Ripple-Down Rationality: A Framework for Maintaining PSMs

Tim Menzies and Ashesh Mahidadia *
Artificial Intelligence Department, School of Computer Science & Engineering,
University of NSW, Sydney, Australia, 2052;
Email: {timm|ashesh}@cse.unsw.edu.au; URL: http://www.cse.unsw.edu.au/~{timm|ashesh}

April 22, 1997

Abstract

Knowledge-level (KL) modeling can be characterised as *theory subset extraction* where the extracted subset is consistent and relevant to some problem. Theory subset extraction is a synonym for Newell's principle of rationality, Clancey's model construction operators, and Breuker's components of expert solutions. In an abductive framework, a PSM is the extraction controller and is represented by a suite of BEST inference assessment operators. Each BEST operator is a single-classification expert system which accepts or culls a possible inference. PSMs can therefore be maintained by ripple-down-rules, a technique for maintaining single-classification expert systems.

1 Introduction

Newell's knowledge-level (KL) approach modeled intelligence [37] as a search for appropriate *operators* that convert some *current state* to a *goal state*. Domain-specific knowledge are used to select the operators according to *the principle of rationality*; i.e. an intelligent agent will select an operator which its knowledge tells it will lead the achievement of some of its goals.

We can characterise expert systems maintenance as the controlled revision of the state space and the rationality operators. The state space is modeled as a directed graph of literals. Inference is controlled by an abductive (§2.1) inference engine which select portions of the graph which are relevant to a problem. Within the abductive inference engine are sets of domain-specific BEST assessment operators which implement the principle of rationality; i.e. they select the preferred inference(s) from the set of available inferences. In an abductive knowledge-level framework, problems solving methods (PSMs) are sets of BEST operators (§2.2). Such abductive PSMs operationalise Clancey's system model construction operators which build a situation-specific model and Breuker's components of solutions (§2.2.8). On a continuum between standard PSMs and systems like SOAR, this abductive PSM approach is closer to SOAR than (e.g.) KADS (§2.3).

The key observation which sparked this paper as follows. Each such BEST operator is a single classification expert system which classifies proposed inferences as cull or accept. Ripple-down-rules (RDR) are a technique that works well for maintaining single classification expert systems (§3.1). While they are poor at representing KADS-style PSMs (§3.2), they could be used in an abductive framework to maintain the BEST operators (§4). That is, we could maintain abductive PSMs with RDR. Two premises of this approach is that the knowledge base includes inference preference criteria (§4.1) and that we have some mechanism for maintaining the functions used in the BEST operators (e.g. using ripple-down-functions or RDF, §4.2). We call this combination of abduction and ripples for KL modeling *ripple-down-rationality* or RD-RA. RD-RA has several advantages over current PSM technology including maintenance, the ability to compare different expert system frameworks, and support for conflict resolution in requirements modeling (§5).

*Workshop on Problem-solving Methods for Knowledge-based Systems in Connection with the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97) Nagoya, Japan, during August 23-25, 1997

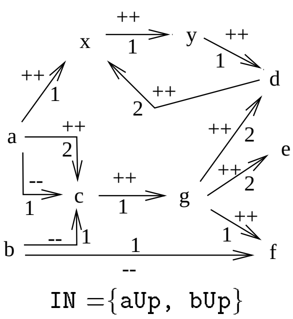
2 Abduction

This section gives our standard short overview on abduction (§2.1); discusses its application to knowledge level modeling (§2.2); contrasts abductive KL with standard KL approaches (§2.3); and comments on the computational complexity of abduction (§2.4). This section summarises material from other papers [33, 34].

2.1 An Overview of Abduction

Elsewhere [34], we have given a detailed overview of abductive research. Here, we offer an approximate characterisation of abduction as the search for consistent subsets of some background theory that are relevant for achieving some goal. If multiple such subsets can be generated, then a BEST assessment operator selects the preferred subset(s) (or *worlds*, W). For example, consider our graph-theoretic HT4 abductive inference engine [33,34,36]. HT4 determines what OUTput goals can be reached from using the INputs shown in a knowledge base like Figure 1. In that Figure, $\texttt{x}\overset{++}{\rightarrow}\texttt{y}$ denotes that y being up or down can be explained by x being up or down respectively and $\texttt{x}\overset{--}{\rightarrow}\texttt{y}$ denotes that y being up or down could be explained by x being down or up respectively. Observe the apparent conflict in the middle of Figure 1 on the left-hand-side: author 1 believes $\texttt{a}\overset{--}{\rightarrow}\texttt{c}$ while author 2 believes $\texttt{a}\overset{++}{\rightarrow}\texttt{c}$. We will return to this apparent conflict later (§5).

