testing becomes easier. Consider a variable with *N* states. If one of those states is mentioned in *I* or *O*, then all its mutually exclusive states are ruled out. For example, if you are told that *day=monday*, then you can exclude from the search space all the other states of *day* (*tuesday, wednesday,* etc). |*O*| = 53 for the Smythe '89 model is the case where all variables are measured; i.e. the domain is no longer vague. At |*O*| = 53 for the Smythe '89 model, two-thirds of the search space has been ruled-out. Hence the greatly-reduced runtimes seen in the changing *O* study.

As to the result from the changing *I* study, perhaps the same factors that restrict growing runtimes in the *Changing B study*[160] had some role in slowing down the growth rate of the runtimes.

The *B*=7 limit found by the changing *B* study imposes an upper limit to model evaluation via generalised test. How restrictive is this limit; i.e. is *B* = 7 bigger than the models we find in modern knowledge bases? To answer this question, we need to know the size and internal complexity of knowledge bases found in contemporary knowledge engineering practice. Such data is very scarce[161]. Relevant parameters from a small

[160] See section 6.3.2.2.

[161]

survey of fielded expert systems are shown in Table 6.8[162]. We acknowledge that this is a very small sample size. However, it does represent all the published data we could find. We use Table 6.8 since it is consistent with the author's knowledge engineering experience [157, 164, 206] and the neuroendocrinological models we are aware of.

Application	*N* *(number of literals)*	*B* *(average number of literals in a rule antecedent)*[163]
mmu	65	7
tape	80	4
neuron	155	4
displan	55	2
DMS-1	510	6

Table 6.8: *Model size* N *and average fan-out* B *in the and-or graph of real-world expert systems. From Preece & Shinghal [205]. We call these numbers "reliable" since Preece & Shinghal takes great care to precisely define their terminology.*

Based on that table, we observe that a practical inference procedure for contemporary KBS must work at least for the range $50 \leq (N=|V|) \leq 510$ and $2 \leq (B=|E|/|V|) \leq 7$. Recall the limits found to HT4: $N \leq 850$ and $B \leq 7$. Clearly, HT4 can process models of the size N we see in contemporary practice. However, some limitation is imposed by the fanout since we have seen that after a fanout of 4.4, most behaviours can be generated from indeterminate models.

6.3.4. Conclusions

The mutation study has shown that the generalised test system called HT4 is limited to models with $B=|E|/|V| \leq 7$ and $N = |V| \leq 850$. The B limit seems fundamental to the process of generalised testing. After some level of internal model inter-connectivity, indeterminate qualitative models can explain all known behaviour and we cannot assess them with generalised test. The N limit is a function of our current implementation. However, we have theoretical reasons[164] for believing that generalised test is exponential. Our experimental results do not refute that theoretical belief. Hence, we suspect that faster machines or smarter algorithms or languages will not significantly increase the N limit.

The encouraging result is that for models underneath the Pendrith limit and $N = 850$, no experimental evidence could be found for a limit to the number of explainable items for a model. In fact, the changing O study suggests that the more definite data points to explain, the faster the testing will be.

[162] We exclude from this survey certain exceptional systems that are much bigger than the numbers reported here since they are either extreme outliers (e.g. XCON [9]) or still under development (e.g. CYC [99, 132]).

[163] At first glance, number of literals in rule antecedents is fan-in. not fanout. However, recall from the above discussion that mean fan-in always equals average fanout for a graph.

[164]

The discouraging result is that while these restrictions do not inhibit the processing of some of the models of the size/complexity we see in contemporary knowledge engineering practice, it imposes strict limits on our ability to test models larger than those developed in current practice. That is, current KA practice may be teetering on the limits to testing.

Animals, which move, have limbs and muscles. The earth does not have limbs and muscles; therefore it does not move: Scipio Chiaramonti

The ultimate law: all general statements are false: Anonymous. *I only know two tunes. One is* Yankee Doodle *and the other isn't:* General President Ulysses S. Grant. *You can never successfully fully determine before hand which side of the bread to butter:* Anonymous

7. General

Previously, we focused on knowledge base validation. We have (i) motivated and generalised the concept of testing; (ii) described how to customise generalised test for different domains; and (iii) demonstrated its practicality for models of the size seen in contemporary KA practice. In essence, the process we have described generates multiple containing consistent sets (worlds) of inferences. A post-processor (*BEST*) then assesses the utility of those worlds. An external audience then uses that assessment to (perhaps) declare a model faulty. Note that the audience is external to `Core`.

In this chapter, we step back from the validation task and ask "what else can we use `Core` for?". We will find that the generation of internally consistent inferences is a computation useful in many non-validation domains. In its most general form, this is exactly the model extraction process which Clancey and Breuker argue is at the core of expert system inference. That is, our general test engine is also a general inference engine for KBS.

7.1. Generality in Abductive Domains

This section (i) demonstrates the connection between HT4/`Core` and abduction; (ii) and reviews applications for abduction/`Core` . We will find that a theoretical level and at a useful engineering level, abductive/`Core`-style processing is useful for a wide range of problems.

7.1.1. About Abduction

7.1.1.1. General Notes

Consider a system with two facts a, b and a rule R_1: *If* $a \Rightarrow b$. Informally, we can say that:

- *Deduction* is the inference from a to b.
- *Induction* is the process of learning R_1 given examples of a and b occurring together.
- *Abduction* is inferring a, given b [133].

Abduction is not a certain inference and its results must be checked by an inference assessment operator (e.g. the *BEST* operator of `Core`)[165].

Descriptions of abduction date back to the "fourth-figure" of Aristotle [256]. In the modern era, abduction was described by Charles Sanders Pierce in the last century as follows. "The surprising fact *C* is observed. But if *A* were true, *C* would be a matter of course. Hence, there is reason to suspect that *A* is true" (from the forward of [182]). Intuitively, abduction is inference to the *best explanation*. This intuitive definition is vague on the definition of "best" and "explanation". Definitions for these terms will be discussed below.

Pople noted the connection between diagnosis and abduction in 1973. Pople's diagnosis inference process explores a first-order theory looking for hypotheses which, if assumed, could explain known symptoms [202]. For example, given the first-order theory in Figure 7.1.i, we could hypothesis *{presence(abcess,liver)}* as an explanation for the symptoms *{chills, pain(upper-right)}*. Pople characterised his process in terms of a general logical theorem prover. We characterise it as the extraction of a proof tree from the space of possible proof trees described in the VCD of Figure 7.1.ii[166]. This proof tree must include the largest possible number of symptoms, have roots that are only symptoms and have root(s) taken from the set of acceptable causes (in this case, *presence(X,Y)*).

```
chills ← presence(P,S)∧
         inflammatory(P).
pain ← presence(P,S)∧
       located-in(S,R).
inflammatory(abscess).
located-in(liver,upper-right).
jaundice ← presence(P, liver).
```

(i)

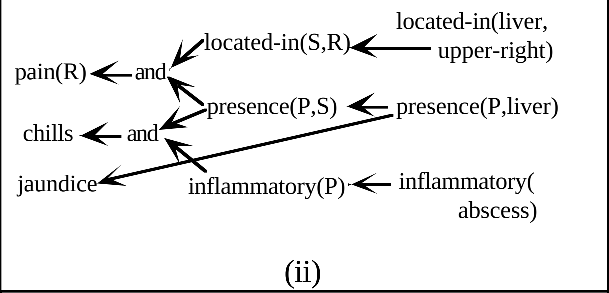


Figure 7.1: *The VCD of Figure (ii) was inferred from the first-order theory of Figure (i) (Figure (i) from [202]).* `X ← Y` *is read as "X could be caused by Y".*

The connection between diagnosis and abduction was confirmed later by Reggia in 1985 [215] and numerous other researchers since (for example, see the discussion below on model-based diagnosis).

By the late 1980s, numerous researchers had recognised the applicability of abduction to a wide-range of domains. The 1990 AAAI Spring Symposium on Automated Abduction [182] lists the following domains as applications of abduction: natural language processing, learning, financial reasoning, analogical reasoning, causal reasoning,

[165] For a more precise definition of abduction, see Table 7.1 in section 7.1.2.1.

[166]

probabilistic and qualitative reasoning, to name a few. Several basic AI algorithms proved to be fundamentally abductive in nature [27, 51, 133]. For example, the ATMS is an incremental abductive inference engine. When a problem solver makes a new conclusion, this conclusion and the reasons for believing that conclusion are passed to the ATMS. The ATMS updates its network of dependencies and sorts out the current conclusions into maximally consistent subsets [56, 59, 85] (which `Core` would call worlds).

7.1.1.2. Definitions & Distinctions

Given a theory *T*, a set of assumables *A*, and a set of goals *G*, then abduction is the search for the subset of *A* (*A*′) such that (i) the goals can be reached given some subset of the assumables without (i) giving rise to contradictions. That is, an abductive search satisfies two rules:

(i) $A \subseteq A', T \cup A' \vdash G$)

(ii): $T \cup A' \nvdash \mathit{false}$

With some text substitutions, this definition is consistent with the definitions of abduction as offered by various authors (e.g. [27, 50, 75, 116, 184, 199, 232]), with certain variants.

In order to test for contradictions, invariant knowledge is required. Some abductive formalisms include the invariants *I* as part of *T* (e.g. Poole [199]), while others explicitly represent them separately (e.g. Kabas & Mancarella [116]).

Poole divides *A* into *defaults*, which are assumed to be true unless we have evidence to the contrary, and *conjectures*, which are assumed only if we have evidence for them [199].

Generally, *T* is a first-order theory. However, this is not universally true. Reggia's diagnosis engine executes over ground frames (i.e. no variables) [213, 214][167]. Formally, Reggia's frames and slots are ground propositions. In a partial match system, frames are inferred if any of its slots are inferred; i.e. $slot_1 \vee slot_2 \vee slot_3 \vee ... \vee slot_n \rightarrow frame_x$ [101].

T is typically acyclic (or, in the terminology of Console *et. al.* "hierarchic" [50]). Cyclic theories present two problems for abduction: *semantics* and *efficiency*.:

- *Semantics:* In qualitative domains, explaining *X* in terms of *X* could be meaningful; i.e. we are explaining *X* at t_1 via *X* at t_2 (where t_2 happened before t_1). However in non-temporal domains, cyclic explanations seem dubious. Further, in the case of non-cyclic theories, a precise object-level semantics for abduction can be proposed in terms of deduction across the Clark completion of a theory [50]).

[167]

- *Efficiency:* Abduction is NP-hard[168] (see below). Algorithms for taming time complexity in abduction often assume acyclic theories (e.g. [75, 201]).

Some researchers (e.g. Kabas & Mancarella [116] and Console et. al. [49, 50]) distinguishes a special set of atoms in T. The *abducible* atoms are those atoms that do not appear in the head of any clause in the logical theory T. For example, in the following theory, `rained_last_night` and `sprinkler_was_on` are abducibles.

```
rained_last_night → grass_is_wet     rained_last_night → road_is_wet
sprinkler_was_on → grass_is_wet      grass_is_wet  → grass_is_cold_and_shiny
grass_is_wet → shoes_are_wet
```

Formal models of diagnosis further distinguish a special set of negated abducibles as the abnormality predicates AB. Reiter's logical framework for diagnosis assumes that all predicates relating to the behaviour of model-component $COMP_i$ (or, in our terminology, V_i) assume a test for "acting normally"; i.e. $\neg AB_i$ [63, 217].

The "explanations" reported by abduction can be either the abducibles (e.g. Console *et. al.* [49, 50]) or the proof trees generated through T that terminate on G (e.g. Poole [199]). Most researchers agree that the explanations should be ground. A' is either the abducibles or the decisions made within the proof tree as it executes over an indeterminate or non-monotonic model. For example, in the theory

$$T_1 = \{ a \rightarrow b, b \rightarrow z\},$$

if $G_1 = \{z\}$, then $A' = \{\mathrm{a}\}$ and A' can be simply the abducibles. Reiter's explanation for a faulty model is a minimal set of assumptions of abnormal component operations (i.e. smallest number of AB_i predicates). However, sometimes knowledge of the proof tree is required for the complete picture. Consider the non-monotonic theory

$$T_2 = \{a \rightarrow b, a \rightarrow c, b \rightarrow z, c \rightarrow d, d \rightarrow z, b \wedge c \rightarrow false\},$$

and the case where $G_2 = \{z\}$. If we return an explanation $Why_2 = A_2' = \{a\}$, then we are not commenting on how the inference managed the incompatibility between b and c. A more complete explanation is the proof trees augmented with the critical assumptions made along the way. $<T_2 , G_2>$ has two proof-trees:

$$Why_3 = \{a \rightarrow b, b \rightarrow z\}; A'_3 = \{a,b\}$$

$$Why_4 = \{a \rightarrow c, c \rightarrow d, d \rightarrow z\}; A'_4 = \{a,c\}$$

Note how abduction treats T differently to deduction. In abduction, T is the space of possible inferences, from which we can extract some subset of inferences that are useful for explaining G. In classical deduction, no portion of T is optional. If the rule *x if y* exists, then in every world where x is true, y is also true [199]. T seem better described as the *RSpace* of Clark & Matwin (i.e. a specification of the space of rules from which ideal domain rules can be learnt [41, 42][169]); or the *scenarios* of Poole:

[168] See section 7.1.1.3 and 7.1.2.2.

[169]

> *The user gives true facts and a pool of possible hypothesis they are prepared to accept as part of an explanation to predict the expected behaviour (i.e. together with the facts implies the observations) which is consistent with the facts (i.e. does not predict anything known to be false) [195].*

Many theories can generate more than one explanation. Sets of consistent explanations are often grouped together and called various terms such as *contexts* [56], or *extensions* [199] (after a term taken from Reiter's default logic [216][170]).

Definitions of the "best" explanation differ widely. Most researchers argue that the best explanations must at least cover all the known output. Some argue that the "best" explanation is the "smallest" one; e.g. (i) smallest number of frames required to explain *G* [213]; (ii) smallest number of abducibles [50]); or (iii) smallest proof size. Smallest proof size would select Why_3 over Why_4 while smallest number of abducibles would prefer either Why_3 or Why_4. Poole [194] and Console *et. al.* [50] have proposed the additional criteria that the "best" explanation also uses the most specific terms from a taxonomic hierarchy; e.g. they prefer explanations in terms of *emu* rather than in terms of the more general term *bird*. Other researchers reject minimally, arguing for more context-dependant "best" assessment. For example, a natural language processing program asked to explain the sentences:

John is an optimist. John was happy. The exam was easy.

could apply minimality to explain John's happiness in terms of his natural optimism. However, the preferred explanation of John's happiness includes information about the nature of the exam. Ng & Mooney use this example to argue convincingly that sometimes the preferred explanation may be the more complex one [177] (see Figure 7.2).

```
(happy j) if
    (succeed j e) if
         (exam e)
         and (easy e)
         and (study j e)
         and (take j e)
```

(i)

```
(happy j) if
     (optimist j)
```

(ii)

Figure 7.2: *Different proof trees used to example "John was happy". Note that the preferred explanation (i) uses more of the theory than the simpler explanation (ii). From [177].*

Due to the variable nature of the definition of "best", Bylander *et al* introduce a plausibility operator *pl* to assess competing explanations [27]. The inner-workings of *pl* are irrelevant to their abductive framework; it may be probabilities, symbolic likelihood, a fuzzy value, etc. However, *pl* must (i) be able to generate a partial ordering on explanations; and (ii) *pl* must be tractable.

[170]

Various researchers note that these abductive definitions of "explanation" are philosophically problematic. Charniak & Shimony comment that, pragmatically, the above logical framework for "explanation" is a useful definition since "it ties something we know little about (explanation) to something we as a community know quite a bit about (theorem proving)." [35]. However, abductive explanation blurs causal implication and logical implication. Charniak & McDermott [34] and Poole [197] caution against mixing up these operators in a single knowledge base. Many researchers acknowledge this as a research area, but then quickly change the topic (e.g. [34] p454, [50] p663, [27] p27, [133] p1061)[171].

7.1.1.3. Complexity

Abduction is much slower than deduction. To see why, consider the theory:

$$T_3=\{a \rightarrow b,\ b \rightarrow c,\ b \wedge c \rightarrow false\}.$$

Given a set of premises (e.g. $\{a\}$), a deductive inference engine could forward-chain to deduce $\{b,c\}$. The forward chainer's search for satisfied rules could be optimised in numerous ways; e.g. an indexing scheme or compiling the rules into a dependency network. Our forward chainer could run in near linear time [69, 119] like some well-trained athlete streaking down the 100-metre dash. However, at the end of the race (inference), the judges will disqualify the results since the deductions include *false;* i.e. in the race to finish the deduction, we forgot to check for invariants. So, we run the race again and this time, we run it abductively. When ever a new conclusion is made, we check it against the invariants. We keep track of the pre-conditions of each conclusion so that if we rule out some conclusion, we can also rule-out the literals that depend on it. Abductive inference is like a race comprising of neurotics. After N steps, inference stops and the current position is intensely criticised. Runners (inferences) are sent back to the start or edged forward in an attempt to make the race legal. After several unsuccessful attempts, it may be realise that this particular race can't be run to any conclusion at all. We may have to split the field and conduct partial races in independent worlds.

This is a slow process. From the space of possible inferences T, some subset is finally chosen which allows us to achieve some goal without generating contradictions. A set of size $|T|$ has $2^{|T|}$ subsets; i.e. a naive abductive inference engine is exponential on theory size. However, even sophisticated abductive procedures may be non-polynomial. Selman & Levesque show that even when only one explanation is required, and T is restricted to acyclic theories, then abduction is NP-hard [232]; i.e. very likely to be computational intractable in the worst-case. An unfortunate feature of abduction is that this worst-case behaviour is often the usual case: most known abductive inference engines exhibit exponential runtimes for real-world inputs, even for sophisticated algorithms. Hence,

[171]

many of the articles in [182] are concerned with heuristic optimisations of abduction. Eshghi report a class of polynomial-time abductive inference problems, but this class of problems require a non-cyclic background theory [75]. Bylander reports techniques for tractable abduction [27], but many of these techniques (e.g. rule-out knowledge to cull much of the search space) are not applicable to arbitrary models developed in poorly-measured domains (e.g. our test domain of neuroendocrinology).

7.1.2. Abduction = Validation

7.1.2.1. Definitions & Distinctions

The model validation algorithms called QMOD/JUSTIN and HT4/`Core` were developed in ignorance of the abductive literature. However, we believe that they can be best characterised as abduction. Recall that a generalised test model M is a graph $<V,E>$ comprising vertices V and edges E. Conceptually, each model has a set of invariants I defined in the negative; i.e. if $I(x,y)$ then x and y violate the known invariants. For each behaviour $B_i = <Inputs, Outputs, Facts, Maybes, Missables>$ generated by the data-compiler:

- A valid search space is defined comprising vertices that are in *Maybes* and are not impossible; i.e. contradict known *Facts* . That is:

  $$Valid \leftarrow Maybes \cap (V - Impossible)$$
  $$Impossible \leftarrow \{v \mid v \in V, f \in Facts \wedge I(f,v)\})$$

- The inference engine attempts to generate proof trees P for $Covered \subseteq Outputs$.
- Initially, a deductive forward sweep finds all the reachable literals from the *Inputs* (the *Relevant* set). This forward sweep is restricted to the *Valid* vertices (i.e. $Relevant \subseteq Valid$). Any vertex $v \in (Relevant - Facts)$ *i*s an assumption A. As a side-effect of the forward sweep, the controversial assumptions $A_c \subseteq A$ are detected (i.e. those assumptions which contradict another assumptions).
- Next, an abductive backwards sweep grow proof trees through *Relevant* from all *Outputs*, searching for *Inputs*. These proof trees will be rooted in $Causes \subseteq Inputs$, and use *Relevant* vertices, no two of which that do not violate I. As a side-effect of the backward sweep, the base controversial assumptions A_b are detected (i.e. those controversial assumptions which are not downstream on other controversial assumptions); $A_b \subseteq A_c$.
- A_b assume X values for Y objects. The `Core` minimal environments *MinEnv* are the consistent combinations of X-Y object-value pairs ($MinEnv_i \subseteq A_c$). The assumptions that contradict $MinEnv_i$ are the exclusions for a world $Exclude_i$.
- A proof is in W_i if the literals it uses do not intersect with $Exclude_i$. A proof may be in more than one world.

- The *Cover* of W_i is the number of output literals it contains, plus any *Missables* that do not contradict the literals in the proofs of that world.
- Worlds are assessed by the *BEST* operator; e.g. $BEST_4$: return all worlds with maximum $|Cover|$.

Note that this maps into our definition of abduction with the substitutions of Table 7.1

Term	Abduction	Generalised test (HT4/`Core`)
Theory	*T*	*M*= <*Valid*,*E'*> where *E'* are edges that connect only *Valid* vertices.
Assumptions	*A'*	<*Causes*, A_b> *Causes* $\subseteq$ *Inputs*
Invariants	*I*	*I*
Goals	*G*	*Covered* $\subseteq$ *Outputs*
Rules	(1) $T \cup A' \vdash G$ (2) $T \cup A' \nvdash false$	(1') $M \wedge Causes \vdash Covered$ (2') $x,y \in vertices(edges(1')) \wedge \neg I(x,y)$
Explanation	*A'* or proof trees of rule 1.	The *E* used in rule 1'.
Sets of consistent explanations	*Context, extension*	World W_x
Assessment	*pl* (many different kinds)	*BEST* (many different kinds)

Table 7.1: *Generalised test is abduction, with some specialised terminology. The functions called by rule 2' return the vertices used in the edges used in the proofs generated in rule 1'.*

Generalised test differs from standard abduction in the following way:

- *T* may be cyclic but *P* must be acyclic.
- Invariants are represented separately to *T*.
- Explanations are proof-tree based.
- Our assumables are always literals.
- No assumption is made about *G* being totally explained; i.e. $|Covered| \leq |Outputs|$.
- No distinction is made between defaults and conjectures. `Core` "defaults" are the vertices that proofs can terminate on which may exist in different worlds; ie. *Inputs* - *Facts* while `Core` "conjectures" = $Inputs \cap Facts$
- Assessment operators are domain-specific, heuristic, and customisable.
- Generalised test is *exhaustive*, *relevant* abduction:
  - *Relevant*: Only those literals that are required for proofs of *Outputs* appear in a world. Consequently: (i) a literal may not share a world with all its logical consequences; (ii) a generalised test world must be defined by data that controlled the inference that created that world, i.e. <*Causes*, *Covered*, *MinEnv*>.

- *Exhaustive*: Generalised test does not seek a single explanation, it seeks *all* explanations. Hence, we would expect generalised test to be slower than the standard abduction defined by Selman & Levesque. This is our theoretical reason for believing that the changing *N* study runtime graph[172] should be non-cubic and probably exponential.

`Core` worlds are *relevant envisionments*; i.e. subsets of the *attainable envisionments*[173] and the *total envisionments*[174] which contain only the literals that exist in proofs connecting *Inputs* to *Outputs*. However, if we simply tell `Core` that the relevant envisionment is the total envisionment, then `Core` can be generate total envisionments. To do this we:

- Set *Inputs* to all root vertices (i.e. vertices with no parents);
- Set *Outputs* to *V* - *Inputs*.
- Set *Facts* to $\emptyset$.
- *Maybes* = *V*
- *Missables* to some domain-specific value; e.g. in the qualitative domains, *Missables* = all the steady vertices.

Note that this computation would be slow. `Core`'s general case is some subset of the total model is ruled-out by known *Facts* or known to be not *Relevant* (i.e. not downstream of some *Inputs*). This `Core`-as-total-envisionment algorithm rules out no portion of the graph.

7.1.2.2. Complexity

HT4/`Core` introduces certain restrictions (for reasons of efficiency):

- *T* is a directed and-or graph with a fixed number of edges and vertices. Such a graph could model a finite propositional theory but cannot model an infinite first-order theory. In order to process models with variables, they must first be partially evaluated by a model compiler into a ground theory.
- *I* has an arity of 2.

Another technique for optimisations HT4/`Core` would be to remove the exhaustive nature of the inference. Generalised test is a unique AI algorithm in that it fundamentally requires an exhaustive search. However, in domains where less-than-all solutions are required, then heuristic cuts can be used to reduce the complexity. The and-or graphs of `Core` would be suitable for standard heuristic search techniques[175].

[172] See Figure 6.16 and 6.17, section 6.3.2.1.

[173] I.E. the literals that are reachable from the inputs, but not required for proofs of outputs, see section 2.3.4.

[174] I.E. the set of all consistent behaviours inherent in some fixed collection of objects in some configuration, see section 2.3.4.

[175] See Pearl & Korf [189] for a general overview and Norvig for detailed implementation guidelines [180]. See

Note, however, that Bylander *et al.* [27] caution that even finding one abductive solution is NP-hard in the case of theories with…

- *incompatibilities* (e.g. the invariants *I* of generalised test); or…
- *cancellations* (e.g. the *and*-vertices and negative propositions of generalised test) that mean a conjunction of literals $L_1 \wedge L_2 \wedge \ldots \wedge L_n$ rules out explanations that are supported by $L_1 \vee L_2 \vee \ldots \vee L_n$ in isolation.