HT4 can find the following proofs P connecting OUTs to INs:



IN ={aUp, bUp}
OUT={dUp, eUp, fDown}

Figure 1: A knowledge base. Edges are labeled with the author of that edge; e.g. author 1 and 2.

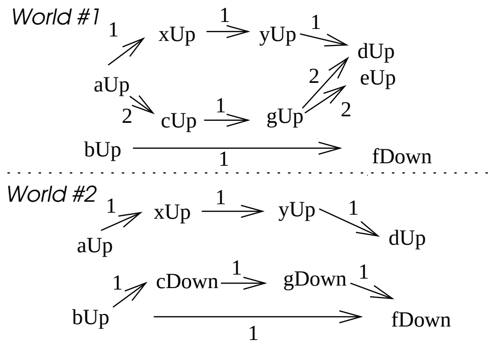


Figure 2: Two generated worlds from Figure 1. Edges are labeled with the author of that edge; e.g. author 1 and 2.

P[1] = aUp → xUp → yUp → dUp;
P[2] = aUp → cUp → gUp → dUp;
P[3] = aUp → cUp → gUp → eUp;
P[4] = bUp → cDown → gDown → fDown;
P[5] = bUp → fDown.

These proofs may contain assumptions, i.e. literals that are not known FACTS. Continuing the example of Figure 1, if FACTS is the union of IN and OUT, then {xUp, yUp, cUp, cDown, gUp, gDown} are assumptions. If we can't believe that a variable can go up and down simultaneously, then we can declare {cUp, cDown, gUp, gDown} to be conflicting (denoted A.c). Figure 1 shows us that g is fully dependent on c. Hence the key conflicting assumptions are {cUp, cDown} (denoted *base controversial assumptions* or A.b). We can used A.b to find consistent belief sets called worlds W. A proof P[i] is in W[j] if that proof does not conflict with the environment ENV[j] (an environment is a maximal consistent subset of A.b). In our example, ENV[1]={cUp} and ENV[2]={cDown}[1]. Hence, W[1]={P[1], P[2], P[3], P[5]} and W[2]={P[1] P[4] P[5]} (see Figure 2). Note that while inconsistencies may be detectable within the knowledge base, each world is guaranteed to be consistent.

[1] The connection of HT4 to DeKleer's ATMS system [10] is explored elsewhere [34]

2.2 Abduction and the Knowledge Level

Abduction generates worlds. Which is the `BEST` world? We have argued previously [34] that:

1. The answer is problem-specific.
2. `KL` modeling is a synonym for the careful selection of the `BEST` assessment operators.

To demonstrate these two points, consider an agent watching the traversal of a search space represented as a directed graph connecting literals. What information does our agent need to to decide if a possible inference is `BEST`? We define five levels of `BEST`:

Level 1 *vertex-level* `BEST`: If our agent can only "see" one edge ahead of the current search, then they could make a *vertex-level* `BEST` decision. For example, if we are implementing a MYCIN-style certainty factor kludge [3], then our agent could decide to avoid all edges with a certainty factor of less than 0.2. That is, those edges are `cull`ed since they are not `BEST`.

Level 2 *proof-level* `BEST`: If our agent can "see" what the *vertex-level* agent can see *and* can query the current proof tree up to the current edge, then it could make a *proof-level* choice. Our proof-level agent could `cull` an edge in the following three cases:

1. The new edge means that the proof would contain "too many assumptions" (defined according to some domain-specific criteria).
2. The edge contains a literal that is incompatible with literals currently on the proof.
3. The edge would introduce a loop in the current proof.

Level 3 *proofs-level* `BEST`: If our agent can "see" what the *proof-level* agent can see *and* can query all the other proofs generated up until now, then it could make a *proofs-level* choice. For example, classic AI search algorithms such as best-first and beam-first search[2] are proofs-level `BEST` operators.

[2] In best-first, the cost of traversal to this point is added to a heuristic guess on what it would cost to use each possible edge to continue on to some goal. The edges are then sorted by increasing combined cost. The search then proceeds in that order. In beam-first, after the sort, only the top N items are explored where N is the "width" of the beam search [41].