One repeated lesson from AI research has been that theoretically intractable problems may be solvable in practice. Norvig comments:

> *For a theoretical computer scientist, discovering a problem is NP-hard is an end in itself. But for an AI worker, it means that the wrong question is being asked. Many problems are NP-hard when we insist on the optimal solution but are much easier when we accept a solution that might not be the best. ([180], p146).*

In the case of generalised test, we disagree with Norvig. In essence, the results of generalised test is a statement that, after making *all possible generous assumptions*, we still can't explain behaviours O_1, O_2, O_3..... The search for all possible explanations is fundamentally exhaustive. However, when applying HT4/`Core` to other domains, then we agree with Norvig. `Core` could be modified to be non-exhaustive. Currently, `Core` is a generate and test algorithm where all the worlds are generated, then subsequently evaluated. Recall that different *BEST* operators require different information to execute. For example:

- $BEST_7$ returns the worlds that uses edges of highest numeric score (score being set either heuristically via domain knowledge, via a probabilistic analysis, or via some other means). $BEST_7$ is a *vertex-level* inference assessment operator since it could be heuristically applied using only knowledge of the different scores on the edges out-going from some current vertex.
- Best-first-search and beam-search can be characterised as *proof-level* inference assessment operator that are applied during proof generation, given knowledge of the cost of all current partial-proofs.
- $BEST_4$ returns the worlds that covers the most number of outputs. $BEST_4$ is a *world-level* inference assessment operator since it can only be applied once all the proofs and worlds have been generated.

If each *BEST* operator was augmented with a declarative statement of the information it requires to execute, then it is theoretically possible to automatically reconfigure `Core` inference such that *BEST* was applied as soon as practical to constrain proof and world generation.

The rest of this chapter explores the application of `Core` to non-exhaustive domains. There are two cases:

- In the case of `Core` being inefficient for these domains, then this is a theoretical exercise only that serves to unify our view of the processing of these domains. This unified view will be used in the next chapter to assess current knowledge-level modelling techniques.
- In the case of `Core` being adequately efficient for these domains , then `Core` could execute these domains directly: e.g. (i) the vagueness of the model is reduced by careful data collection; or (ii) the operator-levels approach for the *BEST* operators described above is used.

We believe that `Core` is adequately efficient for many domains since the last chapter taught us that, over a medium size of models ($|V| \leq 850$), we lose our ability to test a model in vague domains[176]. Subsequently, we will argue that we should limit model construction to those models which we can test; i.e. `Core` should never need to execute more-than-medium sized models.

7.1.3. Model-based diagnosis

Diagnosis has been discussed informally above. This section offers a more rigorous analysis of diagnosis and its connection to abduction and HT4/`Core`.

The connection between abduction and model-based diagnosis is well documented. Pople and Reggia acknowledge that their "diagnosis" systems are really abduction [202, 215]. Poole's abductive framework *THEORIST* can be used as a diagnosis tool [196, 197, 199]. Console & Torasso characterise the two main types of diagnosis as variants of the same abductive inference algorithm [51]. Both types of diagnosis input (i) a system description of the system to be diagnosed (i.e. a model[177]); (ii) a set of observations *OBS*; (iii) and a context *CXT* in which the *OBS* were made. Two sets are then deduced: (i) a set of observables that must be avoided Ψ^- (i.e. any observables inconsistent with *OBS)*; and (ii) a set of observables that must be covered Ψ^+.

Consistency-based diagnosis (e.g. [65, 94, 217]) sets $\Psi^+ = \emptyset$ while *set-covering diagnosis* (e.g. [48, 197]) sets Ψ^+ to *OBS*. Set-covering diagnosis is best when the knowledge base contains knowledge of faulty operations while consistency-based diagnosis is best for knowledge bases containing knowledge of normal operation. For a comparison of the two approaches, see Figure 7.3.

[176] See Figure 6.17.

[177] Console and Torasso further divide a model into a set of components COMP and knowledge of the

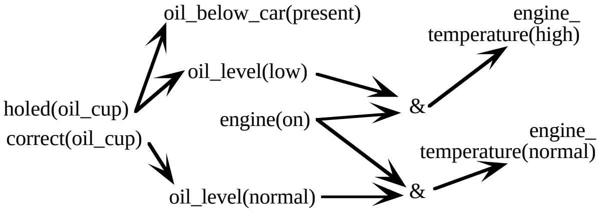


Figure 7.3: *If* CXT = {engine(on)}, OBS ={oil_below_car(present)}*, and we restrict the diagnosis to* oil_cup*, then set covering diagnosis returns* {holed(oil_cup)} *while consistency-based diagnosis returns* {holed(oil_cup)} *or* {correct(oil_cup)}*. Example from* [51].

Konologie argues for the primacy of set-covering diagnosis. He notes that consistency-based diagnosis returns answers that may not be relevant to causal explanations of *OBS* [120] (e.g. *correct(oil_cup)* from Figure 7.3 has little bearing on the problem of *oil_below_car(present)*).

Having stressed the differences, we note that generalised-test can be used for either consistency-based or set-covering diagnosis (see Table 7.2).

	Generalised test configured for set-coverage diagnosis	*Generalised test configured for consistency-based diagnosis*
Inputs	CXT	CXT
Outputs	OBS	V - CXT[178]
V - Maybes	$\Psi^- = \{x \mid x \in V, y \in CXT \cup OBS, I(x,y)\}$	Ψ^-
BEST	favour the world(s) that cover all of $\Psi^+ = OBS$	return all worlds

Table 7.2: *Generalised test configured for different model-based diagnosis tasks of a model with vertices* V. *For set-coverage, explanations are attempted for all* OBS. *For consistency-based diagnosis, explanations are attempted for all non-input vertices. V - Maybes* is the FORBIDDEN *set. In model-based-diagnosis, any vertex that contradicts* CXT ∪ OBS *is forbidden.*

We argue that generalised test is a generalisation of model-based diagnosis since (i) *BEST* permits the generation of worlds with partial coverage of *OBS*; and (ii) the *BEST* operator allows for the customisation of the world preference criteria.

A commonly used support-routine for diagnosis is *probing*; i.e. the guided search for additional information which can confirm/ rule-out a diagnosis [54]. Given a set of possible explanations, a carefully selected probe for a single piece of information can cull numerous explanations. DeKleer uses the TMS structures of his general diagnosis engine GDE to guide probe selection [65]. In the case of `Core`, a probe that checks for the value of any assumption would cull the possible proofs. The usual case is that each probes have an associated cost (e.g. taking a blood-pressure reading is a cheap probe while performing exploratory surgery is an expensive probe). Probe selection is a trade-off between information gain and probe cost. A heuristic `Core`-based probe-selection algorithm could favour probes of the base controversial assumptions A_b over probes that

[178]

test other assumptions since rejecting any of member of A_b can remove many worlds with a single measurement.

7.1.4. Prediction

Like Poole [199], we note that the same architecture can support both explanation and prediction . `Core` can support prediction as follows:

- Set *Inputs* to the known inputs; i.e. the things that our predictions will be based on.
- Set *Outputs* to the model vertices that are not inputs; i.e. *V* - *Inputs*
- Read the generated worlds are mutually exclusive predictions of what could result from the *Inputs*.

Note that:

- `Core` was designed to handle the general case where *X* inferences are generated in *Y* internally consistent worlds. Using `Core` for prediction trivially implements two hard cases for prediction: (i) the case where certain literals can be predicted separately, but not together in the same world; (ii) the case where prediction occurs in domains with less-than-certain knowledge. In case (ii), `Core` can manage the assumptions and *BEST* can favour (e.g.) those predictions that require the least number of assumptions.
- This prediction algorithm computes the attainable envisionments; i.e. a much larger search space than the relevant envisionments executed by standard `Core`-as-a-validation-algorithm. To cull that search space, use the context *OBS* observations *OBS* distinction offered by Console & Torasso [51]. Let *Inputs* = *CXT* and *Facts* = *OBS* $\cup$ *CXT* . The best case for this prediction algorithm is where |*OBS*| » |*CXT*|; i.e. very few starting points (*CXT*) for the prediction and lots of constraints (*OBS*) exist for possible values in the search space. We says that *OBS* constrains the search space since, recalling the changing *O* study, we found increased amounts of data culls indeterminacy in the inference.

7.1.5. Explanation

We have discussed above the definition of abductive explanation. This section explores current research in explanation. This current view of explanation is more elaborate than the "print the rules that fired" approach used in early expert systems such as MYCIN [24].

> *The traditional expert system explanation solution is to augment, prune, transform, or in some other way manipulate a complete trace of the event to be explained… Certainly, support knowledge can be added, describing the knowledge found in the trace, but new knowledge of new movements through the knowledge base cannot be added. For example, traditional expert system explanation methods cannot use additional symptoms to support the conclusion of the expert system. [259]*

In the current view, explanation is a problem solving task in its own right. Explanation is an inference procedure that determines what is to be presented to the user. Explanations are user-specific. "The audience of an explanation can significantly affect the purpose and therefore the content of an explanation" [259].

Leake [129] and Paris [185, 186] discuss explanation algorithms where explanation presentation is constrained to those explanations which contain certain *significant structures*. Paris's significant structures are determined at design time while Leake assigns significance at runtime.

- Paris's experimental results suggest that expert's use parts-based explanations while novices use process-based explanations. Edges in her system are tagged as being part either *process-based* or *parts-based*. Knowledge of the expertise of the audience is used to tag each vertex as "known to user" or "unknown to user". When faced with a choice of edges to be used in an explanation, Paris's explanation algorithm selects either a process-based trace or a parts-based trace according to an examination of the local vertices in the network. If the local vertices are "unknown", then the process-based descriptions are preferred.
- Leake assigns significance at runtime according to a set of eight pre-defined algorithms. For example, when the goal of the explanation is to minimise undesirable effects, the runtime significant structures are any pre-conditions to anomalous situations .

Leake clearly acknowledges the connection of his work with abduction [130]. Note that both Leake's and Paris's algorithms can be characterised in terms of operators that select some subset of the possible inferences according to a user/goal-specific criteria; i.e. they are compatible with our *BEST* formalism.

7.1.6. Classification (= Prediction)

Applying the same notion of "significant structures", we can adapt our symbol-level algorithm to a variety of classification algorithms. Consider the dependency graph of a rule-base developed for classification purposes. The possible output classification *CLASSES* are a special subset of all the literals in the knowledge base (e.g. all the literals with in-edges, but no out-edges).

`Core` views classification as a special kind of classification. $BEST_{classify}$ favours the worlds with the most number of *CLASSES*. Note `Core` handles two non-trivial cases for classification: (i) multiple classification where some classifications are inconsistent with other classifications; (ii) classification with partial match. In case (i), multiple classifications that were specified as incompatible would appear in separate worlds. In case (ii), incomplete information can still be used to make a classification since `Core` would merely make certain assumptions. to the classified classes. Note that `Core`-as-classification makes no distinction between single and multiple-classification.

A common construct in classification system is a taxonomy hierarchy. To use such hierarchies in `Core`, the model-compiler would add edges from sub-classes to super-classes; (e.g. *animal if dog; organicThing if animal or plant*). Specialised inferences would trigger more abstracted inferences.

As to using the most specific terms from a taxonomic hierarchy (i.e. as proposed by Poole [194] or Console *et.al.* [50]), we view this as a report issue. Once the worlds are generated from a classification system, the presentation of those worlds to the user could be customised according to (e.g.) the expertise level of the user. Users fully conversant with all concepts in the domain could be presented with all conclusions at the lowest level of taxonomic abstraction. Beginners might prefer a presentation based on more familiar, and more general conclusions.

7.1.7. Planning

Like Poole [200], we note that the same architecture can support both explanation and planning. In the case of $Inputs \cup Facts = \emptyset$, the knowledge engineer is asking `Core` to hypotheses sets of mutually consistent pre-conditions that would lead to the desired results (*Outputs*). The proof trees connecting these pre-conditions to the results are a plan for achieving the *Outputs*.

Kanovich uses exactly this schema to optimise functional evaluation. Kanovich [119] characterises the processing of functional knowledge as a three-stage process:

- Given a task of the form "for the knowledge base KB, find Z given X…"
- …determine whether the functional dependency $X \rightarrow Z$ follows from the KB…
- …synthesis a *solution* to the task; i.e. using the laws of the KB, compose a program which, given X computes an output list Z.

Note that Kanovich's *solution* is a plan to reach Z, from X.

Re-expressed in terms of `Core`, Kanovich's formalism becomes (i) X = Inputs ; (ii) Z = *Outputs*; (iii) *solution* = world; and (iv) $BEST_{kanovich}$ favours the world(s) with smallest proof cost.

Kanovich's implementation of his theory runs over Pascal source code to extract the headers of each source procedure. The inputs and outputs of the procedures are studied and each procedure header is added to a dependency network that is a and-or graph. Pre-conditions to executing the procedure are stored as conjunctions upstream of proposition representing the procedure call. Possible outputs from each procedure are out-edges from the procedure proposition. A planner runs over this network to return the world with minimum cost. This world is then compiled into the main procedure of a Pascal program which calls the procedures in the original source code. Kanovich ignores invariants between literals and so his generation of the least-cost world executes in near-linear time.

In a similar approach, Freeman-Benson *et. al.* [87] discuss code extraction from a constraint network.

> *Extracting a (compiled or interpreted) plan from (our algorithm) is easy because the plan is inherent in the directed acyclic dataflow graph* P. *A simple procedure* `extract_plan` *traverses* P *and returns a serialised list of constraint methods. There are two ways to extract the plan: top-down and bottom-up. Top downs starts at the source variables (those that are determined by constraints having no inputs) and works forwards. Bottom-up starts at the sink variables (those that are not used as inputs by any constraints) and works backwards. ([87], p59).*

Consider the similarities of their approach to `Core`. Their domain is represented by a directed graph. Once such a declarative representation is available, it can be exercised in multiple ways (bottom-up or top-down). "Plans" are a report of the valid proof trees that can traced out across the graph. However, note that Freeman-Benson *et. al.* stress the simplicity of their `extract_plan` procedure. This suggests that they handle the special case of models containing only one stable state (i.e. no need for multiple-worlds reasoning).

7.1.8. Monitoring

To use `Core` for monitoring, the generated proof trees of the preferred plan (i.e. a world chosen by *BEST*) would be passed to a tracking system. As the plan executes and feedback is received from the process being controlled by the plan, the tracking system could mark the believed literals in the plan. Literals from *Inputs* or the measurements from the feedback would be marked *true*. Pre-conditions for the true literals would be marked *required*. If new feedback contradicted *true* or *required* literals, then the plan has failed. At this point, the tracking system has two options:

(i) If could run `Core` again to get a new plan. Note that in the time since the last run of `Core`, the tracking system has received more input from the process being controlled. If this extra input is passed into the next run of `Core` , then this could serve to constrain `Core` inference.

(ii) Space permitting, when the tracker is initialised, it could be passed *all* the worlds generated by the first run of `Core` . Markers could be inserted simultaneously through all worlds. As feedback rules out certain worlds (i.e. new facts come to light that rule out certain assumptions), then the set of believed worlds shrinks.

Note that option (i) is an JTMS approach [70] (i.e. work with a single world view and generate another when it fails us) and option (ii) is an ATMS approach [56] (generate all worlds simultaneously).

A *passive monitor* just watches for inconsistencies between feedback and the required/true literals. An *active monitor* requests information from the environment in order to check the assumed required literals. In the case of an active monitor using probing to evaluate multiple plans, then that monitor is performing the same probing process described above in the section on model-based diagnosis.

7.1.9. Causal/ Qualitative Reasoning

In this section, we quickly repeat out earlier claim regarding the utility of `Core` for causal/qualitative reasoning[179].

`Core` evolved from experiments with adding causal intuitions to mathematical formalism. After several years of study, our conclusion was that knowledge-based causality was an asymmetrical *optional* inference that allowed (i) an expert to specify some superset of the space of acceptable explanations and (ii) an inference engine to explore that space with respect to a given task (i.e. the generation of a consistent explanation of certain outputs given certain inputs). At its lowest level, such an exploration is exactly the abductive inference described above.

More generally, we offer this as technique for modelling qualitative reasoning. Experts can sketch their intuitions in qualitative vague causal diagrams. Abduction can generate from those diagrams domain theories (read worlds) that can execute via monotonic deduction. Ambiguity in the extraction process can be tamed via partial knowledge of the behaviour of the entity being modelled; i.e. worlds that include literals that contradict known observations are rejected. In the case of KA from multiple experts, we can relatively assess the proposed models of different experts. $Expert_1$*'s* qualitative theory is better than $Expert_2$ *'s* if $Expert_1$*'s* theory covers more of the known behaviour [156].

7.1.10. Others

Leake characterises case-based reasoning as an abductive process where the possible explanations are assessed via a library of prior cases [130].

Numerous papers in [182] describe natural language comprehension in terms of abduction[180].

Poole maps design and recognition into abduction:

- Design is the process of hypothesising components which would imply a design goal [198] (i.e. a process very similar to the planning scheme discussed above).

[179] The connection between causal and qualitative reasoning was discussed previously in sections 2.3.5.

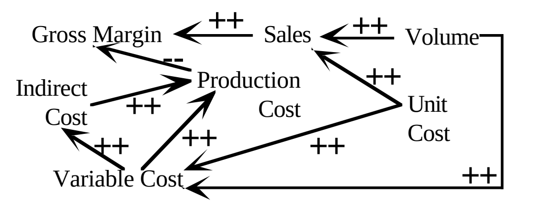
[180]

- Visual pattern recognition is a process of hypothesising scene objects which would lead to the perceived image. Poole demonstrates that this perspective gives the same results as other visual imagery researchers [199].

Hamscher notes that certain sub-tasks in financial reasoning (financial assessment, going concern evaluation, auditing, and the explanation of unexpected financial results) are all abductive tasks [100]. Hamscher's scheme for explaining unexpected financial results relies internally on a vague causal diagram (which Hamscher calls the *hypothesis space*). The equations of Figure 7.4.i could be compiled into the and-or graph of Figure 7.4.ii. Given some observation (e.g.) *Unit Cost increased*, then we could explain (e.g.) *Gross Margin decrease* via the proof *Gross Margin decrease* → *Production Cost increase* → *Variable Cost increase* → *Unit Cost increased* (example from [100]).

```
GrossMargin =
  Sales - ProductionCost
Sales =
  Volume * UnitCost
VariableCost =
  Volume * UnitCost
ProductionCost =
  VariableCost + IndirectCost
IndirectCost =
  Variable Cost * 15%
```

(i)



(ii)

Figure 7.4: *The graph of figure (ii) can be generated from the equations of figure (i). Example from [100]. Note that knowledge of the relationship between literals is used deduce when an edge is a "promote" or "discourages" influence. For example, in the top equation,* ProductionCost *lies behind a minus sign. Therefore inverse(*ProductionCost,GrossMargin*).*

Theoretically, abduction is also a machine learning technique. Recall our definition of abduction:

$$\text{(i)} \quad A \subseteq A', T \cup A' \vdash G\,)$$

$$\text{(ii)} \quad T \cup A' \nvdash false$$

In our work, we have restricted *A* to sets of literals. Pagnucco notes that the above definition makes no such restriction. In the case where *A* is the space of all first-order formulae, then the addition of new knowledge *A'* to a theory *T* is learning [183] or (more precisely) belief-revision [184]. Hiarata makes a similar claim. He characterises inductive logic programming (a technique for learning first-order theories[181]) as an abductive process where the search space for explanations is either in the current theory (*selecting abduction*), an analogous theory (*finding abduction*), or a theory especially created from a generalisation of known theories (*generating abduction*). More generally, Hiarata argues that scientific theory formation is an abductive process [104].

While we do not doubt the theoretical truth of Hirarta's and Pagnucco's arguments, we are unsure about the practical utility of characterising abduction-as-learning. We have

[181]

discovered above that abductive search through a fixed space is a slow process[182]. Our pre-experimental intuition is that an abductive search through a space that is learnt as the search progresses would only be practical for very small models.

7.2. Generality in Non-Abductive Domains

This section argues for the use of `Core` in non-abductive domains.

7.2.1. Deduction

Domains requiring a simple modus ponens (i.e. forward-chaining) could be implemented trivially in `Core`. After the deductive forward sweep, `Core` could skip the abductive backsweep or the world sweep. Candidates for such simple inference include all models with no invariants.

7.2.2. KBS Verification

We have discussed above the utility of `Core` to KBS *validation*; i.e. the assessment of a model according to external semantic criteria such as covering known behaviour. We find that `Core` is also useful for KBS *verification*; i.e. the internal assessment of a model according to internal syntactic criteria such as presence of contradictions. A general framework for KBS verification is offered by the Preece, Shinghal & Batarekh study (PSB) [204] and was summarised previously[183]. `Core` could detect may of the PSB anomalies. To see this, recall some of the data structures we used to define `Core` (reproduced in Figure 7.5).

```
object   = record  name: string; id: integer;
                   class: class;
                   familiarity, probeCost: integer; end;
proposition = record object: object; value: any;
                     test:{≤,<,=, ≠,>,≥};
                     time: integer; end
literal  =  record negated: boolean; p : proposition;
                   vertex  end;
literals =  list of literals;
source   =  … -- some structure in the user view
vertex   =  record l : literal; parents, kids: posints;
                    id: posint; contradicts: bitstring;
                   and, input, relevant,
                   impossible, controversial: boolean; end;
vertices =  list of vertices;
edge        = record id : posint; from,to: posint; kindOf: thing;
                     attempts,successes,cost: integer;
                     source: source  end;
edges    =  list of edge;
graph    =  record e : edges ; v : vertices end;
```

Figure 7.5: *Partial display of* `Core` *data structures.*[184]

[182] See section 6.3.

[183] See section 3.3.1.

Given a knowledge base comprising a set of `source` statements (e.g. rules) referring to `literals`, a `graph` = <`verticies,edges`> is created. Using knowledge of defined `object`s (stored in some variable called, say, `DataDictionary`), KBS anomaly routines can use critique the `graph.` The model-compiler could detect trivial typographical errors such as an unknown `object`, or an illegal `value` for that `object.` As to the PSB anomalies:

- `Core` would detect PSB *rule redundancy* if the `edges` from the same `source` touch `vertices` that are incompatible (defined by the `vertex.contradicts` set).
- PSB *deficiency* has two causes (i) missing values, i.e. not all the range of an object has been used by graph vertices; or (ii) missing rules, i.e. source statements that do not generate edges that connect askables to final hypotheses. To test for case (i), we merely check that the values found in `graph.vertices.lierals.proposition.value` cover the domain of all `object`s in the *DataDictionary*. To detect this, we would run `Core` with *Inputs* = *Askables*, *Outputs* = *Final hypotheses* , *Facts* = Ø, *Maybes* = *V* and check for worlds with zero coverage of *Outputs*.

`Core` does not support anomaly detection for all the PSB anomalies. However, the `Core` data structures offer some support for their detection. For example:

- A PSB *duplicate redundant rule* could be found by a bit-string comparison of the `id`s of the `literals` in the different `source` statements. Given hardware support for the bitstring manipulation, this would execute quickly. `Core` model-compiler partially evaluates the first-order case into the propositional case; i.e. under `Core` the test for PSB *subsumed redundant rule* would have to be implemented as a test for duplicate redundant rules.
- PSB *ambivalence* is invisible to `Core` since any such contradiction in a knowledge base would simply generated separate worlds. We could attempt to compute this under `Core` using ATMS-style total envisionment (i.e. *Askables* = *Inputs* = all known possible system premises, *Final hypotheses* =*Outputs* = *V*-*Inputs*, *Facts* = Ø) but the generated worlds will never contain consistent subsets of the *Inputs* and inconsistent subsets of the *Outputs* since `Core` insists that all worlds are internally consistent. However, we could report PSB ambivalence if more than one world was created.
- PSB *circularity* between knowledge base literals is also invisible to `Core`. However, circularities could be simply detected via transitive closure of the adjacency matrix representing `graph`.

7.2.3. Decision Support Systems

Henri Fayol suggested in 1916 that managers plan, organise, co-ordinate and control. This lead to a view of managers as agents systematically assessing all relevant factors to generate some optimum plan. Sometime in the sixties, it was realised that electronic computers could automatically and routinely generate the information required for the Fayol model. This lead to the era of the management information system (MIS) and the wastage of a lot of paper. Managers found themselves overloaded with more information than they could manage. Mintzberg's classic empirical fly-on-the-wall tracking of managers in the day-to-day work demonstrated that the Fayol model was normative, rather than descriptive. For example, a study of 56 U.S. foreman found that they averaged 583 activities in an eight-hour shift (one every 48 seconds). Another study of 160 British middle and top managers found that they worked for half an hour or more without interruption only once every two days [168]. This frantic pace for decision making does not match with Fayol's model of managers as systematic agents.

The lesson of MIS was that management decision making was not inhibited by a lack of information. Rather, it is confused by an excess of *irrelevant* information [3]. Modern decision-support systems (DSS) evolved to filter useless information to deliver relevant information (a subset of all information) to the manager. "Relevance" and "usefulness" are a user-specific and problem-specific concept. For example, a knife in a kitchen causes no distress and may not interest a non-hungry human. However, the same knife gleaming in a dark alley way may cause a different reaction.

Our preferred definition of a decision-support system is based on Brookes who developed it from Mintzberg's model [22]. The goal of a DSS is "comfort", i.e. a subjective impression that all problems are known and under control. A taxonomy of tasks that create a sense of comfort are shown in Figure 7.6.

```
Comfort = 1+2+3.

1. Finding problems
          1.1. Detection
          1.2. Diagnosis

2. Solving problems
          2.1. Alternative generation
          2.2. Evaluation
          2.3. Judgement

3. Resolution
          3.1. Monitoring
```

Figure 7.6. *Tasks involved in finding "comfort" for a manager. Adapted from* [22]. *Managers seek problems, solve them, then install some monitoring routine to check that the fix works.*

In the case of a model exists of the domain being analysed by the DSS, `Core` offers numerous utilities that support comfort finding:

- *Diagnosis* and *monitoring* were discussed above.

- Problem *detection* could be implemented via *prediction* or *classification* (where some of the classifications are problem sub-types) or even simple forward chaining.
- *Alternative generation* and *evaluation* is another name for *world generation* and *assessment*. `Core` is particularly suited to alternative generation and evaluation since it manages contradictory alternatives in different worlds.

Note that these sub-routines for decision support systems were discussed above; i.e. abduction/`Core` provides a unifying framework for model-based single-user DSS. For example, a standard DSS function is a "what-if" query is which users explore hypothetical options. Implementing such a query system over a multiple-worlds architecture such as `Core` would be trivial.

Boose, Bradsahw, Koszaek, and Shema (BBKS) [14] discuss DSS architectures suitable for groups (GDSS). Portions of the BBKS and the Brookes' models overlap (e.g. the generation and ranking of alternatives). However, BBKS discuss a wide range of other modules such as a *establish relationships* and *generate criteria* sub-systems which model the group dynamics where the group's definition of:

- the model and its inter-relationships; and
- the criteria for selecting the best alternative

Within the BBKS gIBIS system, the *establish relationships* tool is an outliner where users can specific arbitrary taxonomies of arbitrary types. *Criteria* are defined in terms of free text entries. Other tools allow the processing of *mediating representations* (intermediary representations that particular focus groups find to be locally valuable), and a suite of tools taken from the knowledge acquisition community.

The BBKS study teaches us that while `Core` offers substantial support for single-user model-based DSS, it only supplies partial support for group DSS. However, many of the problem-solving methods that BBKS list as GDSS sub-routines (e.g classification, diagnosis, planning) could be implemented as `Core`-variants according to the above discussion. Further, BBKS stress that:

> *The process of generating and scoring alternatives are at the heart of most decision processes. [14]*

Abduction/`Core` offers a principled approach to the generation and assessment of worlds across the indeterminate/vague models that could be generated by groups exploring ideas. `Core` provides considerable support for executing such qualitative models in poorly-measured domains, and dividing the results into mutually exclusive worlds. This is no accident: `Core` evolved from QMOD/JUSTIN which were developed as a type of GDSS for neuroendocrinological theories. `Core` can act as a judge, assessing the models offered by multiple experts to declare one as the "winner" by some locally acceptable *BEST* criteria. Note that group members do not need to fully specify their models prior to executing them. Using `Core`, executables can be generated from half-formed notions to give experts feedback on their ideas. We note that (i) gIBIS appears to lack such an evaluation module for vague ideas; and suggest that (ii) abduction/`Core` would be a useful addition to a GDSS architecture.

7.2.4. Expert Critiquing Systems

The introduction to this report defined our goal: general implementation principles for *deep models* and *differential analysers*. Recall that these terms were from Silverman's expert critiquing system (ECS) formalism [235-237] which he defines as:

> *…programs that first cause their user to maximise the falsifiability of their statements and then proceed to check to see if errors exist. A good critic program doubts and traps its user into revealing his or her errors. It then attempts to help the user make the necessary repairs*[185]*. [237]*

Abduction/`Core` offers a unified framework for portions of the ECS framework. Deep models generate model behaviour and a differential analysers compare the generated behaviour with the expected behaviour. `Core` deep models are propositional systems which can lead to desired outputs from known inputs. $BEST_4$ (coverage of desired outputs by a world) is a pseudonym for differential analyser. If no worlds covers all outputs, then the model's behaviour is different to the required behaviour.

Silverman's critiquing framework describes four levels of tests: *clarity, coherence, correspondence, workability*. `Core` provides detailed support for two of these levels and partial support for a third:

- KBS coherence measures internal syntactic criteria [237]. The `Core`-as-verification algorithms described previously could serve as a KBS coherence critic.
- KBS correspondence measures the equivalence of a KBS to the entity being modelled [237]. Standard `Core` running $BEST_4$ is a KBS correspondence critic.
- A KBS has poor clarity if it is ambiguous. `Core` is a poor clarity critic since its usual case is the processing of ambiguous/indeterminate models. However, an external observer could declare a model to be overly-ambiguous if `Core` generates too many worlds.

Workability is a measure of the inner processing of a model. Workability critics are complex and may be (e.g.) a machine learning program that criticises a rule-trace to optimise the internal processing. `Core` is not a workability critic.

A third component of an ECS is the dialogue generation module that discusses the result of the differential analyser with the user [236]. In Silverman's framework, these are special-purpose domain-specific functions. Dialogue generation is out-of-scope of our

[185] ECSs are therefore much broader than the definition instantiated by ATTENDING [166] which had no

current research. Mahidadia is exploring dialogue generation in an inductive logic programming framework for QCM-type models [137, 140]. Just as `Core` offers general principles for model execution and faulting, we believe that Mahidadia's work will provide general principles for user-guided model repair (a.k.a. dialogue generation).

Silverman's research seems to be focused on an implementation-independent analysis of the process of "critiquing" a program. His focus seems to be on defining "critiquing" as an add-on to existing systems. For example, his COPE system is a stand-alone product. In order to use models generated from COPE, these models must be hand-compiled into other languages. We believe that while this approach is useful, a more extensible approach would to change the structure of knowledge-bases systems such that critiquing is built into the system. In the case where the design of the system can be altered to integrate a testing module, we believe that the built-in approach is superior since built-in critics could guide the knowledge acquisition and maintenance process. However, if the design of the system cannot be altered according to the principles we have described above, then Silverman's add-on approach is appropriate.

7.3. Model Extraction

This chapter has suggested that many knowledge engineering problems can be processed by HT4/`Core`. Is this just coincidence? Or is there some fundamental reason why this is so? To answer that question, we turn to two attempts at meta-level generalisations of expert system inference: (i) Clancey's *model construction operators* [40]; and (ii) Breuker's *components of problem solving types* [21]. In summary, we will argue that both these general descriptions of inference can be described as *model extraction*. We will then argue that `Core` directly implements this model extraction process.

Clancey characterises expert system inference as constructing the system-specific model (SSM) from a general qualitative model (QM) in the KB[186]. Clancey stresses that the QM is not a set of pre-enumerated solutions to known problems. Rather, the QM is a general domain model which is "accessible and interpretable for multiple purposes" [40]. For example, Clancey would take the production rule of Figure 7.7.i and separate it out into (i) the inference strategy it proposes; (ii) the declarative statements about the domain it contains; (iii) and the heuristic knowledge it stores (see Figure 7.7.ii). The new form of the knowledge would be a QM containing edges representing causal relations, subtype relations, temporal relations, etc[187].

[186] In using the phrase "qualitative model", Clancey is appropriating a term used extensively by the naive physics community. Numerous researchers explored qualitative reasoning in the domain of first-order linear differential equations. For more on this research, see sections 2.3.3 to 2.3.5.

[187]

```
METARULE-1
IF      1) The infection is
           PELVIC-ABSCESS
        2) There are rules
           which mention in
           their premise
           ENTEROBACTERIACEAE
        3) There are rules
           which mention in
           their premise
           GRAM-POSITIVE-RODS
THEN:  There is evidence (.4)
       that the former should
       be done before the
       latter.
```

(i)

```
common_cause(enterobacteriaceae,
             pelvic_infections).

unlikely_cause("g+ rods",
               pelvic_infections).

task(testHypothesis,
  Disorder,
  doBefore([task(testHypothesis(Hyp1)),
            task(testHypothesis(Hyp2))
           ]))
 :-
 common_cause(  Disorder, Hyp1),
 unlikely_cause(Disorder, Hyp2).
```

(ii)

Figure 7.7. *Figure (i) shows hard-wired knowledge. Figure (ii) shows the same knowledge represented in a more widely-applicable form. Domain-specific terms (e.g.* ENTEROBACTERIACEAE*) are isolated and the inference strategy implicit in Figure 7.7.i is represented as a first-order theory. Example from [40]*[188].

Clancey's QM is a first-order theory. At runtime, portions of this theory are accessed and the variables are bound. This ground subset of the full theory is the SSM; i.e. "the specific model the program is constructing of the particular system it is (processing)" [40]. This specific model is the subset of the QM that is relevant to the task at hand.

Breuker explores the relationships between problem solving techniques used in expert systems (i.e. modelling, planning, design, assignment, prediction, assessment, monitoring and diagnosis) [21][189]. He offers an abstract description of the "components of a solution" generated by these techniques which , he argues, are of four types:

- A *case model* (equivalent to Clancey's SSM) that represents some understanding of a problem;
- A *conclusion*, which is some answer to a question posed by the problem definition;
- An *argument structure*, which is supporting evidence for the conclusion generated.
- The case model is generated from some *generic domain model* (equivalent to Clancey's QM).

An argument structure is extracted from the case model. The conclusion is the portion of an argument structure that is relevant to the user. In the case where all the solution components are represented as a ground propositional theory whose dependency graph has edges E, then:

[188] For another example, see Figures 8.3 to 8.5 in section 8.2.3.

$$\mathit{edges(answer)} \subseteq \mathit{edges(argument\ structure)} \subseteq$$
$$\mathit{edges(case\ model)} \subseteq \mathit{(edges(generic\ domain\ model) = E)}$$

where *edges(X)* denotes the edges of the dependency graph present in *X*..

Note the commonality between Clancey's and Breuker's view: expert system inference is the extraction of some subset theory from a super theory. Clancey offers a two-layered extraction process (QM to SSM) while Breuker offers a more detailed four-layered view.

Returning now to HT4/`Core`, we note that this algorithm also extracts sub-models from super-models. The extracted models are relevant to a particular task, defined as a pair of *<Inputs, Outputs>*, and is guaranteed to be consistent. For example, the QCM model compiler would convert Figure 7.8.i into Figure 7.8.ii. Figure 7.8.ii represents the superset of all explanations possible within 7.8.i (i.e. it is an explicit ground version of Clancey's QM).

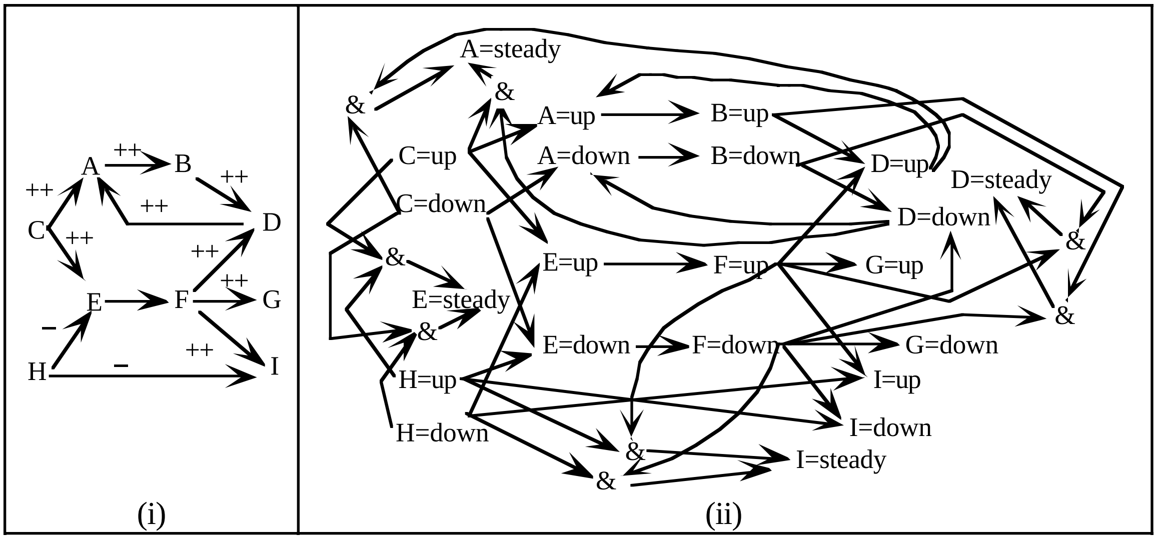


Figure 7.8. *Converting between (i) the model supplied by the expert and (ii) the and-or graph generated by (e.g.) the QCM model-compiler. For example,* D=up & C=down → A=steady.

Recall the example used to introduce the `Core` algorithm[190]: *M* = the model of Figure 7.8.i; *Inputs = {C=up, H=up}*; *Outputs = {B=up, D=up, G=up, I=down}*; and *I* = invariants = no vertex in *M* can be in two states simultaneously. Figure 7.9 shows all the different ways that each member of *Outputs* can be explained.

```
P[1] = H=up→  E=down→ F=down→  I=down
P[2] = H=up→  I=down
P[3] = C=up→  E=up→  F=up→  G=up
P[4] = C=up→  A=up→  B=up→  D=up
P[5] = C=up→  E=up→  F=up→  D=up
P[6] = C=up→  A=up→  B=up
P[7] = C=up→  E=up→  F=up→  D=up→  A=up→  B=up
```

Figure 7.9. *Possible explanations.*

[190]

The proofs of Figure 7.9 makes certain assumptions (i.e. uses literals that are not from *Facts = Inputs ∪ Outputs*). When we sort these assumptions into the base controversial assumptions, we get the worlds shown in Table 7.3.

World #	*MinEnv*	*Exclusions*	*Contains*	*Explains*
1	`{}`	`E=up, E=down`	`P[2], P[4], P[6]`	`I=down, D=up,B=up`
2	`E=up`	`E=down`	`P[2], P[3], P[4], P[5], P[6], P[7]`	`I=down,G=up, D=up,B=up`
3	`E=down`	`E=up`	`P[1], P[2], P[4], P[6]`	`I=down, D=up, B=up`

Table 7.3: *The worlds of Figure 7.7 relevant to a certain task = <Inputs, Outputs>.*

That is, HT4/`Core` has proposed three alternative solutions, or worlds (see Figure 7.10). Note that each solution is some subset of the set of all solutions (shown in Figure 7.8.ii). Breuker and Clancey would call these worlds "SSMs" or "case models".

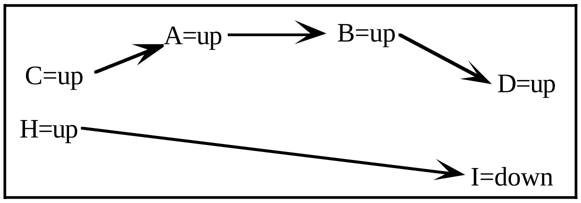


Figure 7.10.i. World 1*: a solution proposed by HT4/*`Core`*. World 1 avoids all controversy; i.e. uses no base controversial assumptions.*

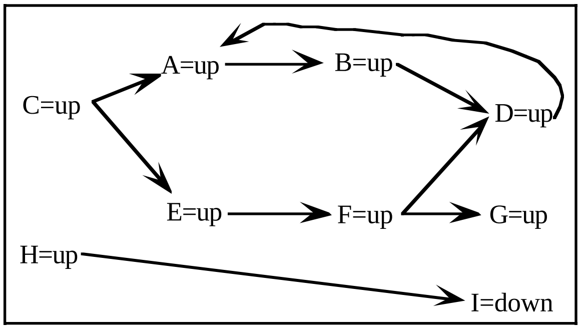


Figure 7.10.ii. World 2*: an alternative solution proposed by HT4/*`Core`*. In this solution, the algorithm has assumed that* C *is the dominant influence on* E.

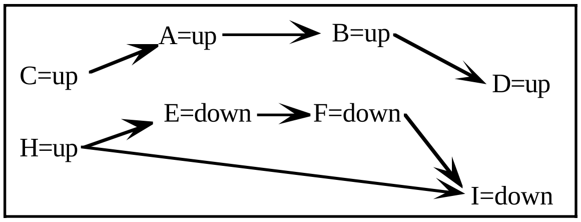


Figure 7.10.iii. World 3*: the third solution proposed by HT4/*`Core`*. In this solution, the algorithm has declared that* H *is the dominant influence on* E.

Queries can be executed over each world to generate Breuker's argument structures or conclusions. For example, in *World 3*, an argument structure for the conclusion *D=up* would be:

$$C{=}up \rightarrow A{=}up \rightarrow B{=}up \rightarrow D{=}up.$$

HT4/`Core` directly operationalises the subset extraction process that Breuker and Clancey argue is at the core of expert systems inference. Further, HT4/`Core` provides a uniform structure for processing of many of problem solving types listed by Breuker[191].

[191] See the above discussions in this chapter on using HT4/`Core` for planning, design, prediction, monitoring and

Such uniformity simplifies the construction of interfaces between the inputs and outputs of different problem solving types. Breuker argues that such interfacing is essential since most problem solving types are used in combination to perform some task.

HT4/`Core` also simplifies an important distinction proposed by Clancey:

> *A broad view of how a solution is computed suggests that there are two basic* problem*-solving methods used by expert systems:* heuristic classification *and* construction *[37].*

By *classification*, Clancey means that the inference engine can report any proof it can find between inputs and outputs across the knowledge base. Clancey-style *classification* is a single operator that quickly selects proof trees from a set of fixed alternatives. In *heuristic classification* [37], this pathway would include:

- Inference to an abstracted description of the problem at hand;
- A match of this problem to an abstracted solution;
- A inference that specialises the abstracted solution to a solution relevant to the current problem[192].

Clancey-style *constructive* inferences generate multiple proofs, but each must be assessed with respect to the other proofs. For example, combinations of processes that cancel or combine their influences must be accounted for; i.e. $A \vee B$ may have different consequences to $A \wedge B$. Further, certain literals in different proofs may be mutually exclusive. The constructed SSM must be built with care in order to take into account these interactions. Multiple, mutually exclusive, SSMs may be possible and these must be managed separately. Extra architecture is required to handle conflicts and dependencies within the SSM.

> *The essential difference between constructive and classification are the need for some 'data structure' to post the assemble solution and operators for proposing and reasoning about solution fragments [37].*

By *solution fragments* and *data structures*, is Clancey is referring to the additional architecture required for assessing competing or interaction sub-solutions. Clancey is describing the HT4/`Core` process, but using different terminology. Here is Clancey's description:

> *When there are multiple causal links for classifying data - multiple explanations- inference must be controlled to avoid redundancy, namely multiple explanations when one would have been sufficient. The aim is to produce a coherent model that is complete (accounting for the most data) and simple (involving one fault process) [37].*

Expressed in terms of HT4/`Core`, Clancey is saying that he prefers worlds which are internally consistent, have maximum cover, and use the minimum number of *Inputs*[193].

[192] For more on heuristic classification, see section 8.2.4.

[193] [illegible]

We find that Clancey-style classification is simple deduction (albeit across a model which may contain specialisation and generalisation links) while Clancey-style constructive inference can be *accurately* and *usefully* described as abduction:

- *Accurately*: The construction of an SSM from a QM that satisfies some task (specified by known inputs and desired outputs) in the presence of invariants is exactly the HT4/`Core` algorithm described above. Both proposals can generate multiple worlds/SSMs. Note that `Core` worlds are guaranteed to satisfy Clancey's consistency requirement. Further, just as deduction is a special case of abduction[194], Clancey-style classification is a special case of Clancey-style construction ([37], p332). As Clancey says "all programs construct inference paths", but construction paths must be constructed more carefully than classification paths.
- *Usefully*: The HT4/`Core` proposal is more general than Clancey's since it makes explicit certain assumptions which are only tacit in Clancey's approach. For example, he assumes that the best world uses the fewest number of *Inputs* ([37], p 331). We have shown previously that this is not universally true[195]. Further, Clancey divides all KBS inferencing into classification or construction . By mapping these two processes into deduction and abduction respectively, we can find the missing component of Clancey's framework. Abduction and deduction are two components of a triad of possible inference procedures. The third component is *induction* which is neither classification or construction. Induction (i.e. machine learning techniques) are KBS tasks, but fall outside of Clancey's framework.

7.4. Conclusion

We have argued `Core` can be used for:

- KBS validation and verification;
- Abduction and deduction;
- Consistency-based diagnosis;
- Set-coverage diagnosis;
- Frame-based reasoning[196];
- Prediction;
- Explanation;
- Classification;
- Planning & Monitoring;
- Causal/ Qualitative reasoning;
- Design;

[194] In the case where models contain no invariants, an abductive inference engine would generate a single world. The same world could be discovered using a simpler forward-chaining deductive algorithm.

[195] See the discussion around Figure 7.2, section 7.1.1.2.

[196]

- Visual pattern recognition;
- Single-user model-based decision support systems.

Further, `Core` would be useful in the following domains:

- Group decision support systems;
- Expert critiquing systems.

Some evidence exists that `Core` would also be useful for:

- Case-based reasoning;
- Natural language processing;
- Certain types of financial reasoning;
- Learning.

More generally, we have argued that `Core` directly operationalises the model extraction process which Clancey and Breuker argue is at the core of expert system inference.

We believe our case to be strong for models of size less than the limits to testing found in the last chapter (i.e. $|V| \leq 850$; $|E|/|V| \leq 7$). For models within those limits, we have shown that the exhaustive world generation of HT4/`Core` is practical. Models larger than those limits can still be process by HT4/`Core`. However, heuristic culling of the search space would be required. We note that general techniques for culling the search space require a representation similar to the and-or graph used by `Core`. Therefore, `Core` could still be used for models with $|V| > 850$ and $|E|/|V| > 7$. However, in poorly-measured domains (and most KBS problems are in poorly measured domains[197]) we caution against this since models that can't be tested should be used very cautiously or not at all.

We noted in our introduction that our goal of generic principles for testing was somewhat ambitious since a generic testing module implies a generic execution engine. The above list suggests that our current generic testing module satisfies our requirement for generic execution.

This leads to an interesting conjecture. From a pragmatic engineering viewpoint, it is useful to replace multiple single-purpose mechanisms with one general-purpose mechanism, particularly when (i) the services offered by the general mechanism is a superset of the services offered by the multiple mechanisms; and (ii) the general mechanism is simpler than the single-purpose mechanisms. Current KBS inference engines do not support generalised test yet generalised test supports numerous inference tasks (e.g. the above list). That is, we could replace our inference modules with generalised test modules.

Summary: Our generalised test does not merely *augment* existing KBS architectures, it could *replace* them.

[197]

Rail travel at high speed is not possible, because passengers, unable to breathe, would die of asphyxia: Dr. Dionsys Lardner (1793-1865).

The solution to the problem changes the problem: Peer's Law. *The only good answers are those that destroy the question:* Susan Sontag. *Legend: a lie that has attained the dignity of age:* H.L. Mencken. *People will accept your idea much more readily if you tell them Benjamin Franklin said it first:* Lomin's Law. *People will believe anything if you whisper it:* Anonymous. *If a little knowledge is dangerous, where is the man (sic) who has so much as to be out of danger?* T.H. Huxley

8. Radical

Previously, we have (i) motivated and generalised the processing of testing a KBS; (ii) described how to customise testing for different domains; (iii) demonstrated the practicality of our approach for models at least as big as those seen in contemporary KA practice; (iv) discussed the utility of the `Core` algorithm to a range of KBS tasks. We argued that we could replace numerous KBs tasks with `Core`.

In the this chapter, we will strengthen that claim: not only *could* we replace current KBS architectures, we *should* do so. This is a radical argument since current KA practice is to abstract away from low-level inference routines such as `Core`.

This chapter is structured as follows. The radical case for testing is made in sections 8.1 and 8.3. Section 8.2 is a digression that introduces *knowledge level modelling*, the dominant paradigm in contemporary knowledge acquisition. Knowledge level modelling will be contrasted with abduction/`Core` in section 8.3.

8.1. Need for Testing

We have argued above that testing is an essential and on-going process for KBS[198]. As it is usually done, KBS testing is a labour intensive process. For example, the MYCIN and CASNET validation studies [258, 269] required the convening of special committees to assess their models. Given the on-going and continual nature of testing, such specially-convened committees are too cumbersome. We require automated support for testing. Further, this testing process needs to be able to execute in the poorly measured domains processed by modern KBS. In poorly-measured domains, a test engine must make assumptions and maintain those assumptions in consistent worlds. Formally, this is abduction and can be implemented by systems such as `Core`.

In the previous chapter, we have argued that the generation of sets of consistent inferences is an inference process applicable to many domains. We have made the case

[198]

on pragmatic engineering grounds that it is possible to replace numerous KBS inference architectures with an abductive architecture. Such abductive architectures have the added advantage that they can serve as both inference engines and test engines for poorly-measured domains and indeterminate models. Therefore, we strongly recommend the reorganisation of KBS systems around abductive/`Core` principles.

8.2. Knowledge Level Modelling

In the next section, we contrast implementation techniques for implementing *knowledge-level* (hereafter, KL) inferencing. We distinguish two types of knowledge level modelling. KL_A and KL_B refer to (respectively) *first* and *second generation knowledge modelling tools*. KL_A is exemplified by the SOAR project [223] which instantiates Allen Newell's vision of the knowledge level [173, 174]. The KL_B paradigm is the union of a set of researchers who argue for a similar structure using different terminology; e.g. KADS [260], Chandrasekaran's generic tasks [32], Steels' components of expertise [246], Clancey's model construction operators [40], and the architecture of SPARK/BURN/FIREFIGHTER (SBF) [145]. Currently, KL_B is the dominant paradigm in the KA community.

8.2.1. Before-KL

Knowledge engineering in the 1970s was dominated by Feigenbaum expertise transfer principle. Knowledge engineers "mined the jewels in the expert's head" and transferred those "jewels" in symbolic form into a program ([78], p104). A commonly-applied technique for facilitating the transfer process was the identification of the constructs supposedly used within human cognition. Once identified, special-purpose interpreters could be built that could process some sentential form of those constructs. Figure 8.1 shows the standard architecture for expertise-transfer using such interpreters.

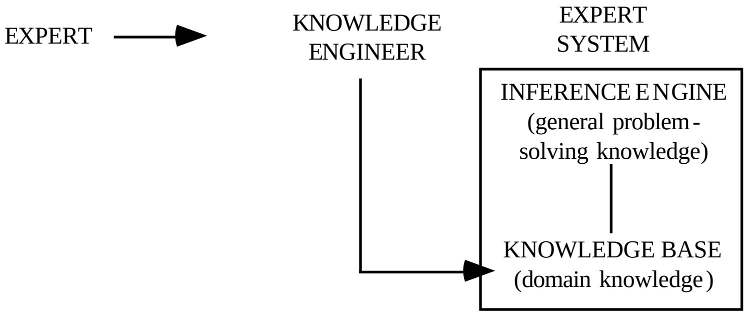

EXPERT
KNOWLEDGE ENGINEER
EXPERT SYSTEM
INFERENCE ENGINE (general problem-solving knowledge)
KNOWLEDGE BASE (domain knowledge)

Figure 8.1: *Standard expertise-transfer architecture; from [102] p130.*

Many styles of interpreters were developed (see [102] for a survey). Several of these were used to develop systems that demonstrated expert-level competency. Weiss & Kulikowski used causal networks in the CASNET system [258] to model disease processes. Duda, Gasching, & Hart [29, 72] modelled geological domains using an inference network of assertions in the PROSPECTOR system. Brachman and Levesque generalised the notion of a semantic network to inference over terms defined in an inheritance hierarchy and global rules defined in terms of first-order predicate calculus [16]. Newell & Simon's experiments with production-rules suggested that a library of rewrite rules acting over a working memory had some cognitive likelihood [128, 175]. This rule-based approach had several high-profile successes; e.g. the R1/XCON system [9, 146, 147, 243, 255] and MYCIN [269]. A sample MYCIN rule is shown in Figure 8.2.

```
DOMAIN RULE-513-A
IF   1) The infection is MENINGITIS.
     2) The type of infection is BACTERIAL.
     3) The patient has undergone SURGERY.
     4) The patient has undergone NEUROSURGERY.
     5) The NEUROSURGERY-TIME was less than 2
        months  ago.
     6) The patient received a VENTRICULAR-
        URETHRAL-SHUNT
THEN: There is evidence that the organism which
      might be causing the infection is E.COLI(.8)
      KLEBSIELLA-PNEUMOBIAE (.75).
```

Figure 8.2: *A MYCIN domain rule (from* [40]*).*

In verification studies, MYCIN, CASNET and PROSPECTOR demonstrated that they could reproduce expert-level performance in the their test domains[199], [200]. Some of these systems even survived outside of the research environment and entered routine use[201].

Despite early attempts to formalise the above process [247], expert system construction remains a somewhat hit-and-miss affair. By the end of the 1980s, it was recognised that our design concepts for knowledge-based systems (KBS) were incomplete [25].

8.2.2. Knowledge Level Modelling

An influential alternative design concept was Newell's *knowledge level* approach[202]. Newell's viewed human cognition as a search for appropriate *operators* that convert an agent's *current state* to a desired *goal state*. This approach can be viewed as a refinement of the classic AI position that search is a central problem in human intelligence [126]. Domain-specific knowledge was used to select the operators according the principle of rationality; i.e. *the agent will select an operator to perform next which, according to its knowledge, leads to the achievement of one of its goals*. Beneath the knowledge level is the *symbol level*. At the symbol level, programs consist of data structures, algorithms, etc. However, to a knowledge-level agent, these sub-

[199] Though Duda *et. al.* carefully distance themselves from certain exaggerated claims regarding the success of PROSPECTOR [73].

[200] Verification of R1/XCON is harder and its utility is usually demonstrated in operational terms; e.g. its sponsor believes in it sufficiently to use it and maintain it for the 14 years since it was first developed. See [9] for impressive performance measures.

[201] See [74] for a partial list as of 1983 and [25] for an updated list as of 1989.

[202] Newell stresses that while he popularised the idea of the knowledge level, he did not invent it. To him, the knowledge level paper was a simple extension of two existing ideas in computer science: layered architectures

cognitive symbol-level constructs are the tools used "sub-consciously" (as it were) as it performs its knowledge-level processing [173, 174]. Wielinga *et. al.* contrast the symbol/knowledge levels as follows:

> *One can take two different perspectives on modelling the expertise required from a system. A first perspective- one that is often taken in AI- is to focus on the computational techniques and the representation structures (e.g. rules, frames) that will provide the basis of the implemented system. A second perspective focuses on the behaviour that the system should display and on the types of knowledge that are involved in generating such behaviour, abstracting from the details of how the reasoning is actually realized (sic) in the implementation. These two perspectives correspond to the distinction Newell makes between respectively the symbol level and the knowledge level. [260]*

Newell's KL proposal was consistent with other research. A reverse engineering of existing expert systems suggested that above the level of rules, frames, forward/ backward chaining, etc., there exist repeated meta-level inference procedures [31, 32, 37]. In at least four applications, the "Digital school" found that these higher-level procedures were productivity tools (e.g. MOLE [117], RIME [255], SALT [142, 143], SBF [145]). In the MOLE application, for example, it was found that exploring eight specific questions could lead to major improvements in the knowledge base (e.g. when considering *X*: "what events would rule out *X*?"). Customised editors were written that directly manipulated these higher-level structures (e.g. the MOLE questions or the operator-selector search space of RIME). These editors included specialised compilers that could convert these higher-level constructs into executable symbol-level constructs.

The terminology of the RIME editor included Newell-style knowledge level constructs (e.g. local goal spaces and operators that need to be evaluated/selected/applied). Indeed, RIME was developed initially using Newell's SOAR system [222]. However KL_B researchers (such as the Digital school and the KADS community) argued that problem-solving behaviour that transcended a single state was also a knowledge level construct. In particular, the KADS ESPRIT project developed a KBS methodology based on libraries of known problem-solving techniques:

- KADS researchers argue that parameterised versions of these meta-level inference procedures (e.g. *classify, monitor, predict)* offer a structured approach to the knowledge acquisition endeavour.
- KA can initially be the hunt for a match between known expert behaviour and known meta-level inference procedures [260]. Once a match is found, then the meta-level pattern offers a rich description of the high-level processing loops as well as the data structures that the experts have to supply.

- Apart from the inference structures, KADS also offers a methodology for the whole life-cycle of a software project building an expert system[203].

Other researchers have processed KADS-style variants on KL_B. Some of these pre-date KADS or were developed simultaneously. All assume that high-level inference procedures are customisable knowledge and offer tools for representing that knowledge (e.g. Clancey's model construction operators, hereafter MCO [40]; as well as [28, 33, 145, 246]).

Meanwhile Newell, Rosenbloom, and Laird (amongst others) were implementing Newell's vision of the knowledge level in the SOAR project. SOAR combined Newell's earlier research on (i) state-space traversal with (ii) his work on production systems and, eventually, (iii) his knowledge-level proposal:

- The primitives of the SOAR rule-based language explicitly represent Newell's model of human cognition as a search for appropriate operators that convert an agent's current state to a desired goal state [223].
- Like our abductive approach, minor manipulations of SOAR's operator selection strategies (which are controlled by rules) is all that is required to fundamentally change the inferencing [127].
- Initially, SOAR was not a knowledge-level modelling tool. Despite Newell's public endorsement of KL_B approaches[204], SOAR did not offer language primitives for representing meta-state structures.
- Subsequent work added *a problem-space computational model* (PSCM) on top of the basic SOAR architecture. Programming a PSCM involves the consideration of multiple, nested problem spaces. Whenever a "don't know what to do" state is reached in SOAR, an *impasse* is declared. Impasses automatically fork a new problem-space to resolve the problem. Impasses can be recursively generated within impasses. The PSCM recognises that the explicit representation of these nested impasses is a significant programming construct. A Lisp pre-processor converted constructs in the PSCM description language TAQL into SOAR productions [176, 268].
- Newell argued that the PSCM was the bridge between SOAR and true knowledge-level modelling [174, 176]. Note carefully, however, the difference between KL_B and KL_A. Unlike KL_B, in PSCM-SOAR KL_A, high-level inference procedures such as *classify*, *monitor* or *predict* are not explicitly represented. The observation that a knowledge base is performing (e.g.)

[203] For more on KADS, see the "related work" section of [260] for an historical overview of KADS and a comparison of it with other approaches. For a gentle introduction to KADS, see the short tutorial in [134] followed by [260]. For a detailed overview of the technique, see [251]. For a critique of KADS, read on.

[204] Newell's name appears on top of Clancey's classic KL_B *heuristic classification* paper [37] in the

classification is a user-interpretation of a lower-level inference (operator selection over a state space traversal) [268].

HT4/`Core` is closer to KL$_A$ than KL$_B$. Like PSCM-SOAR, HT4/`Core` offers few organisational principles above some low-level inference (except the notion of domain-specific data and model compilers[205]). Like PSCM-SOAR, HT4/`Core` does not explicitly model (e.g.) *classify*, *monitor* or *predict* at the inference level. As we have seen, these inference procedures are implemented by small pre-processors and post-processors to the basic `Core` inference[206]. However, HT4/`Core` differs from PSCM-SOAR in several basic respects:

- PSCM-SOAR executes over an implicit and-or graph while we prefer to execute over an explicit and-or graph[207]. Efficiency is a non-trivial issue in generalised test. Building and caching the search space prior to inferencing was one or our techniques for taming complexity[208].
- Given a vertex with *N* out edges (or, in SOAR-speak, a state space with *N* associated operators), HT4/`Core` assesses the utility of each edge using a deferred global analysis. SOAR must make its operator assessment at the local level. SOAR's run-time selective generation of the and-or graph has efficiency advantages since it culls unacceptable alternatives as they are first encountered. Our approach has the potential to be slower, but the explicit representation of all alternatives permits allows for global assessment criteria (e.g. $BEST_4$). Experiments with adding abductive inference to SOAR relied on an interface to an external abductive theorem prover. In Steier's CYPRESS-SOAR/ RAINBOW system, SOAR production rules modelled control decisions, while the RAINBOW abductive inference engine[209] generated possible designs [248].

8.2.3. KL$_B$: An Example

As an example of a KL$_B$ analysis, recall the rule in Figure 8.2. An MCO analysis of this rule is that it includes three general inference procedures, plus one piece of domain-specific knowledge. Figures 8.3 and 8.4 shrink the rule by extracting two of the inference procedures[210].

[205] See chapter 5.

[206] See chapter 7.

[207] Recall that an explicit and-or graph does not grow at runtime while an implicit and-or graph can add new edges and vertices during inference.

[208] Note that this is only practical for finite theories. Hence HT4/`Core`'s current emphasis on propositional models.

[209] Steier calls abduction "antecedent derivation".

[210]

```
% domain knowledge
subtype(meningitis,
        bacterialMeningitis).
subtype(bacterialMeningitis,
        eColi).
subtype(bacterialMeningitis,
        klebsiellaPneumoniae).
differential(Hypothesis).

% inference procedure
task(exploreAndRefine, Hypothesis,
     doBefore(
     [task(pursueHypothesis(Hypothesis)),
      task(pursueHypothesis(Child)]))
     :-
     differential(Hypothesis),
     subtype(Hypothesis,Child).
```

(i)

```
DOMAIN RULE-513-B
IF 3) The patient has
      undergone SURGERY.
   4) The patient has
    undergone NEUROSURGERY.
   5) The NEUROSURGERY-TIME
      was less than 2
      months  ago.
   6) The patient received
      a VENTRICULAR-URETHE-
      RAL-SHUNT
THEN: There is evidence
   that the organism which
   might be causing the
   infection is E.COLI(.8)
   KLEBSIELLA-PNEUMOBIAE
   (.75).
```

(ii)

Figure 8.3: *Rule-513-B = Rule-513-A with explicit and separate representations of* subtype*s, names of* differential *hypotheses, and the* exploreAndRefine *inference procedure (i.e. explore super-types before sub-types). Adapted from [40]*[211]*. Note that* exploreAndRefine *removes the need for rule conditions 1 and 2.*

```
% domain knowledge
subsumes(surgery, neuroSurgery).
subsumes(neuroSurgery,recentNeuroSurgery).
subsumes(recentNeuroSurgery,
         ventricularUreteralShunt).
value(recentNeuroSurgery,true) :-
     value(neuroSurgery,time(Mons)),
     Mons < 2.

% inference procedure
task(findOut,Finding, conclude(Cntx,
             Parent, yesTally, -1000)) :-
     context(Cntx),
     subsume(Parent,Finding),
     not(Cntx = Parent).
```

(i)

```
DOMAIN RULE-513-C
IF    The patient received
      a VENTRICULAR-URETHE-
      RAL-SHUNT
THEN: There is evidence
   that the organism which
   might be causing the
   infection is E.COLI(.3)
   KLEBSIELLA-PNEUMOBIAE
   (.3).
```

(ii)

Figure 8.4: *Rule-513-C = Rule-513-B with explicit and separate representation of* subsume*s and the* findOut *inference procedure (i.e. if a desired finding is a subtype of a class of findings and that class of findings in not present in this case, then conclude the desired findng has a negative infinity certainty factor; i.e. is not present). Adapted from [40]. Note that* findOut *removes the need for all rule conditions except the last one.*

Figure 8.5 shows the remaining inference procedure. It is a conflict resolution rule that selects causal rules before mere circumstantial rules like `513-B`.

[211] Clancey expresses his knowledge bases in a LISP syntax. We use Prolog here since we have already used Pascal and Prolog and believe that the use of yet another language in this document for abstract specifications

```
% domain knowledge
enablingCause(bacterialMeningitis, exposure).
circumstantialCause(bacterialMeningitis,neuroSurgery).

% inference procedure
task(testHypothesis,Hypothesis,
     doBefore([task(applyRules(Rules1)),
               task(applyRules(Rules2)))
     :-
     enablingCause(         Hypothesis, Finding1),
     evidence(              Hypothesis, Finding1, Rules1),
     circumstantialEvidence(Hypothesis, Finding2),
     evidence(              Hypothesis, Finding2, Rules2).
```

Figure 8.5. *Explicit representation of the* testHypotheiss *inference procedure (i.e. test causal rules before testing circumstantial rules). Adapted from [40].*

Some evidence suggests that encoding knowledge bases (KBs) using these inference procedures gives the inference engine enough control knowledge to improve its performance. Clancey reports that after repeating the above analysis for the 176 MYCIN domain rules, 80% of them reduced down to single condition rules such as Figure 8.4.ii. When executed, the meta-control offered by rules such as Figure 8.5 removed all uncontrolled backward chaining [40]. This result has not been repeated elsewhere.

In MCO, Clancey argues eloquently that knowledge of inference procedures is still knowledge and should be expressed in a customisable form. The explicit and separate declarative representation of inference procedure knowledge allows a domain expert/knowledge engineer to record their processing knowledge (see Figure 8.6).

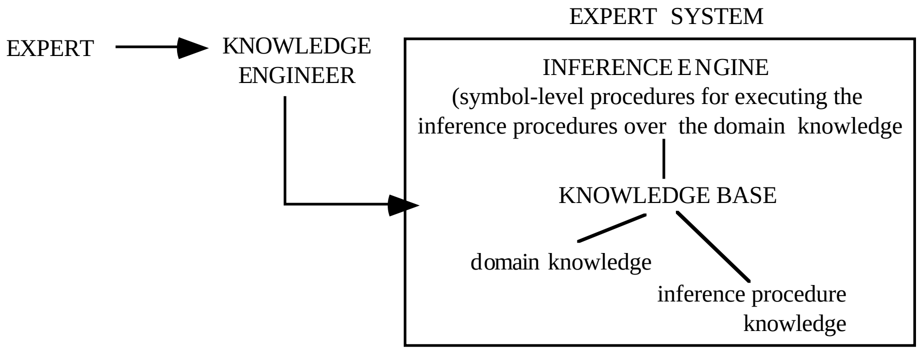

EXPERT SYSTEM
EXPERT
KNOWLEDGE ENGINEER
INFERENCE ENGINE
(symbol-level procedures for executing the inference procedures over the domain knowledge
KNOWLEDGE BASE
domain knowledge
inference procedure knowledge

Figure 8.6: *Clancey's proposed extension to the architecture of Figure 8.1 from MCO [40]. Inference procedure knowledge is still knowledge and should be maintained in a customisable form within the knowledge base .*

8.2.4. Knowledge Engineering with KL$_B$

Wielinga *et. al.* argue that the inference procedures discovered from a KL$_B$ analysis form the basis of re-usable problem-solving library that can be applied to other domains [260]. Within the KL$_B$ inference procedures, there often exist sub-routines used in other procedures (e.g. *select*, *decompose*, *match*, *evaluate*, etc) [32, 40, 145, 260]. For example, the KADS description of the *heuristic classification*, *verification* and *correlation* inference procedures are shown in Figures 8.7, 8.8, and 8.9. They share some processing modules (with some variations); e.g. *match* and *select*.

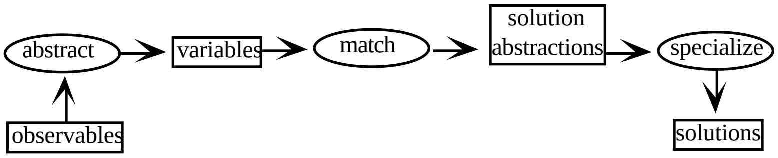


Figure 8.7: *Heuristic classification: the original KL_B construct, circa 1985. From [251]. Squares denote data types and ellipses denote processing modules. Arrows denote data flow. In between observations and solutions is an abstraction process that matches an abstracted description of the problem to an abstracted description of possible solutions. Clancey's original KL discovery was that a range of expert systems could be described as heuristic classifiers [37] .*

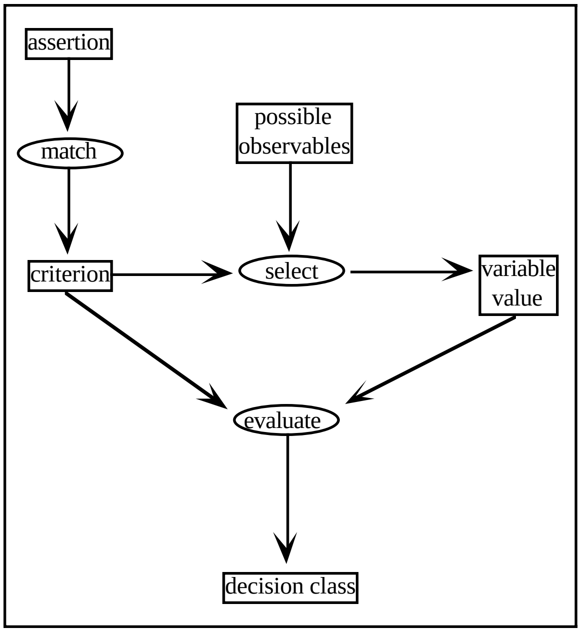


Figure 8.8: *A KADS description of the* verification *inference procedure; i.e. testing if a system is consistent with (at least some of) the actual values of the observables of the system.* Match *inputs an* assertion *of some observation and outputs a test* criterion *which will perform the verification. The* criterion *can optionally select other* possible observations. *Once all the required observations are collected, the criterion is* evaluate*d to generate the* decision class. *From* [251].

Software libraries that support these processing modules can be used to decrease development time. Marques *et al* compare development times with and without such a software library (i.e. SBF): 200 days (without) to 20 days (with) [145].

It is claimed that explanations generated from tasks are more insightful for the domain expert and the software designer than a (e.g.) low level trace of rule firings [36, 260]. Such explanations, it is said, can be used to:

- Identify and fix knowledge anomalies during KA.
- Offer justifications of the delivered system's conclusions for the end-user.

KL_B offers a succinct vocabulary for describing, summarising, and comparing expert systems. For example, a KADS-level description of heuristic classification fills 10 pages [5] while Clancey's original description is somewhat more verbose (61 pages) and not as precise [37]. Practitioners find this retrospective second-glance at their systems useful for developing more generalised architectures for future work [134]. Expert systems theoreticians have used KL_B to assess and clarify the essential features and differences of applications [228]. Lastly, knowledge engineering novices can use a KL_B analysis of classic expert systems to quickly review successful techniques.

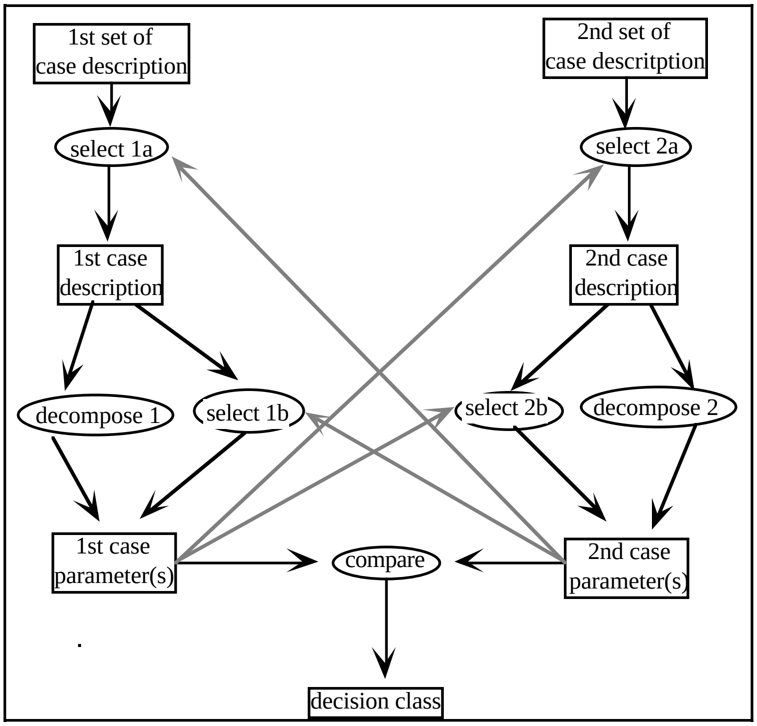


Figure 8.9: *A KADS description of the* correlation *inference procedure; i.e. compare two systems (e.g. an actual case vs some reference case) and generate a* decision class *that is a report of the comparison.*

Select Xa *extracts one case from a set of cases.* Select Xb *extracts one parameter of the selected case. Selections may be made on the basis of other case parameters (hence the feedback arrows shown in gray).* Decompose *is an alternative to* select Xb *and teases apart some case using (e.g.) part-of knowledge. From* [251].

8.3. Against KL_B Analysis

8.3.1. General Criticisms

One bottleneck in KL_B is the mapping between problem description and meta-level inference patterns. In the SBF work, for example, the application domains were all pre-selected and the mappings between a library of application types and the library of mechanisms was hand-coded. Techniques to assist/automate this process have yet to evolve.

One technique we have found useful for expert system development is prototyping [157]. While (e.g.) KADS is officially a prototyping approach [260], the overheads associated with documenting the KADS approach may introduce an organisational inertia inhibiting the prototyping process ("you mean you want we to re-write the design document... again?"). For example, after one KADS training course (conducted by specially-imported consultants) for an experienced Australian knowledge engineering group, that group concluded that the overheads of KADS made it unsuitable for projects less than 6 months long (i.e. the majority of their applications).

KL_B is an active research area and mature tools for this technique are still evolving. Even simple documentation tools are lacking. Prior to the publication of [251] there existed no central site, or even a Internet FAQ list of commonly used models.

Any reading of the KL_B literature suggests that one requires a high level of skill to use this approach. Allemang notes that "generic task analysis" (the Chandrasekaran school of KL_B) is difficult and requires a knowledge engineer [6]. Marques *et al* similarly note that experts are enmeshed in the details of using their skills and find it difficult to understand what they are doing in more abstract terms [145]. Aben provides lists of authors who express discontent with imprecisions in the KADS formalism [1].

For more specific criticisms of KL_B , we turn to the explanation, knowledge acquisition, and knowledge maintenance literature.

8.3.1.1. Explanation

Wielinga *et. al.* argue that one of the advantages of KADS was its ability to explain the inner workings of an expert system [260]. Clancey's *Heuristic Classification* paper [37] is an impressive reverse engineering of numerous expert systems in terms of his heuristic classification technique. The reader is left with a strong impression that heuristic classification *explains* the inner-workings of the surveyed expert systems. However, this is not to say that KL_B is a general explanation technique. An approach that can offer satisfying explanations for a small and select audience (i.e. knowledge engineers) may not generalise to a wider audience.

The problem of explanation is not solved merely by offering a trace of the system's traversal over a task description (e.g. a KADS interpretation model):

- Modern explanation research views explanations as a user-specific and goal-specific construct. From a range of possible inferences, some subset is selected that meets some understandability criteria for different users and different goals[212]. That is, a model that is good for explanatory purposes contains some degree of indeterminacy (can generate > 1 behaviours).
- Leake argues convincingly that a cache of prior explanations and an active user model are essential components of a good explanation module [130].

User-profiles, indeterminate models, and case libraries are not issues addressed in current KL_B approaches. Therefore, in their current form, KL_B is not a good generalised explanation tool[213].

8.3.1.2. Knowledge Acquisition

A premise in KL_B is that the higher-level abstractions of the knowledge modelling approach is a good thing. Soloway *et al.* comment:

[212] Recall section 7.1.5.

[213] Previously, we have distinguished between pure performance KBS systems (which only have to solve some problem) and explanations systems (which have to solve a problem and explain how they do it [159]. At that time, we believed that the higher-level abstractions of KL_B were required for explanation, but were not required for performance systems. Since that time we have reversed our thinking. We now doubt the utility of KL_B for general-purpose explanations. Further, we now believe that a performance system based on an

> *While hard numbers are few and far between, the overwhelming sense of the software engineering community is that the use of higher-order languages has had a positive impart on maintenance. Thus, on these grounds alone, it is quite reasonable to predict that XCON-in-RIME , written in RIME, a higher-level language, should be significantly easier to maintain that XCON, written in a lower-level language ([243], p827).*

We call this assumption that abstraction is a good thing the *utility-of-abstraction assumption*.

We note that if the utility-of-abstraction assumption was correct, then we would should see some superior performance of higher abstraction tools (KL_B) over lower-level abstraction tools (e.g. the KL_A tool SOAR) when they are used for the same purpose. In terms of the current knowledge acquisition literature, this has not been the case.

Initially proposed in 1990, the Sisyphus[214] project aimed at creating a library of repeatable knowledge engineering tasks. These tasks were to serve as benchmarks to compare different KA approaches. Sisyphus-2 was a medium-sized task (elevator configuration) and a prior to solution to the Sisyphus-2 problem was known in the literature from 1986 (SALT [142, 143]). This prior solution was used to generate a precise specification [267] which was the starting point for the Sisyphus-2 groups. The problem was attempted by six KL_B techniques (DESIRE [19], VITAL [170], Karl [193], Protege-II [224], CommonKADS [227]), one KL_A technique (PSCM-SOAR [266]) and one hybrid first/second generation approach (DIDS [226]).

At the Sisyphus-2 workshop, we collected development times for the different approaches (either from the developers or from the publications). Two conclusions were reached:

1) Data collection techniques for Sisyphus-2 was poorly managed. Most Sisyphus-2 were less-than-rigorous in documenting their development times (exceptions: the DIDS & PSCM-SOAR groups). We have made our case to the Sisyphus-2 community that data collection should be more rigorous in future.
2) Point #1 notwithstanding, no evidence could be found of productivity benefits of KL_B over KL_A in the Sisyphus-2 results. None of the times we collected suggested that any the groups out-performed the original SALT solution. That is, despite years of intensive research since SALT (1986), it is not clear that the KL_B approach offers any additional leverage. Indeed the "old-fashioned" KL_A solution (developed using SOAR) was developed in times comparable to SALT and faster than many of the other approaches[215].

Opponents of point #2 could argue that this sample size was too small and too inaccurately measured to yield meaningful results. Further, they could argue that the

[214] Sisyphus was doomed for eternity to roll the same rock up the same hill again and again and again...

[215]

Soloway quote (above) stressed the utility of abstraction for the purposes of maintenance, and not just development work such as Sisyphus-2.

In reply, we would concede the sample size problem but would add that we can't find alternative evidence justifying the current enthusiasm in KL_B. We list below a summary of the achievements of knowledge-level modelling. Note that this is not an overly-impressive list[216].

- Clancey's heuristic classification offered a unified retrospective view on numerous, seemingly different, expert systems. Similar (but smaller) studies by Linster & Musen [134], Akkermans *et. al.* [5] suggest that KL_B can retrospectively clarify design issues. However, the more important case that KL_B clarifies *initial* designs is yet to be proven.
- Clancey's MCO analysis of MYCIN removed all search from the domain rules. To our knowledge, this is the only published example of this effect.
- The Digital school of knowledge engineering reports productivity gains from second-generation knowledge modelling. The SBF results are particularly impressive (over 90% reduction in development time). However, recall from the above discussion the hard-wired nature of the links between the SBF problem space and the SBF solution space. Significantly, the SBF group did not offer solutions to Sisyphus-1 or Sisyphus-2. This is surprising since, given the development times reported in the SBF experiments, we would have thought that they could have built a Sisyphus-2 solution in a less than a week. Perhaps, the SBF technique is not a general principle.

8.3.1.3. Knowledge Maintenance

As to the issue of maintenance, recall that the utility-of-abstraction assumption was based on little empirical evidence (in Solloway's words: "hard numbers are few and far between..."). We know of only two data points:

- The RIME KL_B editor is used to maintain XCON, a 5000+ rule expert system[217], of which approximately 50% changes each year [243]. The system is successful enough for its sponsors to continue the project for several years. However, beyond that qualitative statement, little empirical evidence exists for the utility of RIME (but see [243] for some informal evaluations).
- PIERS was built using Compton's ripple-down-rules approach RDR[218]. PIERS's expertise covers 20% of the biochemical tests performed at the hospital. PIERS

[216] This list was presented at the Banff '94 AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Participants were asked to augment the list with any missing results. Over the week-long period of the workshop, no additions were offered.

[217] The precise size changes as the products it services change.

[218]

> processes 500 cases per day at 95% accuracy, contains 2037 rules, and is one of the largest expert systems in routine use in the world today. PIERS can be described as an almost zero-level abstraction program. After a simple pre-processor turns numbers such as *X=3.37* into statements of the form *X is low*, the system uses no further abstractions. Despite this lack of an abstraction mechanism, development was simple. Minimal preliminary analysis was required. After the database connections were made (using standard software engineering techniques), experts just considered the cases presented on a particular day and told PIERS what to say for each such case. Maintenance time is constant (2-6 new rules per day) and very simple (a total of a few minutes each day).

The PIERS conclusion is that, given certain environment support (i.e. an RDR structured patching environment), extensive preliminary analysis and high-level abstractions are an *optional* part of the software development life-cycle. After some initial minimal analysis, the system goes live and is patched in the context of its errors. Global re-organisations are forbidden. Like Soloway, our pre-experimental intuition was that such global reorganisations and high-level abstractions were an essential precondition for good software. This did not prove to be the case.

Summary: The higher-level abstractions offered by KL_B are not necessarily useful for knowledge maintenance. The utility-of-abstraction assumption has yet to be proven (and PIERS represents one major counter-example).

8.3.2. Core-Specific Criticisms

We argue that a abductive/`Core`-based architecture is better than the KL_B approach. Not only can `Core` supply the required inferences (see the list in Figure 8.10), but it can also implement generalised test.

```
Knowledge level inference procedures:
   1.  Causal/ qualitative reasoning;
   2.  Classification;
   3.  Design;
   4.  Consistency-based diagnosis;
   5.  Set-coverage diagnosis;
   6.  Explanation;
   7.  Monitoring;
   8.  Planning;
   9.  Prediction;
   10. Validation;
   11. Visual pattern recognition.

Symbol level inference procedures:
   1. Abduction;
   2. Deduction;
   3. Frame-based reasoning;
   4. Verification.

Composite:
   1. Single-user model-based DSS.
```

Figure 8.10: *Chapter seven made a strong case that* `Core` *can support eleven knowledge-level inference procedures, four symbol-level inference procedures, and one composite procedure. A weaker case was also made that* `Core` *is useful for group decision support systems; expert critiquing systems; case-based reasoning; natural language processing; certain types of financial reasoning; and learning. Further, a general theoretical case was made that* `Core` *directly operationalises the model extraction process which is at the core of all expert systems inference.*

Test is an essential part of KBS development[219]. Standard KL_B does not model the and-or graph of dependencies internal to the inference procedure sub-routines. The internals of these sub-routines are symbol-level constructs and hence out-of-bounds for KL_B. KL_B tools like KADS stress their implementation independent nature. No restriction is placed on the implementation tools. Hence, there is no guarantee that the implementation tool can generate the and-or graphs required for `Core`. Therefore:

- KL_B is not a candidate for generalised test.
- `Core` should replace the inference modules of KL_B since it can deliver inference as well as validation.

Further, we find that our abduction-based taxonomies of inference procedures are simpler than those offered by KL_B community. Figure 8.11 shows inference procedures from two different KADS researchers.

```
Transformation
   …
Synthesis
   Design
   …

Analysis
   Identification
      Diagnosis
         Systematic diagnosis
         Fault-model diagnosis
            Heuristic classification
            Cover & differentiate
      Monitoring
      …
   Prediction
```

(i) KADS: 1992 [260]

```
Modification
   …
Synthesis
   Design
   Planning
   …
Analysis
   Identification
      Diagnosis
         Single Model Diagnosis
            Systematic Diagnosis
               Localisation
               Causal Tracing
         Multiple Model Diagnosis
            Mixed Mode Diagnosis
      Verification
      Correlation
         Assessment
      Monitoring
      Classification
         Simple Class.
         Heuristic Class.
         …
   Prediction
      …
```

(ii) KADS: 1993 [251]

Figure 8.11: *Partial taxonomies of KL_B inference procedures. The term* transformation *(in Figure (i)) is analogous to* modification *(in Figure (ii)).*

We approve of some of the clusterings in the hierarchy of Figure 8.11. For example, the authors of Figure 8.11.ii argue that *localisation* and *causal tracing* are basically the same process (*systematic diagnosis*, see Figure 8.12), except the former using *part-of* knowledge while the latter uses *causal* knowledge. In terms of our framework, both would execute over the same and-or graph but the user's interpretation of the edges differs. However, we doubt (e.g.) the separation of *diagnosis* from *monitoring* (or worse, the large separation of *design* from *prediction*) since we have seen above that they are intimately connected.

[219]

To be fair, the authors of Figure 8.11 do discuss the connection between the various inference procedures. For example, the authors of Figure 8.11.ii note that (i) *heuristic classification* could be used for diagnosis; or (ii) that *scheduling*, *planning*, and *configuration* are actually the same problem, divided on two dimensions ("goal states known or not" and ""temporal factors considered or not"[220]). Also, the authors of Figure 8.11.i acknowledge certain similarities between the inference procedures. For example, Figure 8.14 identifies a common sub-graph of the inference procedures *systematic diagnosis* (Figure 8.12) and *monitoring* (Figure 8.13).

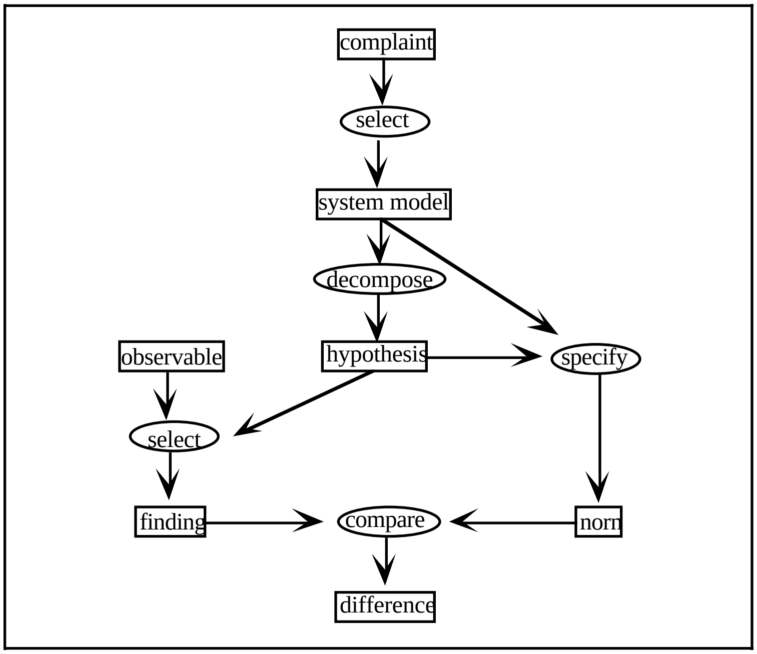


Figure 8.12: *KADS inference procedure for systematic diagnosis. Assumes single fault diagnosis. From* [260]. *Given a certain* complaint, *the* system model *is* decomposed *into* hypothet*ical candidate faulty components. A* norm *value for each candidate is collected from the* system model. *An observation for that candidate is requested from the* observables *(stored internally as a* finding*). The candidate hypothesis is declared to be the diagnosis based on the* compar*ison between the* norm *value and the* finding.

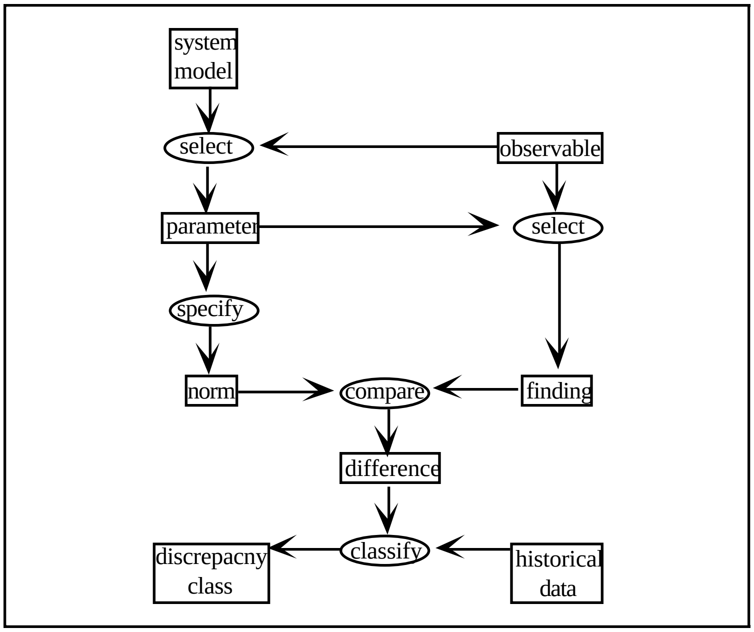


Figure 8.13: *KADS inference procedure for monitoring. From* [260]. *A* parameter *is selected from a* system model. *It's expected* norm*al value is generated from the model and collected from the* observable*s (stored as a* finding*). The current state of the monitoring system is reported as a* discrepancy class *after comparing the* finding *with the expected* norm*al value.*

[220]

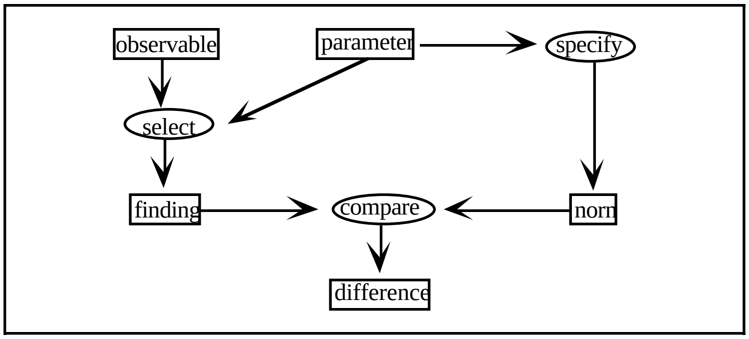


Figure 8.14: *Wielinga* et. al. [260] *note that this sub-graph is common to both the monitoring and diagnosis inference models of Figures 8.13 and 8.12 respectively.*

Having noted some low-level similarities between the distinctions that they have proposed, KL_B researchers do not take the next step and simplify their distinctions according to these observed similarities. The study of such similarities could identify some core mechanism that is central to several knowledge level tasks. Our case is that abduction/`Core` is such a central task and that reorganising inference procedures on abductive grounds simplifies design and implementation of KL processing:

- When we compare the differences between sub-hierarchies between 8.10 and 8.11, we see that our sub-hierarchies are much smaller variants of their roots than in conventional knowledge-level modelling. For example, at the symbol level, the difference between set-covering-based diagnosis and consistency-based diagnosis is very small[221]. In contrast, sibling inference models in conventional knowledge-level modelling may have totally different inference models (e.g. KADS has totally different inference models for *correlation*[222] and *verification* [223] even though they are both sub-types of *identification*).
- Note that Figure 8.10 is broader and flatter than the taxonomies of Figure 8.11; i.e. a hierarchy of inference models based on abduction makes fewer distinctions than conventional knowledge-level modelling.

Our general point here is that KL_B may have confused the modelling process, rather than clarifying it. In this regard, the analysis of Zdrahal & Motta of the *Propose & Revise* inference procedure to be particularly interesting. Having developed a KL_B solution to the Sisyphus-2 VT problem, the VITAL group then went on to analyse the symbol-level processing required [270]. We find that this symbol-level description of Propose & Revise to be far more useful in building working systems than the abstract KL_B description they originally offered for their system.

Further, when we review the KL_B inference procedures that we are familiar with, it is not clear that these inference procedures reflect current research in those areas:

[221] See Table 7.2, section 7.1.3.

[222] See Figure 8.9, section 8.2.4.

- *Diagnosis* : The pioneering work of Reiter [217] and the rigour of analysts such as Poole [197] and Console *et. al.* [51] have resulted in a common terminology and a well-defined set of research issues. The KADS diagnosis inference procedure[224] seems isolated from that work and only handles the simple case of strict hierarchical decomposition of components and a single fault. An informal KADS seminar convened at DX '93 [103] concluded that the KADS interpretation models for diagnosis were premature generalisations of a developing field.
- *Verification* : KADS "verification" is really validation since it uses data collected from the domain to perform test suite assessment. All the current work in KBS validation requires meta-knowledge of the dependencies between, at least, KB rules and, at most, KB literals. TMS architectures are used to build consistent subsets of the literals/rules . Validation then becomes a hunt for certain literals in the base dependants or the rule conclusions within those subsets [274]. KBS verification research uses the same dependency networks (but ignores the TMS worlds computation) and looks for logical inconsistencies in that network [204]. A generous reading of the KADS verification inference procedure[225] could say that at some level of abstraction, the KADS verification procedure captures this process (e.g. the *criterion* generation process uses TMS knowledge and *BEST* assessment knowledge within *match*). However, based on our work with HT4/`Core`, we would argue that the details missed by the KADS definition of *verification* are non-trivial. We find our own KBS validation framework (`Core`) more insightful and pragmatically more useful than the KADS model. For example, the need for managing assumptions in different worlds is a major addition to an inference architecture and is missing from the KADS definition.

Lastly, with the exception of the SBF experiment [145] (a result that has yet to be repeated or successfully generalised) there is little empirical evidence for the re-useability of inference procedures. Quite the opposite, in fact. Our reading of the KL$_B$ literature is that inference pattern re-use is rare:

- Buchanan & Smith's survey article ignores the 8-way classification of problem solving types proposed in the early 1980s (e.g. interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, control [102]) and remarks that:

  > *…there is no clear, unambiguous taxonomy of problem types that is independent of the methods used to solve problems. ([25], p154)*

[224] See Figure 8.12, section 8.3.2.

[225]

- The KADS-style inference procedures proposed by Bredeweg [20] for prediction via qualitative reasoning is very different to the qualitative prediction inference procedure proposed by Tansley & Hayball [251].
- Between the various camps of task researchers, there is little agreement on the details of the sub-routines of the inference procedures. Contrast the list of repeated processing modules from MCO [40], KADS [260] and SBF [145]. While there is some overlap, the lists are different.

8.3.3. Summary & Discussion

Numerous systems have been built using KADS [260]. However, other knowledge engineering approaches have published examples of their utility (e.g. MYCIN [269], CASNET [258], PIERS [206], PIGE [157], QMOD [79, 80, 153]). R1/XCON system [9], and PROSPECTOR [72]). The crucial case for KADS in particular and KL_B in general is *not* that KL_B can deliver applications. Rather, it should be that KL_B can deliver applications *better* than the alternatives. With regard to KL_B, this case has not been made.

For example, the general productivity benefits gained from re-using libraries of known inference procedures is yet to be demonstrated. While their exists some impressive experimental evidence on the utility of KL_B (e.g. Clancey removing search from MYCIN and the SBF experimental results), these results are too specific and isolated to form the basis of a general conclusion.

KADS is not the only software methodology that claims re-usable components without being able to demonstrate it. The object-oriented (OO) community also claims that objects are re-usable, with little or no experimental evidence to support that claim[226]. The OO community finds that their "re-usable" class libraries are ignored, or extensively modified when they are re-used. A similar process is observable with the KADS inference procedure libraries:

- Different KL_B schools have different lists of inference procedures (e.g. comparing SBF with KADS)
- In any one school, the list changes; e.g. (i) the different qualitative reasoning models proposed by Bredeweg [20] and Tansley & Hayball [251]; or (ii) new approaches are introduced such as [21].

[226] Claims that OO provides a large level of re-use are poorly documented and not conclusive [162]. For example, Stark reports 80% verbatim code re-use with "OO" for a range of NASA projects. On careful reading, it is discovered that Stark's "OO" re-use was seen in Fortran applications that ignored OO during analysis and design. However, mid-way through the coding phase, Stark's colleagues re-organised their text files into conceptual groups, each surrounded by a uniform interface and database access routines [245]. This style of "OO" programming does not satisfy Meyer's definition of OO [165]. Hence, we reject Stark's claim. We have argued elsewhere that OO researchers should collect their data in a more scientific manner; i.e. in a

- Often when a domain is analysed using tasks, a new inference procedure is required [134, 254].

One reason that the KADS inference procedures have not stabilised is that they may represent premature generalisation of a developing field (recall the problems we found above with the "verification" and diagnosis inference procedures).

We have argued against claims that KL_B is a useful explanation tool. A general purpose explanation tool contains:

- a model that can generate multiple proofs for any given behaviour;
- a profile of the user's knowledge;
- a case library of previous solutions;
- an abductive inference engine that selects and filters the multiple proofs according to the contents of the user model.

KL_B methodologies such as KADS make no mention of indeterminate models, user-profiles, or case libraries. Hence, in their current form, they are unsuitable for general explanation.

In proposing a symbol-level inference procedure (`Core`) as the basis of KBS, we are challenging the utility-of-abstraction assumption. KL_B makes the utility-of-abstraction assumption and abstracts to a very high level. Tests of the utility of the abstractions provided by KL_B have been, to date, inconclusive (e.g. the Sisyphus-2 experiments). Further, we know of almost zero-level abstraction tools that have out-performed systems based on extensive abstraction (e.g. PIERS [206] vs ABEL [187]). We suspect that KL_B is excessively abstracted. If we look at the details of an inference procedure, we may see similarities that allow us to simplify and unify seemingly different processes. The KADS community recognises that certain similarities exist between different inference procedures, but do not explore these low-level similarities (exception: [270]).We believe that if those similarities are rigorously explored, then seemingly different inference procedures can be unified and simplified. The HT4/`Core` process is a candidate for unifying inference procedures. HT4/`Core` provides a single inference procedure (exhaustive abduction with customisable inference assessment operators) which implements a wide-range of KBS tasks[227].

Not only can HT4/`Core` implement the inference associated with these tasks, it can also serve as a test engine. We have argued previously that testing is an essential module in the modern "knowledge-acquisition-as-modelling" perspective. Test requires dependency and invariant knowledge of all literals in all parts of the knowledge base. KADS makes no commitment to the internal processing of the sub-routines within its

[227]

inference procedures. Such sub-routines (e.g. classify, select, etc) are black boxes. Hence, we doubt the testability of KADS systems.

Note that our criticisms of the KL approach are focused on KL_B, not KL_A; i.e. PSCM-SOAR. Like PSCM-SOAR, we reject the KL_B assumption that modelling high-level inference procedures are the fundamental source of power for KBS systems:

- An assumption at the heart of KL_B is that once the problem solving framework has been chosen (i.e. the appropriate inference procedure has been selected), then the rest of KA is a simple matter of filling in the details required for that framework. This assumption is not supported by the available empirical evidence. Further, counter-examples exist that challenge this assumption; i.e. the successes of PIERS and the PSCM-SOAR Sisyphus-2 experiment.
- Our alternative proposal is to use testing as the basis of KA. We are not the first to propose this. Boehm's spiral model [12], Jacobson's use cases [115], and Wirfs-Brock's CRC cards [262] are all attempts to structure iterative software development based around a testing/prototyping process. Also, other KA researchers have discussed techniques for the iterative testing and revision of ideas; e.g. Silverman's expert critiquing systems[228]; Preece *et al*'s rule-base verification tools[229]; Zlatereva's test case generators[230]; Boose *et al'*s group decision support systems[231]; and Compton's ripple-down-rules[232,233]. HT4/`Core` is in the same spirit as these KA researchers. However, we have developed a detailed computational model of testing which, we argue, is more general than these above systems.

We believe that testing is a more powerful KA technique than using abstract inference procedures. Fundamentally, we do not believe that the different inference procedures described by (e.g.) KADS are actually different. Rather, they seem to be blurred reflections of a single inference procedure (abduction). Underlying the intricacies of existing KL_B methodologies is a minimal set of KR techniques that are the essential components of artificial expert competency. We ask proponents of KL_B if they have experimented with simpler alternatives? We note that designs that seem naive at first

[228] See section 7.2.4.

[229] See section 3.3.1.

[230] See section 3.3.2.

[231] See section 7.2.3.

[232] See sections 2.3.7 & 8.3.1.3.

[233] Note two exclusions from this list: repertory grids and KADS. Repertory grids are (described in section 2.3.8) are excluded since they only loosely connect concepts (e.g. see Figure 2.10 in section 2.3.8). KADS is excluded since it is not clear to use the role of prototyping in this approach (see our remarks in section 8.3.1.

glance may in fact produce satisfactory competency with comparatively less design effort[234].

Even when a KB is developed using these abstract inference procedures, it must still be tested; i.e. KL$_B$ may be best viewed as a pre-processor to our general abductive-based technique. We view knowledge as a dynamic structure that is created in the context of its need [17, 39, 47]. Inference procedure knowledge is still knowledge, and therefore a context-dependant dynamic construct that may not be relevant outside of the domain in which it was primarily evolved. In arguing that inference procedures can be reused outside of the context in which they were developed, KL$_B$ proponents are falling into the same trap as the expertise transfer techniques it was evolved to avoid.

An opponent to our position could argue that HT4/Core uses KL-style modelling, but under a different name. Recall that experts do not interact directly with the and-or graphs executed by HT4/Core. Like Boose et. al. [14], we acknowledge the importance of *mediating representations* that allow an expert to specify their reasoning in some form other than the runtime representation. In our framework, experts/ knowledge engineers use a domain-specific macro language that the model compiler expands into an and-or graph. Proponents of KL$_B$ could argue that these macro languages are really knowledge level constructs. We would disagree. While our macro languages do not *preclude* the use of KL$_A$ or KL$_B$ constructs, it is not necessarily true that they *include* such constructs. We impose no structure on the macro languages, excepting that they must be convertible into and-or graphs. We do not demand that these languages describe goals, states, operators, or abstract inference procedures. Macro languages that lack such constructs (e.g. QCM[235]) are not KL$_A$ or KL$_B$.

We have been very critical of the KL$_B$ approach in general and KADS in particular. In order to place these criticisms in perspective, we note that:

- One of the reasons that we can critique KADS so thoroughly is that the KADS community has documented its work with great care. Wielinga *et. al.* note that, as of 1992, has been used in some 40-50 KBS projects [260], 17 of which are described in published papers. This is an impressive list. Other software methodologies (e.g. the dominant object-oriented methodologies of [13, 115, 225, 262]) do not report themselves so thoroughly.
- Apart from the area of documentation, the problems we see with KADS are the same problems we see with most of contemporary software engineering. In our experience, most methodologies are normative *prescriptions* of how their authors think software should be developed rather than *descriptions* of

[234] See section 2.3.7.4.

[235]

empirically proven useful practice [163]. While early versions of the KL$_B$ methodologies were based on a careful reverse engineering of successful applications [37], we find that subsequent work tends more to invention rather than a generalisation of prior practice[236],[237]. As a symptom of this, we note that there is insufficient evaluation of KL$_B$ approaches[238].

[236] For example, many of the KL$_B$ models listed in [251] were created especially for the book by the authors from their own undocumented sources (see paragraph 4, page 260 of [251]).

[237] Exception: Clancey is always careful to develop his KL$_B$ with respect to previously developed systems [40].

[238]

We don't like their sound. Groups of guitars are on the way out: Decca Recording Company when turning down the Beatles, 1962.

I have fought a good fight, I have finished my course, I have kept the faith: 2 Timothy iv. 7. *A poem is never completed, only abandoned*: Verlaine. *What is observed is not nature itself, but nature exposed to our method of questioning… (Concepts) initially formed by abstraction from particular situations or experiential complexes acquire a life of their own:* Wiener Heisenberg. *The trouble isn't what people don't know; it's what they do know that isn't so:* Will Rogers. *It is no longer my moral duty as a human being to achieve an integrated and unitary set of explanations for my thoughts and feelings:* Browyn Davies.

9. Conclusion

Modern KA views KBS construction as a modelling activity. Experienced modellers caution us that models are inaccurate surrogates of reality. The inaccuracies introduced into models in poorly measured domains may be non-trivial. Most KBS domains are poorly measured. Potentially inaccurate models containing possibly non-trivial errors must be tested, lest they generate output that is inappropriate for certain circumstances. Such testing should not be based on internal syntactic assessment since we know of examples of working systems that contain internal syntactic anomalies. Our preferred test is external test suite assessment. Given a model to be tested and a library of known inputs and outputs, test suite assessment checks that the inputs can lead to the outputs. In the case of testing in poorly-measured domains, some assumptions may have to be made. In the case of testing indeterminate or non-monotonic models, these assumptions may be mutually exclusive and should be managed in independent and internally consistent assumption sets.

Testing can only demonstrate the presence of bugs, never their absence. A model that has "passed" a test may still produce inappropriate output in the future. Hence, testing must be repeated whenever new data relating to the domain becomes available. Pragmatically, this implies that testing is an on-going process right though out the knowledge acquisition and maintenance process.

Experiments with generalising qualitative hypothesis testing have lead to the development of a generalised architecture for automatic external test suite assessment of models. Formally, that architecture is abductive. Hence, generalised test-as-abduction can not only validate models, but also execute KBS for other abductive domains; e.g.:

- Verification;
- Deduction;

- Set-coverage diagnosis;
- Frame-based reasoning;
- Prediction;
- Explanation;
- Classification;
- Planning & Monitoring;
- Causal/ Qualitative reasoning;
- Design;
- Visual pattern recognition;
- Single-user model-based decision support systems.

Further, generalised-test-as-abduction would be useful in the following domains:

- Group decision support systems;
- Expert critiquing systems.

Some evidence exists that generalised-test-as-abduction would also be useful for:

- Case-based reasoning;
- Natural language processing;
- Certain types of financial reasoning;
- Learning.

More generally, we have argued that `Core` directly operationalises the model extraction process which Clancey and Breuker argue is at the core of expert system inference.

When compared with standard KBS frameworks (e.g. KL_B) we have found that generalised test-as-abduction offers a simpler, more unified core for expert systems. We have proposed generalised test-as-abduction as an alternative to KL_B since it can support (i) many of the inference procedures defined for KL_B; (ii) the on-going testing required for a KBS.

Testing imposes certain constraints on the KBS process:

- Assumption management requires dependency knowledge. Models must support an and-or graph of edges E and connecting the literals in its vertices V . This low-level view is not manageable by humans. Hence, we have defined model and data compilers that input higher-level constructs and convert them down into $<E,V>$.
- After a certain level of internal connectivity ($B= |E|/|V|$), indeterminate models can explain anything. Such models are not amenable to generalised test since they will incorrectly infer that all behaviours are possible. Hence, we believe that testing implies a limit to model complexity. Experimentally, we have seen that this limit in the neuroendocrinological domain is $B = 7$.
- Testing indeterminate models and maintaining separate assumption sets is a fundamentally slow process. Experimentally, we have seen that our current implementation displays an apparently non-cubic behaviour. Theoretically, we have reasons to believe that the process is exponential on model size ($N=|V|$). Hence, we believe that testing implies a limit to model size. Our current implementation has a limit of $N = 850$.

Based on the size and connectivity of models constructed by contemporary KE practice, we believe that we are already teetering on the edge of testability. Our results regarding the limits to testing are taken from the easy case where (i) the search space is explicit and pre-computed; (ii) I has an arity of two (i.e. given one literal, we can deterministically determine which other literals are inconsistent); and (iii) B does not include time-series data. Logically, such an explicit and-or graph is a finite propositional theory. To extend our results to full first-order theories/ implicit graphs/ invariants of arbitrary arity/ testing time series data, simply increase the runtimes.

Note that the discovery of a $O(N^3)$ abductive inference procedure for cyclic propositional theories in poorly-measured domains would remove this limit to testing. Given the fundamentally slow nature of our inference, we do not expect such a discovery in the near future. Nevertheless, we would encourage active research on this issue. We have argued for abduction as a unifying design principle for KBS. Research on this inference procedure would therefore benefit the general KBS process.

The standard reply to our limits to test argument is something like "We'll just divide our system into little sections". If the sections are truly separate, then this strategy will tame the complexity of testing. However, if any of the sections share a literal (i.e. refer to the same concept) then these "separate" sections are part of the same graph; i.e. they are not truly separate.

The implications of these limits-to-testing results are:

- We should not aim for large models. Rather, our KBS applications should comprise multiple small, totally independent sub-sections. We should develop systems in a breadth-first rather than a depth-first manner. Rather than exploring the inner complexities of some particular concept, we should build systems based on large numbers of separate concepts.
- If we do build models bigger than our test-limits, then we should plan what to do when these models behave inappropriately. While the usual case may be that model-based automation can remove the need for scare and expensive workers, the expected case should be that the system will fail and must be (temporarily) replaced with a manual system run by skilled staff. Note that organisations will have to budget for the costs incurred during the failure period. If the failure costs are prohibitively high (e.g. a computer inappropriately ordering a nuclear missile attack), then the automatic system *should not be built.*

There is one case where we concede that models bigger than our test-limits can be built. Models that are never executed will never produce inappropriate output. Such models do not need testing. The Boeing school of knowledge acquisition (e.g. [14, 17]) build models as intermediaries in the group decision support process. Such models may

Once this task is achieved, then the model can be discarded. We view this as a special case. The usual case is that models are built to be executed. Models that will be executed must be tested.

Note that in the case where we build models larger than our test limits, then we can still use Core to perform inference over those models. The internal data structures of Core are the same as the general state-space approached used by the search community to heuristically optimise search. However, in poorly-measured domains (and most KBS problems are in poorly measured domains[239]) we caution against this since models that can't be tested should be used very cautiously or not at all.

The original QMOD/JUSTIN goal was an interactive environment that let multiple experts around the world rapidly specify and assess their models, and those of their fellows. We now believe that goal to be unobtainable:

- HT4/Core used a TMS architecture to avoid the chronological backtracking of JUSTIN. In doing so, we speed up generalised test by two orders of magnitude. However, we do not expect much larger increases in the runtime speed. Generalised test is abduction and this is NP-hard. The creation of an international QMOD-knowledge base is fundamentally impossible.
- On the positive side, however, it seems that while we cannot build large consensus knowledge bases, we have found that generalised test prefers large amounts of data. Hence, we still could aim for the international database of experimental results.

We find that we have at least three possible research directions:

- We have made a theoretical case that HT4/Core can be used as the kernel of a variety of KBS tasks. This claim should be explored. Core should be used to build systems in the domains listed above.
- We believe that configurable *BEST* operators can declaratively specify much of the control knowledge for processing and-or graphs. We believe that combining meta-knowledge of *BEST* operators with the RDR approach is a starting point for the creation of ripple-down-modelling (RDM) environment. Standard RDR only allows the expert to patch declaractive knowledge. RDM would also permit the patching of control knowledge (expressed as sets of *BEST* operators). Ideally, the patching would still allow all old successful inferences to repeat, but corrects some flaw with the current inference. The goal of such an environment would be the creation of models that exhibit satisfying behaviour for non-trivial domains using a combination of no prior analysis and structured patching. If such a goal

[239]

could be reached, then not only would we have replaced KBS inferencing and validation with generalised test, but KBS analysis as well.

- The experimental studies with the HT4/`Core` algorithm could continue. Results like the fanout limit of 7 were very surprising. This result was only found after executing numerous models. Perhaps further experiments could yield further exciting discoveries.

This report began with a discussion of our philosophy of knowledge. It is appropriate that it ends in the same way:

- Our belief is that if implementation techniques for structured testing environments are widely known, then their use will lead us to a more realistic view of human knowledge. Knowledge will become an active adventure/exploration that never stops. Those with the wit to carry on the fight against confusion will earn the rewards of the modern intellectual hero; i.e. the honour of bearing new ideas to their knowledge-hungry fellows.
- We do not share the pessimism of some researchers who lament the end of the age of certainty[240]. If we reject a Platonic view of the universe and its somewhat spurious belief in an absolute "truth", we need not plunge into confusion. Our implementation experience has been that structured testing environments such as RDR and generalised-test-as-abduction are highly ordered entities. Recall our comparative analysis of abductive inference models versus knowledge level inference modules. We argued that abductive inference modules can unify and clarify knowledge level inference models. That is, (i) non-Platonic architectures are still amenable to rigorous analysis; so (ii) the opposite to Platonic certainty need not be chaos.

[240] For example, Agnew *et. al.* forcibly argue for the non-absolute nature of human knowledge. However, between the lines, we read that they do not like their general conclusion, but reluctantly acknowledge its

And for the tourist who really wants to get away from it all - safaris in Vietnam: Newsweek, late 1960s.

Imagine that the computational limits of generalised testing were somehow magically lifted. International knowledge bases could then be built, and constantly reviewed. Knowledge is power. Could we resist the temptation to abuse such a resources?

10. HT in 2040

Last Monday I got rich. It happened like this...

Every Monday, "OldManEmu" meets in my office (cause I've got the best graphics terminal). Officially, we're the "TestNet OZ-A hypothesis testing team" but since everyone else was naming their groups "Socrates" and "Plato" we changed our name to something all-wise and local.

A fly on the wall would have noticed that we were all a little distracted. As we argued (yet again) about our local policy for billing outside searches, we'd occasionally peek a glance at my screen. Nothing big was happening on TestNet so the screen was just doodling away to itself with a screen saver. There's always something happening: some low priority, long half-life query kicking round from site to site looking for an insight that might make it useful. We call them ghosts. Once I set off a ghost to do the following:

> *for every thing that can be proved,*
> *try and disprove it,*
> *and explain the reasoning that lead to the refutation*

I'd forgotten all about it till a few years back when it arrived home. The poor thing had been kicked off every site in the world since it used too much CPU and memory. Finally, after an exhaustive search of the globe, it came back to me and said that false was false by definition. Roughly speaking, I had asked it "why not?" and it had replied "just because." Such is the miracle of modern hypothesis testing. I shudder to think of the query-debt my little joke built up.

Anyway, my screen was programmed to run the screen save until some non-trivial spike in the network traffic happened (say, more than 2 gig per minute). And we were expecting a big spike, real soon now. Meanwhile, we got on with the paper work. We've got expert systems managing our queries, that but still, some human has to make the policy decisions based on some very-illogical political reasons.

John was arguing for a you-scratch-me and I'll-scratch-you policy. "Lets log the queries that extend our hypothesis space and favour transactions with sites that score high on that log.", he argued.

I wasn't too sure. "How does that sit with local control?", I asked. Local control is the official non-organising principal of TestNet. It takes a lot of resources to set up a TestNet node like us. The way hypothesis testing works, no one can remove an old hypothesis. Knowledge has to be write once/ delete never otherwise you can't test old premises. So, our basement floor is chock full of all our disc drives that store all the versions of our knowledge bases. No one wants to hand over those resources to some central body (remember, knowledge is power) so all the nodes in TestNet have guardians that control the in-coming searches. Once a query arrives, its up to the local site to control the inferencing associated with that query.

"Guardian could monitor our out-goings as well", argued John, "and he'd maintain the logs. We'd preferentially discount queries from sites that give us more CPU-share and better results when our queries hit their sites."

"And the network overheads associated with the monitoring?" asked Paula. John shrugged. "Compared to actually running the queries? Trivial."

I was surprised. John's shrug is usually louder and ruder. Especially with Paula. Don't know why they ever got married. Now that would be a good hypothesis to test: "John and Paula really love each other". Should chew up a few CPU cycles. Don't laugh. Ever since we cracked natural language parsing, we can process free-form queries like that. I mused on how it would be processed. First, TestNet would be explored for partially confirming evidence and I guess the marriage certificate would be found. Once some tentative evidence had been found for the proposition, then a larger search could be justified. Canberra's computers would be accessed for income tax and health records. The query would be processed by refutation. "Innocent till proven guilty" is the general hypothesis testing principal. Any statement can be added to the hypothesis space and remains unchanged until it generate contradictions or can be explicitly proven false. Unless some record could be found of marital discord (police records of bashing, spending sprees that could not be covered by their budgets, etc), TestNet would conclude that John and Paula really did love each other.

Not that a human would notice of course. The way they fight.

I remember the dull old days when hypothesis testing was confined to published papers. We used to have a floor of galley slaves that read the papers from all the journals, extracted an abstracted description, entered an executable version of that description in our hypothesis space, and then looked for any experimental evidence that referred to those hypotheses. Then, one fine day in 2003, someone added a natural language writer to their hypothesis tester and it wrote a paper outlining the discrepancies in what it had just been looking at, as well as some proposed fixes to the faulty model. The next step was obvious. Us humans stopped reading and writing the papers. Instead, we monitored changes in the hypotheses spaces around the world. Pretty soon all the hypothesis testing sites had a permanent background process running that reviewed the additions to the other sites. Rapidly changing hypotheses collected more attention as we performed our vulture act around new ideas. Sure, we still published. But we published active knowledge bases and hooked them into TestNet. I heard that Addison-Wesley once tried to sue TestNet. No one took much notice. Just the last kick of the paper dinosaur.

"But how do we know that the other sites are favouring us the same way?", asked Paula.

John went on to explain his ideas of "test queries", simple little assertions that any mainframe could answer in isolation. "We just shoot off a couple and measure the response time. The quicker the response, the more favoured we are." he explained.

"It shouldn't just be a time thing", she said. "Some sites spend longer because their hypothesis space is richer. I'd rather a query was explored thoroughly and not in some rapid-fire superficial way that missed an answer."

John disagreed but I only half-listened to them. I mused about what hypothesis testing had done to the scientific process (while keeping half an eye on my screen. Which still doodled to itself). I read in the history books that it used to take up to two years for a paper to go from be written to being published. The same cycle now takes about two minutes. What was that as a speed-up? 60 minutes an hour, 24 hours in a day, 365 days in a year, two years, divided by two minutes... I reached for the calculator to work that out when BING!, my screen lit up like a Christmas tree.

Paula and John instantly forgot their argument (it occurred to me later they were just amusing themselves while we were waiting). We all scrambled for the screen. Paula glanced at the new display of the TestNet traffic and bellowed out a gutsy laugh. "Its back on the air!", she cried. "Just when I said it would."

"China-B?" I asked. Paula's hands flew over the controls. The map of the world on screen spun a round and we fly into the Chinese mainland. The screen showed network traffic as vertical height. Over Beijing, a spike was forming.

China-B. It had to be. Two weeks ago, the Beijing TestNet site had announced a temporary withdrawal from the net. Sites did that from time to time to process the back-log of new hypothesis that couldn't be sorted out while the rest of the net demanded CPU time. The genetic algorithms crew call this "computer punctuated evolution". Which is a fancy way of saying its better to work on a new idea in a quiet room rather than in the middle of a crowd. Crowds tend to shout things down. Ideas grown in (temporary) isolation can develop the power required to survive back in the fray.

But China-B was special. Some of the intuitive jumps being made from that site were getting a little outrageous. Every site made a few little leaps, every now and again. We run our learning programs against our hypothesis space and sometimes discovered some new descriptors or abstracters. Usually the new ideas were only a small distance from the old hypothesis. But not at China-B. Some of its new ideas were really over the top. That one about whales talking to the dolphins convinced TestNet that some program had a diode loose over there in Beijing. Soon, net traffic to that site dropped. China-B then announced a two-week holiday. I thought that was a little odd. If a learning program goes screwy, I just amputate it without leaving the net. If China-B was withdrawing, then I felt that some new learning algorithm was being debugged and China-B wanted some peace and quiet to do some tests. So I checked the idea. I posted the hypothesis to the net and it generated no contradictions.

(Most people don't appreciate that TestNet never proves its' hypothesis- it merely reports failures to disprove. Experiments last century came to the reluctant conclusion that a closed-world assumption means you have to load all your common sense knowledge into the closed world. TestNet is more pragmatic. We just load in whatever knowledge source is a available and always check it when new data arrives. But the public doesn't understand. There was a famous interview on a day-time chat show where some business analyst was moaning about the way TestNet dribbles out conclusions to the rest of the world.

"You guys sit on information that could be of enormous national importance and keep us in the dark", he complained.

"Course we do" said the controller from TestNet Chicago. "Every time we let something out, the markets go crazy. Its scary to think that you guys trust TestNet's ideas so completely."

"And why not? TestNet is the most checked information source on the planet."

Chicago got really upset at this. She leant over the analyst and shouted at him, veins bulging on her neck: "Just because we can't refute something, doesn't mean its true!")

Anyway, when a new version of a learning program logs back on to the net, things usually go wild for a while till the rest of the world sorts out if it is genius or really just plain crazy. We watched the net traffic around China-B scream like a new born baby

"Its growing unusually fast." commented Paula.

I agreed. "Pull back a little. Lets get the big picture."

Paula flew us backwards from Beijing. We hovered some 30 clicks out of town and watched it grow. And grow. And grow.

"Ever seen anything grow that fast before?" John asked me. I shook my head and grabbed the second terminal. "Lets just get an average vector on the data transfer."

The phone rang. John took it. The face of the technical from the basement flashed up on the screen. "Hey," he asked," what's going on? We're on fire down here."

"Show us what you mean." asked John, glancing at me. I'd got my vector and a little alarm bell was ringing in my mind. The technical grabbed the handset and panned it over the disc drives. On the side of the drives are little lights that indicate the network traffic. Normally, write-once knowledge bases have a large redundancy rate. The rule-of-thumb was that the knowledge base at any TestNet site was only every 1% queried in any one day. That meant that most of our drives spun quietly to themselves without being queried. But now, all the access lights buzzed like angry red bees.

"Its China-B", I explained. "The vector?" asked Paula. I nodded.

"Why?" said John. "What's so special about our site?"

"Nothing."

"???"

"It's chewing up everyone's CPU. Paula, pull back and show us the Pacific picture."

Paula hesitated. "This is a hard movie to turn off", she commented. John and me turned back to the view over Beijing. China-B's spike now was towering over the city and reaching orbital heights. The spike's colours indicated the useful query rate. Queries resulted in hypothesis refinements were coloured gold. Queries that were timed-out before they terminated were coloured blood-red. Useless queries were black. Active queries were green. The shape of the spike is also coded. Width is the data transfer rate and height the derivative of the transfer rate. Brightness measures the average age of the queries: fresh queries are really bright while old ones are kind of dim. China-B's spike was a slender green shoot rearing up over the city, shooting skywards, and glowing with a neon brightness. Little gold flashes sparkled up and down the spike. I'd never seen anything so big.

"Never mind that now. You can watch the replays tomorrow." John said . "Take us up."

Paula snarled at John, then threw him the keyboard. "You fly."

John made a neat catch and pulled us out of Beijing. On the screen we flew upwards till the horizon was a circle beneath us. The whole Pacific region was on display.

"Cutting in access filter... now" said John, then gasped. The whole Pacific region lit in green cobwebs. Radiating out from China-B was more net traffic that we'd ever seen. Every node in TestNet was being flooded with queries like the ones at work downstairs.

"Now do you see?" I said. "There's nothing special about our sight. China-B is re-testing *every* hypothesis. Also, it seems its been hooked into a truly huge network over there. Response time over at Beijing is currently averaging 11.3 seconds."

Paula looked startled. "But that's a tenth of ours and they're handling all that traffic."

"I know", I said, "but for some reason Beijing wants the world to use China-B. They've given it massive resources."

"Now why do they want to do that?" mused John. "Why be so nice to the rest of the world? Bit suspicious, don't you think?"

"You're not suggesting that China-B is trying to LIE to the net?" asked Paula sarcastically. I winced at her tone but agreed with the sentiment. Once upon a time, a lot of people were worried that TestNet would be used to spread disinformation. My PhD thesis was a hush-hush DoD project to try and lie to the net. Didn't work. Every lie I produced generated so many inconsistencies that traffic to my node dropped off to zero. My general conclusion was that the effort required to generate a good lie was equal to the effort required to enter all the knowledge into the net. Nowadays, no one worries about lies. But still, China-B was acting very strange.

My computer went PING! and a little dialogue box popped up.

"Priority query returning." said John, reading the screen. "You got any ghosts running?"

I shook my head. "What's it say?"

John read the fine print. "It's a funny one: False is false by definition."

Two surprises in one day. "You're kidding. That ghost terminated years ago."

"Well its back on the screen now."

Then Paula made the statement that made me very rich (yes, yes, I paid her a cut).

"Now why would China-B have reset it's knowledge base?"

We asked her what she was talking about. No one reset KBs. TestNet has to be a write-once system. Etc. She waved all our remarks aside. "John, give us a read-out on the CPU and elapsed time on Tim's ghost."

"Four years in CPU, twenty years elapsed."

Paula leant over to me. "When were you an undergrad? 2010? Thirty years ago? That ghost has lost ten years of its elapsed life. China-B has reset."

"I see what you mean." I said. If a TestNet site reset its hypothesis space to some point in the past, then when it came back on to the net it would have some catching up to do. However, the catch-up would be relatively fast. Many of the hot topics current at the reset point would be resolved during the "lost" time. The catch-up would really be just a run around of the other TestNet sites asking them for the answers. It would generate a query pattern much like the one we were seeing here. China-B was now getting all the solutions to last decade's problems: the aids vaccine, the hyper-space drive, and which religion was true.

There was a catch, however. Simply throwing away a decades thinking would be like amputating with a chain saw. Messy. We'd notice the wound. China-B would have to set off a whole suite of demons to patch the wound. "And that would take weeks", I said.

Paula nodded. "China-B's been down for weeks. I bet you they spent that time resetting and vacuuming their KB. Your little query was so low priority and so unpopular that their consistency demons just missed it." She pushed John away from the terminal. "Lets do a little hunting. Can we come off the net for a while? I may need the CPU."

There's no such thing as "spare CPU" in a TestNet node. Every time we upgrade the machines, some smart-arsed postgrad comes up with a better learning program that chews up more processor time. Also, each upgrade makes us more attractive to queries from other sites. And the number of TestNet users increases exponentially each year. Paula had a hypothesis to check and she wanted to get all our OS queries off our machines. I didn't know what she was up to but she's pulled a few rabbits out of the hat before. I grabbed the phone and called down to the Technical. Told him we were coming off the net.

John leant over and said "What do you need?" I have a theory about those two (which I'll never put onto the net). They only stayed married because they can navigate TestNet better than any other pair I know. I sat back to enjoy the show. This could be good.

Paula explained. "I'm looking for a China-related hypothesis that is true now and wasn't true before China-B came up."

John nodded. "Something that could be disproved in a month or two, and China-B would still benefit."

"Check" she said. "Generation-one should relate to current motivations of China-B. What's their current problem space?"

Paula and John bent to work. I began to get the idea. They thought China-B was trying to flood TestNet with some mis-information. But to do it, they had to bury it in a mountain of "new" conclusions. By the time we sorted it all out and found the inconsistency, it would be too late. But too late for what?

The generation-one queries were spreading out over our site. Hypothesis testing is mostly spent trying to come up with an appropriate language for describing the queries. A query language is designed, used, and the usage monitored. A few test queries are posted and these suggest promising lines of inquiry. Or they don't. In which case the language is modified (maybe by a human and maybe by a learner).

John and Paula had posted some queries and were watching the results. A background demon popped up and reported the estimated cost of the extra resources now cut into China-B.

John whistled. "This is big. Major league government expense."

Paula snapped her fingers. "Government! Modify generation one. Change motivation search from China-B to Chinese government."

"Gotcha" said John and set to work to modify the descriptors in the generation-one language. The new query was posted and the results were much shorter this time. This encouraged them to work on and devise a generation-two language especially for small-scale temporal events.

I went for coffee. Ten minutes later, I brought back two cups for them and placed then by the terminal. They drank the coffee, but I don't think they even noticed. I watched as these two maestros reduced the search space. Another demon woke up and suggested that agricultural goods could be a useful descriptor in the queries. John quickly whipped up generation-four of the query language. (Generation-three had come and gone while I was at coffee. Some dead-end to do with weather variations).

Then, Paula nodded. "Enough private thinking. Lets get back on the net." I rang the technical and got us re-installed. We got a glimpse then of the havoc China-B as causing. We did a quick spin around the planet and green tendrils showed everywhere. Suddenly, over the horizon, a huge spoke came flying towards us. John grabbed the joystick and banked us right. The China-B spike (now a huge thick tree jutting out over much of northern China) swiped by us. For a moment, our screen was full of gold stars.

"That makes me giddy." I said.

"Hey, look" said John, "Tokyo is down." On the screen, the TestNet picture of Tokyo was a white gap.

"Now why would they be off the net?" I asked.

"Same reason as us?" wondered Paula.

"Lets give them a call." I suggested. I got their telephone number and contacted the Tokyo controller.

"Hail Emu!" she said. "How do you like the show?"

"Pretty lights." said John.

"Pretty suspicious if you ask me," said Tokyo. "We saw you going on vacation. Good to see you back. Any ideas?"

"You show us yours and we'll show you ours." said Paula. "Got a land-line?"

Tokyo and Paula arranged a language swap. They were slightly ahead of us. Their generation-five and our generation-four descriptors combined and we ran the new query. No luck. We both cut out of the net to give our learners a chance to chew on the generation-six language. Soon our screen was full of nouns and verbs all to do with crops.

"What is all this?" asked Paula.

"Looks like learner wants us to check out the farms." said John. "Lets give it a shot..." Generation-eight was all based around farming crops. We wrote some queries and set them off in. Five minutes later...

"Wheat" said Paula.

"What?" I asked.

"Wheat." she explained. "China-B has reversed the sign on an equation that relates to the nitrogen uptake of a new wheat fertiliser used in the southern provinces. That's the lie."

"Sure?"

"Damn positive." she said. "And its an obscure a reference as you will ever find."

"OK." I said. "John, get some models going. Predictions of crop yields for wheat with that sign negated... Oh, already done?"

"You betcha", he said pointing to the screen. "Someone's made a BIG mistake. Yield down 30%. Real famine stuff. China-B wanted to hide it for a month while it bought up big on the world market."

"And in the meantime, they're telling us that they have an over-production so the prices drop. " I said. "Then, in a month's time when the lie surfaces, they've bought their wheat at a depressed price. Clever."

"So what do we do?" asked Paula.

"What do you mean?" asked John.

“We can't just tell everyone." said Paula.

"Why not?"

"Because China needs it's wheat. If the prices go up..."

And so the next John/Paula war started. Paula wanted to squash the finding. John said that it couldn't be done; that Tokyo had our generation four language and learners just as good as us and that they'll work it out any minute now.

Me? I kept silent but I agreed with John. You can't lie to TestNet. Its too big. Just maintaining truth was hard enough, let alone a good lie. Sure, we had a head start from my old ghost but other sites would catch on soon enough. In fact, over John and Paula's shoulders, I could see more and more white gaps on the screen. All over the globe, sites were going off the net (no prizes for guessing what they were working on). China will just have to negotiate a wheat deal. And check its fertilisers better next time.

I rose to go. Paula stopped me. "Where are you going?" she demanded.

"Well..." I said reluctantly, "you guys are making such a racket and I need to make a phone call."

To my stock broker. Like I said, Monday was the day I got rich. Buying wheat.

X-rays will prove to be a hoax: Lord Kelvin.

Many shall run to and fro, and knowledge shall be increased: Daniel 12:4. *Those who do not remember the past are condemned to repeat it:* Santayana. *Knowledge cannot start from nothing- from a tabula rasa- nor yet from observation. The advance of knowledge consists, mainly, in the modification of earlier knowledge:* Karl Popper. *If you want new ideas, read old books:* Ivan Pavlov.

11. References

1. Aben, M., *On the Specification of Knowledge Model Components,* in **Formal Methods for Knowledge Modelling in the CommonKADS Methodology: A Compilation (KADS-EE/T1.2/TR/ECN/014/1.0)**, J. Balder and H. Akkermans, Editor. 1992, Netherlands Energy Research Foundation: p. 110-127.
2. Abu-Hakima, S. *Automating Knowledge Acquisition in Diagnosis* in **Fourth International Workshop on Principles of Diagnosis**. 1993. University College of Wales, Aberyswyth, Wales, United Kingdom:
3. Ackoff, R.L., *Management Misinformation Systems* **Management Science**, 1967. (December): p. 319-331.
4. Agnew, N.M., K.M. Ford, and P.J. Hayes, *Expertise in Context: Personally Constructed, Socially elected, and Reality-Relevant?* **International Journal of Expert Systems**, 1993. **7**(1).
5. Akkermans, H., F. van Harmelen, G. Schreiber, and B. Wielinga, *A Formalisation of Knowledge-Level Models for Knowledge Acquisition,* in **Formal Methods for Knowledge Modelling in the CommonKADS Methodology: A Compilation (KADS-EE/T1.2/TR/ECN/014/1.0)**, J. Balder and H. Akkermans, Editor. 1992, Netherlands Energy Research Foundation: p. 53-90.
6. Allemang, D., *Modelling a configuration problem with generic tasks*. 1992, GMD (Gesellschaft fur Mathematik und Datenverarbeitung mbH):
7. Anderson, J.R., **Cognitive Psychology and its Implications**. 1985, New York: W.H. freeman and Company.
8. Apple, **Hypercard Script Language Guide ,**. Apple Technical Publications, ed. APDA. 1987, .
9. Bachant, J. and J. McDermott, *R1 Revisited: Four Years in the Trenches* **AI Magazine**, 1984. (Fall): p. 21-32.
10. Baile, C.A., *Putative neurotransmitters in the hypothalmus and feeding* **Control of Feeding and the Regulation of Energy Balance, Federated Proceedings**, 1974. **33**(5): p. 1166-1175.
11. Bandekar, V.R., *Causal Models for Diagnostic Reasoning* **Artificial Intelligence in Engineering**, 1989. **4**(2).
12. Boehem, *Spiral Development Cycle* **Computer**, 1988. (May): p. 61-72.
13. Booch, G., **Object-Oriented Design with Applications**. 1991, Redwood City: Benjamin/Cummings.
14. Boose, J.H., J.M. Bradshaw, J.L. Koszareck, and D.B. Shema. *Knowledge Acquisition Techniques for Group Decision Support* in **Proceedings of the 7th Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1992. Banff, Canada:

15. Brachman, R.J., V.P. Gilbert, and H.J. Levesque, *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton*, in **Readings in Artificial Intelligence and Databases**, J. Mylopoulos and M.L. Brodie, Editor. 1989, Morgan Kaufmann: p. 293-300.
16. Brachman, R.J. and J.G. Schmolze, *An Overview of the KL-ONE Knowledge Representation System* **Cognitive Science**, 1985. **9**(2): p. pp171-216.
17. Bradshaw, J.M., K.M. Ford, and J. Adams-Webber. *Knowledge Representation of Knowledge Acquisition: A Three-Schemata Approach* in **6th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, ,October 6-11 1991**. 1991. Banff, Canada:
18. Bratko, I., I. Mozetic, and N. Lavrac, **KARDIO: a Study in Deep and Qualitative Knowledge for expert Systems**. 1989, MIT Press.
19. Brazier, F., P.H.G. van Langen, A. Phillipsen, N.J.E. Wijngaards, and M. Willems, *DESIRE: Designing an Elevator Configuration*, in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 39.1-39.33.
20. Bredeweg, B., *Expertise in Qualitative Prediction of Behaviour*. 1992, University of Amsterdam:
21. Breuker, J. *Components of Problem Solving and Types of Problems* in **8th European Knowledge Acquisition Workshop, EKAW '94**. 1994.
22. Brookes, C.H.P., *Requirements Elicitation for Knowledge Based Decision Support Systems*. 1986, Information Systems, University of New South Wales:
23. Bruno, L., *Patterns of Citation Errors,* 1991, Redwood City: Benjamin/Cummings.
24. Buchanan, B.G. and E.H. Shortliffe, **Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**. 1984, Addison-Wesley.
25. Buchanan, B.G. and R.G. Smith, *Fundamentals of Expert Systems*, in **The Handbook of Artificial Intelligence, Volume 4**, A. Barr, P.R. Cohen, and F. E.A., Editor. 1989, Addison-Wesley: p. 149-192.
26. Bylander, T., *A Critique of Qualitative Simulation from a Consolidation Viewpoint* **IEEE Transactions on Systems, Man, and Cybernetics**, 1988. **18**(2, March/April): p. 252-263.
27. Bylander, T., D. Allemang, M.C. Tanner, and J.R. Josephson, *The Computational Complexity of Abduction* **Artificial Intelligence**, 1991. **49**: p. 25-60.
28. Bylander, T. and S. Mittal, *CRSL: A Language for Classificatory Problem Solving and Uncertainty Handling* **AI Magazine**, 1986. (August): p. 66-77.
29. Campbell, A.N., V.F. Hollister, R.O. Duda, and P.E. Hart, *Recognition of a Hidden Material Deposit by an Artificially Intelligent Program* **Science**, 1982. **217**(3 September): p. 927-929.
30. Catlett, J. *Inductive learning from subsets or Disposal of excess training data considered harmful.* in **Australian Workshop on Knowledge Acquisition for Knowledge-Based Systems**. 1991. Pokolbin.:
31. Chandrasekaran, B., *Towards a taxonomy of problem solving types* **AI Magazine**, 1983. (Winter/Spring): p. 9-17.
32. Chandrasekaran, B., *Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design* **IEEE Expert**, 1986. (Fall): p. 23-30.
33. Chandrasekaran, B., *Design Problem Solving: A Task Analysis* **AI Magazine**, 1990. (Winter): p. 59-71.
34. Charniak, E. and D. McDermott, **Introduction to Artificial Intelligence**. 1987, Addison-Wesley. 701.
35. Charniak, E. and S.E. Shimony, *Cost-based abduction and MAP explanation* **Artificial Intelligence**, 1994. : p. 345-374.

36. Clancey, W., *The epistemology of rule-based systems: a framework for explanation.* **Artificial Intelligence**, 1983. **27**: p. 289-350.

37. Clancey, W., *Heuristic Classification* **Artificial Intelligence**, 1985. **27**: p. 289-350.

38. Clancey, W., *Viewing knowledge bases as qualitative models* **IEEE Expert**, 1989. (Summer): p. 9-23.

39. Clancey, W. *A Boy Scout, Toto, and a Bird: How Situated Cognition is Different from Situated Robotics* in **NATO Workshop on Emergence, Situatedness, Subsumption, and Symbol Grounding,**. 1991.

40. Clancey, W.J., *Model Construction Operators* **Artificial Intelligence**, 1992. **53**: p. 1-115.

41. Clark, P. and S. Matwin, *Learning Domain Theories Using Abstract Background Knowledge*. 1993, Department of Computer Science, Ottawa University, Canada:

42. Clark, P. and S. Matwin. *Using Qualitative Models to Guide Inductive Learning* in **Proceedings of the Tenth International Machine Learning Conference, ML-93**. 1993. Department of Computer Science, Ottawa University, Canada.

43. Coiera, E., *The Qualitative Representation of Physical Systems* **The Knowledge Engineering Review**, 1992. **7**(1): p. 1-23.

44. Coles, H., S, **Thinking About the Future: A Critique of the Limits to Growth**. 1974, Sussex University Press.

45. Compton, P., K. Horn, J.R. Quinlan, and L. Lazarus, *Maintaining an Expert System,* in **Applications of Expert Systems**, J.R. Quinlan, Editor. 1989, Addison Wesley: London. p. 366-385.

46. Compton, P., B. Kang, P. Preston, and M. Mulholland. *Knowledge Acquisition Without Analysis* in **European Knowledge Acquisition Workshop**. 1993.

47. Compton, P.J. and R. Jansen, *A philosophical basis for knowledge acquisition.* **Knowledge Acquisition**, 1990. **2**: p. 241-257.

48. Console, L., D. Dupre, and P. Torasso. *A theory of diagnosis for incomplete causal models* in **Proc. 11th IJCAI**. 1989. Detroit:

49. Console, L., D.T. Dupre, and P. Torasso. *A Completion Semantics of Object-level Abduction* in **Working Notes of the 1990 Spring Symposium on Automated Abduction**. 1990. Stanford University: Department of Information and Computer Science, University of Irvine.

50. Console, L., D.T. Dupre, and P. Torasso, *On the Relationship Between Abduction and Deduction* **Journal of Logic Programming**, 1991. **1**(5): p. 661-690.

51. Console, L. and P. Torasso, *A spectrum of definitions of model-based diagnosis* **Computational Intelligence**, 1991. **7**(3): p. 133-141.

52. Crawford, J., A. Farquhar, and B. Kuipers, *QPC: A Compiler from Physical Models into Qualitative Differential Equations,* in **Recent Advances in Qualitative Physics**, B. Faltings and P. Struss, Editor. 1992, The MIT Press: Cambridge, Mass. p. 17-32.

53. Darden, L., *Diagnosing and Fixing Faults in Theories,* in **Computational Models of Scientific Discovery and Theory Formation**, J. Sharager and P. Langley, Editor. 1990, Morgan Kaufmann Publishers Inc.: San Mateo, California.

54. Davis, R. and W. Hamscher, *Model-based reasoning: Troubleshooting,* in **Readings in Model-Based Diagnosis**, W. Hamscher, L. Console, and J. DeKleer, Editor. 1992, Morgan Kaufmann: p. 3-24.

55. Davis, R., H. Shrobe, and P. Szolovits, *What is a Knowledge Representation?* **AI Magazine**, 1993. (Spring): p. 17-33.

56. DeKleer, J., *An Assumption-Based TMS* **Artificial Intelligence**, 1986. **28**: p. 163-196.

57. DeKleer, J., *Extending the ATMS* **Artificial Intelligence**, 1986. **28**: p. 163-196.

58. DeKleer, J., *Problem Solving with the ATMS* **Artificial Intelligence**, 1986. **28**: p. 197-224.

59. DeKleer, J., *A perspective on assumption-based truth maintenance* **Artificial Intelligence**, 1993. **59**(February): p. 63-67.
60. DeKleer, J., *A view on qualitative physics* **Artificial Intelligence**, 1993. **59**: p. 105-114.
61. DeKleer, J. and J.S. Brown, *A Qualitative Physics Based on Confluences* **Artificial Intelligence**, 1984. **25**: p. 7-83.
62. DeKleer, J. and J.S. Brown, *Theories of Causal Ordering* **Artificial Intelligence**, 1986. **29**: p. 33-61.
63. DeKleer, J., A. Mackworth, and R. Reiter, *Characterizing Diagnosis Systems,* in **Readings in Model-Based Diagnosis**, W. Hamscher, L. Console, and J. DeKleer, Editor. 1992, p. 54-65.
64. DeKleer, J. and O. Raiman. *How to Diagnose well with very Little Information* in **DX-93: Fourth International Workshop on Principles of Diagnosis**. 1993. Aberystwth: Hewlett-Packard.
65. DeKleer, J. and B.C. Williams, *Diagnosing multiple faults* **Artificial Intelligence**, 1987. **32**(1): p. 97-130.
66. DeKleer, J. and B.C. Williams. *Diagnosis with Behavioural Modes* in **Proceedings of IJCAI '89**. 1989.
67. Dennett, D.C., *Letter to the Editor* in *The New York Review of Books*. 1982, p. 56-57.
68. Dieng, R., O. Corby, and S. Lapalut, *Acquisition of Gradual Knowledge,* in **Knowledge Acquisition for Knowledge-Based Systems: 7th European Workshop, EKAW '93**, N. Aussenac, *et al.*, Editor. 1993, p. 407-426.
69. Dowling, W.F. and J.H. Gallier, *Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae* **Journal of Logic Programming**, 1984. **3**: p. 267-284.
70. Doyle, J., *A Truth Maintenance System* **Artificial Intelligence**, 1979. **12**: p. 231-272.
71. Dreyfus, H., **What Computers Can't D: A Critique of Artificial Reason**. 1979, San Francisco: Freeman.
72. Duda, R., J. Gasching, and P. Hart, *Model Design in the Prospector Consultant System for Mineral Exploration,* in **Expert Systems in the Microelectronic Age**, D. Michie, Editor. 1979, Edinburgh University Press: Edinburgh. p. 153-167.
73. Duda, R.O., P.E. Hart, and R. Reboh, *Letter to the Editor* **Artificial Intelligence**, 1985. **26**: p. 359-360.
74. Duda, R.O. and E.H. Shortliffe, *Expert Systems Research* **Science**, 1983. **220**(15 April): p. 261-268.
75. Eshghi, K. *A Tractable Class of Abductive Problems* in **IJCAI '93**. 1993. Chambery, France:
76. Etherington, D.W., *Formalizing Nonmonotonic Reasoning Systems* **Artificial Intelligence**, 1987. **31**: p. 41-85.
77. Faltings, B. and P. Struss, ed. *Recent Advances in Qualitative Physics* 1992, The MIT Press: Cambridge, Mass.
78. Feigenbaum, E. and P. McCorduck, **The Fifth Generation**. 1983, New York: Addison-Wesley.
79. Feldman, B.T., P.J. Compton, and G.A. Smythe. *Towards Hypothesis Testing: JUSTIN, Prototype System Using Justification in Context* in **Proceedings of the Joint Australian Conference on Artificial Intelligence, AI '89**. 1989.
80. Feldman, B.Z., P.J. Compton, and G.A. Smythe. *Hypothesis Testing: an Appropriate Task for Knowledge-Based Systems* in **4th AAAI-Sponsored Knowledge Acquisition for Knowledge-based Systems Workshop**. 1989. Banff, Canada, October 1989.:
81. Forbus, K., *Qualitative Process Theory* **Artificial Intelligence**, 1984. **24**: p. 85-168.
82. Forbus, K. *Interpreting Measurements of Physical Systems* in **AAAI '86**. 1986.

83. Forbus, K., *Pushing the Edge of the (QP) Envelope,* in **Recent Advances in Qualitative Physics**, B. Faltings and P. Struss, Editor. 1992, The MIT Press: Cambridge, Mass. p. 245-261.

84. Forbus, K., *Qualitative process theory: twelve years after.* **Artificial Intelligence**, 1993. **59**: p. 115-123.

85. Forbus, K. and J. DeKleer, **Building Problem Solvers**. 1993, A Bradford Book, The MIT Press. 702.

86. Fouche, P. and B. Kuipers, *An Assessment of Current Qualitative Simulation Techniques,* , B. Faltings and P. Struss, Editor. 1992, The MIT Press: Cambridge, Mass. p. 263-278.

87. Freeman-Benson, B.N., J. Maloney, and A. Borning, *An Incremental Constraint Solver* **Communications of the ACM**, 1990. **33**(1): p. 54-63.

88. Gaines, B., *AAAI 1992 Spring Symposium Series Reports: Cognitive Aspects of Knowledge Acquisition* in *AI Magazine*. 1992, p. 24.

89. Gaines, B. and P. Compton. *Induction of Ripple Down Rules* in **AI '92**. 1992. Hobart, Tasmania: World Scientific.

90. Gaines, B.R. and M.L.G. Shaw. *Comparing the Conceptual Systems of Experts* in **IJCAI '89**. 1989. Detroit, USA:

91. Gaines, B.R. and M.L.G. Shaw. *Using Knowledge Acquisition and Representation Tools to Support Scientific Communities* in **AAAI '94**. 1994.

92. Gautier, P.O. and T.R. Gruber. *Generating Explanations of Device Behaviour Using Compositional Modelling and Causal Ordering* in **AAAI '93**. 1993. Washington, USA:

93. Geigner, D., A. Paz, and J. Pearl, *Learning Simple Causal Structures* **International Journal of Intelligent Systems**, 1993. **8**: p. 231-247.

94. Genesereth, M.R., *The Use of Design Descriptions in Automated Diagnosis* **Artificial Intelligence**, 1984. **24**: p. 411-436.

95. Ginsberg, A. *A new approach to checking knowledge bases for inconsistentcy and redundancy* in **Proc. 3rd Annual Expert Systems in Government Conference**. 1987. IEEE Computer Society.

96. Ginsberg, A. *Theory Reduction, Theory Revision, and Retranslation* in **AAAI '90**. 1990.

97. Glass, L. and M.C. Mackey, **From Clocks to Chaos**. 1988, Guilford, Surrey: Princeton University Press.

98. Gleick, J., **Chaos**. 1987, Cardinal. 352.

99. Guha, R.V. and D.B. Lenat, *Enabling Agents to Work Together* **Communications of the ACM**, 1994. **37**(7): p. 127-142.

100. Hamscher, W. *Explaining Unexpected Financial Results* in **AAAI Spring Symposium on Automated Abduction**. 1990.

101. Hayes, P.J., *The Logic of Frames,* in **Frame Conceptions and Text Understanding**, D. Metzing, Editor. 1979, Walter de Gruyter and Co.: Berlin. p. 46-61.

102. Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, **Building Expert Systems**. 1983, Reading, Massachusetts: Addison-Wesley. 444.

103. Hewlett-Packard. *DX '93* in **Fourth International Workshop on Principles of Diagnosis**. 1993. University College of Wales, Aberyswyth, Wales, United Kingdom:

104. Hirata, K. *A Classification of Abduction: Abduction for Logic Programming* in **Proceedings of the Fourteenth International Machine Learning Workshop, ML-14**. 1994.

105. Hofstader, D.R. and D.C. Dennett, **The Mind's I**. 1981, Basic Books.

106. Hofstadter, D.R., **Gödel, Escher, Bach: An Eternal Golden Braid**. 1980, Penguin Books. 777.

107. Hogger, C.J., **Introduction to Logic Programming**. 1984, London: Orlando.

108. Ishida, Y. *Using Global Properties for Qualitative Reasoning: A Qualitative System Theory* in **Proceedings of IJCAI '89**. 1989.
109. Ishida, Y. *A Graphical Approach to Qualitative Reasoning on Dynamic Systems* in **ECAI '92**. 1992. Vienna, Austria:
110. Iwasaki, Y. *Causal Ordering in a Mixed Structure* in **Proceedings of AAAI '88**. 1988.
111. Iwasaki, Y., *Qualitative Physics,* in **The Handbook of Artificial Intelligence**, A. Barr, P.R. Cohen, and E.A. Feigenbaum, Editor. 1989, Addison Wesley: p. 323-413.
112. Iwasaki, Y. and S. H.A., *Retrospective on "Causality in device behaviour"* **Artificial Intelligence**, 1993. **59**: p. 141-146.
113. Iwasaki, Y. and H. Simon, *Theories of Causal Ordering: Reply to deKleer and Brown* **Artificial Intelligence**, 1986. **29**: p. 63-72.
114. Iwasaki, Y. and H.A. Simon, *Causality in Device Behaviour* **Artificial Intelligence**, 1986. **29**: p. 3-31.
115. Jacobson, I., M. Christerson, P. Jonsson, and G. Overgaard, **Object-Oriented Software Engineering: A UseCase Driven Approach**. 1992, Addison-Wesley. 520.
116. Kabas, A.C. and P. Mancrella. *Generalized Stable Models: A Semantics for Abduction* in **ECAI-90**. 1990. Stockholm, Sweden:
117. Kahn, G., S. Nowlan, and J. McDermott, *Strategies for Knowledge Acquisition* **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 1985. **Vol. PAMI-7**(No. 5. September): p. 511-522.
118. Kahneman, D. and A. Tversky, *The Psychology of Preferences* **Scientific American**, 1982. **246**: p. 136-142.
119. Kanovich, M.I. *Efficient Program Synthesis: Semantics, Logic, Complexity* in **Theoretical Aspects of Computer Software**. 1991. September, 1991, Sendai, Japan:
120. Konolige, K., *Abduction versus Closure in Causal Theories* **Artificial Intelligence**, 1992. **53**: p. 255-272.
121. Krieger, D.T., *The Hypothalmus and Neuroendocrinology,* in **Neuroendocrinology**, D.T. Krieger and J.C. Hughes, Editor. 1980, Sinauer Associates, Inc.: Sunderland, Massachusetts. p. 3-122=.
122. Kuhn, T., **The Structure of Scientific Revolutions**. 1962, New York: Cambridge Press.
123. Kuipers, B., *Qualitative Simulation* **Artificial Intelligence**, 1986. **29**: p. 229-338.
124. Kuipers, B.J., *Reasoning with qualitative models* **Artificial Intelligence**, 1993. **59**: p. 125-132.
125. Kuipers, B.K., *Qualitative simulation: then and now* **Artificial Intelligence**, 1993. **59**: p. 133-140.
126. Laird, J.E. and A. Newell. *A Universal Weak Method: Summary of Results* in **IJCAI '83**. 1983.
127. Laird, J.E., A. Newell, and P.S. Rosenbloom, *Soar: An Architecture for General Intelligence* **Artificial Intelligence**, 1987. **33**(1): p. 1-64.
128. Larkin, J., J. McDermott, D.P. Simon, and H.A. Simon, *Expert and Novice Performance in Solving Physics Problems* **Science**, 1980. **208**(20 June): p. 1335-1342.
129. Leake, D.B., *Goal-Based Explanation Evaluation* **Cognitive Science**, 1991. **15**: p. 509-545.
130. Leake, D.B. *Focusing Construction and Selection of Abductive Hypotheses* in **IJCAI '93**. 1993.
131. Lee, M., P. Compton, and B. Jansen. *Modelling with Context-Dependant Causality* in **Proceedings of the Second Japan Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1992. Kobe, Japan:
132. Lenat, D.B. and R.V. Gutha, *CYC: A Midterm Report* **AI Magazine**, 1990. (Fall): p. 32-59

133. Levesque, H. *A Knowledge-Level Account of Abduction (preliminary version)* in **IJCAI '89**. 1989. Detroit, Michigan, USA:

134. Linster, M. and M. Musen, *Use of KADS to create a conceptual model of the ONCOCIN task* **Knowledge Acquisition**, 1992. **4**(1): p. 55-88.

135. Mackworth, A.K., *Consistency in Networks of Relations* **Artificial Intelligence**, 1977. **8**: p. 99-118.

136. Mackworth, A.K. and E.C. Frueder, *The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems* **Artificial Intelligence**, 1985. **25**: p. 65-74.

137. Mahidadia, A., C. Sammut, and P. Compton. *Applying Inductive Logic Programming to Causal Qualitative Models in Neuroendocrinology* in **AI '94**. 1994. Armidale, NSW:

138. Mahidadia, A.J. *Personal communication* . 1992. Personal communication.

139. Mahidadia, A.J., P. Compton, T.J. Menzies, C. Sammut, and G.A. Smythe. *Inventing Causal Qualitative Models: A Tool for Experimental Research,* in **Proceedings of AI ‘92**. 1992. Hobart, Tasmania, November:

140. Mahidadia, A.J., C. Sammut , and P. Compton. *Building and Maintaining Causal Theories* in **AAAI Symposium on Knowledge Assimilation**. 1992. Stanford University, Spring, 1992.:

141. Mahidadia, A.J., C. Sammut, and P. Compton. *Building and Maintaining Causal Qualitative Theories Using Inductive Logic Programming* in **AI '93**. 1993. Melbourne, Australia: World Scientific.

142. Marcus, S. and J. McDermott, *SALT: A knowledge acquisition language for propose-and-revise systems* **Artificial Intelligence**, 1989. **39**(1): p. 1-37.

143. Marcus, S., J. Stout, and J. McDermott, *VT: An Expert Elevator Designer That Uses Knowledge-Based Backtracking* **AI Magazine**, 1987. (Winter): p. 41-58.

144. Marques, D., G. Dallemagne, G. Kliner, J. McDermott, and D. Tung, *Easy Programming: Empowering People to Build Their Own Applications* **IEEE Expert**, 1992. (June): p. 16-29.

145. Marques, D., G. Klinker, G. Dallemangne, P. Gautier, J. McDermott, and D. Tung. *More data on useable and reusable programming constructs* in **6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1991. Banff, Canada:

146. McDermott, J., *R1: The Formative Years* **AI Magazine**, 1981. **2**(2): p. 21-29.

147. McDermott, J., *R1 ("XCON") at age 12: lessons from an elementary school achiever* **Artificial Intelligence**, 1993. **59**: p. 241-247.

148. McIntosh, J.E.A. and R.P. McIntosh, **Mathematical Modelling and Computers in Endocrinology**. 1980, Berlin: Springer-Verlag.

149. Meadows, D.H., D.L. Meadows, J. Randers, and W.W. Behrens, **The Limits to Growth**. 1972, Potomac Associates. 205.

150. Medawar, P.B., **Advice to a Young Scientist**. 1979, London and Sydney: Pan Books.

151. Menzies, G.D., *An Econometric Analysis of the Dark Figure of Crime*. 1985, University of New England:

152. Menzies, T. and P. Compton. *The (Extensive) Implications of Evaluation on the Development of Knowledge-Based Systems* in **Proceedings of the 9th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1995.

153. Menzies, T., A. Mahidadia, and P. Compton. *Using Causality as a Generic Knowledge Representation, or Why and How Centralised Knowledge Servers Can Use Causality* in **Proceedings of the 7th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1992. Banff, Canada, October 11-16,:

154. Menzies, T.J. *The Complexity of Model Review* in **Dx -93: The International Workshop on Principles on Model-Based Diagnosis**. 1993. Aberystwyth, Wales, UK:

155. Menzies, T.J. *Exhaustive Abduction: A Practical Model Validation Tool* in **ECAI '94 Workshop on Validation of Knowledge-Based Systems**. 1994. Amsterdam, Holland:

156. Menzies, T.J. *A Precise Semantics for Vague Diagrams* in **AI'94**. 1994. Armidale, Australia:

157. Menzies, T.J., J. Black, J. Fleming, and M. Dean. *An Expert System for Raising Pigs* in **Th first Conference on Practical Applications of Prolog**. 1992. London, UK.:

158. Menzies, T.J. and P. Compton. *Causal Explanations as a Tool for refining Qualitative Models* in **ECAI '92 Workshop on Improving the Use of Knowledge-Based Systems with Explanations**. 1992. Vienna, Austria:

159. Menzies, T.J. and P. Compton. *Knowledge Acquisition for Performance Systems; or: When can "tests" replace "tasks"?* in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1994. Banff, Canada:

160. Menzies, T.J., P. Compton, B. Feldman, and T. Toft. *Qualitative Compartmental Modelling* in **Proceedings of the AAAI Symposium on Diagrammatic Reasoning**. 1992. Stanford University, March 25-27:

161. Menzies, T.J., P. Compton, and A. Mahidadia. *Communicating Research Models of Human Physiology using Qualitative Compartmental Modelling* in **Communicating Scientific and Technical Knowledge, a AAAI '92 workshop**. 1992. San Jose, California, July 12-17:

162. Menzies, T.J., J. Edwards, and K. Ng. *The Mysterious Case of the Missing Re-usable Class Libraries* in **Tools Pacific 1992**. 1992. Sydney, Australia: Prentice Hall.

163. Menzies, T.J. and P. Haynes. *The Methodologies of Methodologies; or, Evaluating Current Methodologies: Why and How* in **Tools Pacific '94**. 1994. Melbourne, Australia:

164. Menzies, T.J. and B.R. Markey. *A Micro-Computer, Rule-Based Prolog Expert-System for Process Control in a Petrochemical Plant* in **Proceedings of the Third Australian Conference on Expert Systems, May 13-15**. 1987. Sydney, Australia:

165. Meyer, B., **Object Oriented Software Construction**. 1988, Prentice Hall.

166. Miller, P.L., **Expert Critiquing Systems**. 1986, Spriner-Verlag.

167. Minsky, M., *A Framework for Representing Knowledge,* in **The Psychology of Computer Vision**, P. Winston, Editor. 1975, McGraw-Hill: New York. p. 211-277.

168. Mintzberg, H., *The Manager's Job: Folklore and Fact* **Harvard Business Review**, 1975. (July-August): p. 29-61.

169. Moberg, D. *Personal communication* . 1992. Personal communication.

170. Motta, E., O. K., N. Shadbolt, A. Stutt, and Z. Zdrahal, *A VITAL Solution to the Sisyphus II Elevator Design Problem,* in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 40.1-40.25.

171. Mulholland, M., P. Preston, B. Hibbert, and P. Compton. *An expert system for ion chromatography developed using machine learning and knowledge in context* in **Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems**. 1993. Edinburgh:

172. Myers, G.J., *A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections* **Communications of the ACM**, 1977. **21**(9, September): p. 760-768.

173. Newell, A., *The knowledge level* **Artificial Intelligence**, 1982. **18**: p. 87-127.

174. Newell, A., *Reflections on the Knowledge Level* **Artificial Intelligence**, 1993. **59**(February): p. 31-38.

175. Newell, A. and H.A. Simon, **Human Problem Solving**. 1972, Englewood Cliffs, N.J.: Prentice Hall.

176. Newell, A., G.R. Yost, J.E. Laird, P.S. Rosenbloom, and E. Altmann, ed. *Formulating the Problem Space Computational Model* **The Soar Papers**, ed. P.S. Rosenbloom, J.E. Laird, and A. Newell. Vol. 2. 1991, MIT Press: . 1321-1359.

177. Ng, H.T. and R.J. Mooney. *The Role of Coherence in Constructing and Evaluating Abductive Explanations* in **Working Notes of the 1990 Spring Symposium on Automated Abduction**. 1990. UC Irvine.

178. Nguyen, T.A., W.A. Perkins, T.J. Laffey, and D. Pecora, *Knowledge Base Verification* **AI Magazine**, 1987. (Summer): p. 69-75.

179. Norman, D.A., **The Design of Everyday Things**. 1989, DoubleDay Currency. 257.

180. Norvig, P., **Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp**. 1992, Morgan Kaufmann. 946.

181. O'Hara, K. and S. N. *AI Models as a Variety of Psychological Explanation* in **IJCAI**. 1993. Chambery, France:

182. O'Rourke, P., *Working Notes of the 1990 Spring Symposium on Automated Abduction*. 1990, University of California, Irvine, CA.:

183. Pagnucco, M. *On Abduction as Learning* . 1994. Personal communication.

184. Pagnucco, M., A.C. Nayak, and N.Y. Foo. *Abductive Expansion: Abductive Inference and the Process of Belief Change* in **AI '94**. 1994. Armidale, Australia:

185. Paris, C. *Combining discourse strategies to generate descriptions along a naive/expert spectrum.* in **IJCAI '87**. 1987. Milan, Italy:

186. Paris, C.L., *The Use of Explicit User Models in a Generation System for Tailoring Answers to the User's Level of Expertise,* in **User Models in Dialog Systems**, A. Kobsa and W. Wahlster, Editor. 1989, Springer-Verlag: p. 200-232.

187. Patil, R.S., P. Szolovitis, and W.B. Schwartz. *Causal Understanding of Patient Illness in Medical Diagnosis* in **IJCAI '81**. 1981.

188. Pearl, J. *Embracing Causality in Formal Reasoning* in **AAAI'87**. 1987.

189. Pearl, J. and R.E. Korf, *Search Techniques* **Ann. Rev. Comput. Sci. 1987**, 1987. **2**: p. 451-67.

190. Pearl, J. and T.S. Verma. *A Theory of Inferred Causation* in **Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference**. 1991. Morgan Kaufmann.

191. Pendrith, M. *Complexity of a depth-first search through an acyclic network (not fully-connected).* . 1993. Personal communication.

192. Phillips, L.D., *A Theory of Requisite Decision Models* **Acta Psychologica**, 1984. **56**: p. 29-48.

193. Poeck, K., D. Fensel, D. Landes, and J. Angele, *Combining KARL and Configurable Role Limiting Methods for Configuring Elevator Systems,* in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 41.1-41.32.

194. Poole, D. *On the Comparison of Theories: Preferring the Most Specific Explanation* in **IJCAI '85**. 1985.

195. Poole, D., *A Logical Framework for Default Reasoning* **Artificial Intelligence**, 1988. **36**: p. 27-47.

196. Poole, D. *Representing knowledge for logic-based diagnosis.* in **Proc. of the Int. Conf. on Fifth Generation Computer Systems**. 1988. Tokyo:

197. Poole, D. *Normality and Faults in Logic-Based Diagnosis* in **IJCAI '89**. 1989. Detroit, USA:

198. Poole, D. *Hypo-Deductive Reasoning for Abduction, Default Reasoning, and Design* in **Working Notes of the 1990 Spring Symposium on Automated Abduction**. 1990. UC Irvine.

199. Poole, D., *A Methodology for Using a Default and Abductive Reasoning System* **International Journal of Intelligent Systems**, 1990. **5**: p. 521-548.

200. Poole, D., *Who Chooses the Assumptions*? 1991, Computer Science Department, University of British Colombia:

201. Poole, D., *Logic Programming, Abduction, and Probability* **New Generation Computing**, 1993. **11**: p. 377-400.

202. Pople, H.E. *On the mechanization of abductive logic.* in **IJCAI '73**. 1973.

203. Popper, K.R., **Conjectures and Refutations,**. 1963, London: Routledge and Kegan Paul.

204. Preece, A.D., *Principles and Practice in Verifying Rule-based Systems* **The Knowledge Engineering Review**, 1992. **7**(2): p. 115-141.

205. Preece, A.D. and R. Shinghal. *Verifying Knowledge Bases by Anomaly Detection: An Experience Report* in **ECAI '92**. 1992. Vienna:

206. Preston, P., G. Edwards, and P. Compton. *A 1600 rule expert system without knowledge engineers.* in **Second World Congress on Expert Systems**. 1993. Lisbon: Pergamon.

207. Puccia, C.J. and R. Levins, **Qualitative Modelling of Complex Systems: An Introduction to Loop Analysis and Time Averaging**. 1985, Cambridge, Mass.: Harvard University Press. 259.

208. Quinlan, J.R., *Induction of Decision Trees* **Machine Learning**, 1986. **1**: p. 81-106.

209. Quinlan, J.R., **C4.5: Programs for Machine Learning**. 1993, Morgan Kaufmann Publishers.

210. Rajaraman, C. and M.R. Lyu, *Reliability and Maintainability Software Coupling Metrics in C++ Programs* 1992. : p. 303-311.

211. Ramsey, C.L., J.A. Reggia, D.S. Nau, and A. Ferrention, *A Comparative Analysis of Methods for Expert Systems* **Int. J. Man-Machine Studies**, 1986. **24**: p. 475-499.

212. Reggia, J., *Sample System-D knowledge base sent via email*. 1992,

213. Reggia, J., D.S. Nau, and P.Y. Wang, *Diagnostic expert systems based on a set covering model.* **Int. J. of Man-Machine Studies**, 1983. **19**(5): p. 437-460.

214. Reggia, J.A., *Knowledge-Based Decision Support Systems: Development Through KMS*. 1981, Department of Computer Science, University of Maryland:

215. Reggia, J.A. *Abductive Inference* in **Proceedings of the Expert Systems in Government Symposium**. 1985. Washington, D.C.:

216. Reiter, R., *A Logic for Default Reasoning* **Artificial Intelligence**, 1980. **13**: p. 81-132.

217. Reiter, R., *A theory of diagnosis from first principles* **Artificial Intelligence**, 1987. **32**(1): p. 57-96.

218. Reiter, R. and J. DeKleer. *Foundations of Assumption-Based Truth Maintenance: Preliminary Report* in **AAAI '87**. 1987.

219. Rennels, G.D., E.H. Shortliffe, F.E. Stockdale, and P.L. Miller, *A Computational Model of Reasoning from the Clinical Literature* **AI Magazine**, 1989. (Spring): p. 49-57.

220. Rieger, C., *An Organization of Knowledge for Problem Solving and Language Comprehension* **Artificial Intelligence**, 1976. **7**(2).

221. Rieger, C. and M. Grinberg. *The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms* in **IJCAI '77**. 1977.

222. Rosenbloom, P., J. Laird, J. McDermott, A. Newell, and E. Oruich, *R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture* **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 1985. **PAMI-7**(No. 5, September): p. 561-569.

223. Rosenbloom, P.S., J.E. Laird, and A. Newell, **The SOAR Papers**. 1993, The MIT Press.

224. Rothenfluh, T.E., J.H. Gennari, H. Erikson, A.R. Puetra, W. Tu, and M.A. Musen, *Reusable Ontologies, Knowledge-Acquisition Tools and Performance Systems: PROTEGE-II Solutions to Sisyphus-2.* in **Proceedings of the 8th AAAI-Sponsored Banff**

Knowledge Acquisition for Knowledge-Based Systems Workshop, B.R. Gaines and M. Musen, Editor. 1994, p. 43.1-43.30.

225. Rumbaugh, J., M. Blaha, P. W., F. Eddy, and W. Lorenson, **Object-Oriented Modelling and Design**. 1991, Englewood Cliffs, NJ: Prentice-Hall.

226. Runkel, J.T. and W.P. Birmingham, *Solving VT by Reuse,* in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 42.1-42.28.

227. Schreiber, A., P. Terpstra, P. Magni, and M. van Velzen, *Analysing and Implementing VT using CommonKADS,* in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 44.1-44.29.

228. Schreiber, A.T., B.J. Wielinga, and J.M. Akkermans, *Using KADS to Analyse Problem Solving Methods,* in **Formal Methods for Knowledge Modelling in the CommonKADS Methodology: A Compilation (KADS-EE/T1.2/TR/ECN/014/1.0)**, J. Balder and H. Akkermans, Editor. 1992, Netherlands Energy Research Foundation: p. 53-90.

229. Searle, J.R., *Minds, Brain, and Programs* **The Behavioural and Brain Sciences**, 1980. **3**: p. 417-457.

230. Searle, J.R., *John R. Searle replies* in *The New York Review of Books*. 1982, p. 57-58.

231. Searle, J.R., *The Myth of the Computer* in *The New York Review of Books*. 1982, p. 3-6.

232. Selman, B. and H.J. Levesque. *Abductive and Default Reasoning: a Computational Core* in **AAAI '90**. 1990.

233. Shaw, M.L.G. *Validation in a Knowledge Acquisition System with Multiple Experts* in **Proceedings of the International Conference on Fifth Generation Computer Systems**. 1988.

234. Shaw, M.L.G. and B.R. Gaines. *Repgrid-net: Combining Conceptual Modelling with Electronic Mail to Provide Decision Support* in **7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1992. Banff, Canada:

235. Silverman, B.G., *Critiquing Human Judgment Using Knowledge-Acquisition Systems* **AI Magazine**, 1990. (Fall): p. 60-79.

236. Silverman, B.G., *Building a Better Critic: Recent Empirical Results* **IEEE Expert**, 1992. (April): p. 18-25.

237. Silverman, B.G., *Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers* **Communications of the ACM**, 1992. **35**(4): p. 106-127.

238. Simon, H.A., *On the Definition of the Causal Relationship* **Journal Philosophy**, 1952. **49**: p. 517-528.

239. Smythe, G.A., *Hypothalamic noradrenergic activation of stress-induced Adrenocorticotropin (ACTH) release: Effects of acute and chronic dexamethasone pre-treatment in the rat,* **Exp. Clin. Endocrinol. (Life Sci. Adv.)**, 1987. (6): p. 141-144.

240. Smythe, G.A., *Brain-hypothalmus, Pituitary and the Endocrine Pancreas,* in **The Endocrine Pancreas**, S. E., Editor. 1989, Raven Press: New York.

241. Smythe, G.A. *Concerning errors in the Smythe '87 model.* . 1992. Personal communication.

242. Smythe, G.A., M.W. Duncam, J.E. Bradsahw, W.Y. Cai, and R.G. Symons, *Control of Growth Hormone Secretion: Hypothalmic Dopamine, Norepinephrine and Serotonin Levels and Metabolism in Three Hyposomatrophic Rat Models and in Normal Rats* **Endocrinology**, 1982. **110**(2): p. 376-383.

243. Soloway, E., J. Bachant, and K. Jensen. *Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base* in **AAAI '87**. 1987.

244. Spohrer, J.C. and C.K. Riesbeck, *Reasoning-driven Memory Modifications in the Economics Domain*. 1984, Yale University.

245. Stark, M., *Impacts of Object-Oriented Technologies: Seven Years of Software Engineering* **J. Systems Software**, 1993. **23**: p. 163-169.

246. Steels, L., *Components of Expertise* **AI Magazine**, 1990. **11**(2): p. 29-49.

247. Stefik, M., J. Aikins, R. Balzer, J. Benoit, L. Birnhaum, F. Hayes-Roth, and E. Sacerdoti, *The Organisation of Expert Systems, A Tutorial* **Artificial Intelligence**, 1982. **18**: p. 135-127.

248. Steier, D.M., *CYPRESS-Soar: A Case Study in Search and Learning in Algorithm Design*, in **The Soar Papers**, P.S. Rosenbloom, J.E. Laird, and A. Newell, Editor. 1987, MIT Press: p. 533-536.

249. Suwa, M., A.C. Scott, and E.H. Shortliffe, *An Approach to Verifying Completeness and Consistency in a Rule-based Expert System*. 1982, Department of Computer Science, University of Stanford:

250. Swanson, D.R., *Medical Knowledge as a potential source of new knowledge* **Bull Med Libr Assoc**, 1990. **78**(1): p. 29-37.

251. Tansley, D.S.W. and C.C. Hayball, **Knowledge-Based Systems Analysis and Design**. 1993, Prentice-Hall. 528.

252. Top, J.L., *Bond graphs for causal explanations*, in **Bond Graphs for Engineers**, P.C. Breedveld and G. Dauphin-Tanguy, Editor. 1992, Elsevier Science Publications B.V. (North-Holland): p. 313-321.

253. Top, J.L. and J.M. Akkermans. *Computational and Physical Causality* in **IJCAI '91**. 1991. Sydney:

254. Tu, S., Y. Shahar, J. Dawes, J. Winkles, A. Puetra, and M. Musen. *A Problem-Solving Model of Episodic Skeletal-Plan Refinement* in **6th BANFF Knowledge Acquisition for Knowledge-Based Systems Workshop**. 1991. Canada:

255. Van de Brug, A., J. Bachant, and J. McDermott, *The Taming of R1* **IEEE Expert**, 1986. (Fall): p. 33-39.

256. Wang, P., *From Inheritance Relation to Non-Axiomatic Logic*. 1993, Center for Research on Concepts and Cognition, Indiana University, Bloomington, Indiana, 1993:

257. Warren, D.S., *Memoing for Logic Programs* **Communications of the ACM**, 1992. **35**(March): p. 93-101.

258. Weiss, S.M., C.A. Kulikowski, and S. Amarel, *A Model-Based Method for Computer-Aided Medical Decision-Making* **Artificial Intelligence**, 1978. **11**(145-172).

259. Wick, M.R. and W.B. Thompson, *Reconstructive Expert System Explanation* **Artificial Intelligence**, 1992. **54**: p. 33-70.

260. Wielinga, B.J., A.T. Schreiber, and J.A. Breuker, *KADS: a modelling approach to knowledge engineering.* **Knowledge Acquisition**, 1992. **4**(1): p. 1-162.

261. Williams, B.C. and J. DeKleer, *Qualitative Reasoning about Physical Systems: a Return to Roots* **Artificial Intelligence**, 1991. **51**: p. 1-9.

262. Wirfs-Brock, R.J., B. Wilkerson, and L. Weiner, **Designing Object-Oriented Software**. 1990, Englewood Cliffs, NJ: Prentice-Hall.

263. Yip, K.M. *Extracting Qualitative Dynamics from Numerical Experiments* in **AAAI '87**. 1987.

264. Yip, K.M. *Generating Global behaviours Using Deep Knowledge of Local Dynamics.* in **AAAI '88**. 1988.

265. Yip, K.M., *Understanding Complex Dynamics by Visual and Symbolic Reasoning* **Artificial Intelligence**, 1991. **51**: p. 179-221.

266. Yost, G., *Implementing the Sisyphus-93 Task Using Soar/TAQL*, in **Proceedings of the 8th AAAI-Sponsored Banff Knowledge Acquisition for Knowledge-Based Systems Workshop**, B.R. Gaines and M. Musen, Editor. 1994, p. 46.1-46.22.

267. Yost, G.R., *Configuring Elevator Systems*. 1992, Digital Equipment Co., Marlboro, Massachusetts:

268. Yost, G.R. and A. Newell. *A Problem Space Approach to Expert System Specification* in **IJCAI '89**. 1989.

269. Yu, V.L., L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J.F. Hanigan, R.L. Blum, B.G. Buchanan, and S.N. Cohen, *Antimicrobial Selection by a Computer: a Blinded Evaluation by Infectious Disease Experts* **Journal of American Medical Association**, 1979. **242**: p. 1279-1282.

270. Zdrahal, Z. and E. Motta, *An In-Depth Analysis of Propose & Revise Problem Solving Methods,* in **Proceedings of the Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop: JKAW '94**, R. Mizoguchi, *et al.*, Editor. 1994, Japanese AI Society:

271. Zlatareva, N., *CTMS: A General Framework for Plausible Reasoning* **International Journal of Expert Systems**, 1992. **5**(3): p. 229-247.

272. Zlatareva, N. *Distributed Verification and Automated Generation of Test Cases* in **IJCAI '93 workshop on Validation, Verification and Test of KBs**. 1993. Chambery, France:

273. Zlatereva, N., *Truth Maintenance Systems and Their Application for Verifying Expert System Knowledge Bases* **Artificial Intelligence Review**, 1992. **6.**

274. Zlatereva, N. and A. Preece, *State of the Art in Automated Validation of Knowledge-Based Systems* **Expert Systems with Applications**, 1994. **7**(2): p. 151-167.