Level 4 *world-level* `BEST`: Suppose our agent "sees" that a generated world explains all of the `OUT` set. It could declare this world to be `BEST` and halt the generation of any other worlds.

Level 5 *worlds-level* `BEST`: When our agent can "see" multiple worlds, it could make a comparative evaluation of the different worlds.

A variety of knowledge-level tasks can be implemented via an appropriate selection of `BEST` assessment operators [34] and careful construction of the `IN`put and `OUT`put sets: prediction (§2.2.1), diagnosis and probing (§2.2.2), classification (§2.2.3), explanation and tutoring (§2.2.4), qualitative reasoning (§2.2.5), planning and monitoring (§2.2.6), validation (§2.2.7), amongst other (§2.2.8).

2.2.1 Prediction

Prediction is the process of seeing what will follow from some events `IN`. Given a theory represented as a directed graph with vertices `V`, then we can find all the non-input vertices reachable from `IN` by making $\texttt{OUT} \subseteq \texttt{V} - \texttt{IN}$ (note that `IN` and `OUT` are just subsets of `V`). A efficient case for prediction is when `IN` is smaller than all the roots of the graph and some *interesting subset* of the vertices have been identified as possible reportable outputs (i.e. $\texttt{OUT} \subset \texttt{V} - \texttt{IN}$).

2.2.2 Diagnosis and Probing

It is well-known that diagnosis is an abductive process [8, 48, 52]. Parsimonious *set-covering diagnosis* [51] uses a `BEST` that favors worlds that explain the most things, with the smallest number of diseases (i.e. maximise $\texttt{W[x]} \cap \texttt{OUT}$ and minimise $\texttt{W[x]} \cap \texttt{IN}$).

The opposite of set-covering diagnosis is *consistency-based diagnosis* [8, 53] where all worlds consistent with the current observations are generated. Computationally, this is equivalent to *prediction* (§2.2.1), with $\texttt{OUT} = \texttt{V} - \texttt{IN}$.

In Reiter's variant on consistency-based diagnosis [53], all predicates relating to the behaviour of a theory component `V[x]` assume a test that `V[x]` in not acting `AB`normally; i.e. $\neg\texttt{AB(V[x])}$. `BEST.REITER` is to favour the worlds that contain the least number of `AB` assumptions.

A related task to diagnosis is *probing*. When exploring different diagnosis, an intelligent selection of tests (probes) can maximise the information gain while reducing the testing cost [11]. In this abductive framework, we would know to favor probes of `A.b` over probes of `A.c` over probes of non-controversial assumptions.

2.2.3 Classification

Classification is just a special case of prediction with the *interesting subset* set to the vertices representing the possible classifications. The and-or graph of a classification theory would include edges (i) from class attributes to the proposition that some class is true; and (ii) from sub-classes to super-classes (e.g. if `emu` then `bird`). `BEST.CLASSIFICATION` could favors the worlds that include the most-specific classes [46] (e.g. `emu` is better than `bird`).

2.2.4 Explanation and Tutoring

If we have a profile of our users comprising vertices familiar to the user and the edges representing processes that the user is aware of, then we can build an *explanation* and *tutoring system*. `BEST.EXPLANATION` could favors the worlds with the largest intersection to this user profile. That is, we return the world(s) that the user is most likely to understand. We base this explanation proposal on analogous work by Paris [43].

Further, suppose we can assess that the `BEST` explainable world was somehow sub-optimum. We could then make a entry is some log of teaching goals that we need to educate our user about the edges which are not in their `BEST` explainable world but are in other, more optimum, worlds.

2.2.5 Qualitative Reasoning

The above abductive inference engine was developed from a *qualitative reasoning* algorithm for neuroendocrinology [17]. A fundamental property of such systems is their indeterminacy which generate alternative values for variables. These alternatives and their consequences must be considered separately. Abduction can maintain these alternatives in separate worlds.

2.2.6 Planning and Monitoring

Planning is the search for a set of actions that convert some current state into a goal state. Given a set of actions, we could partially evaluate them into the dependency graph they propose between literals. `BEST.PLANNING` could favor the world(s) with the least cost (the cost of a world is the maximum cost of the proofs in that world). If we augment each edge with the identifier of the action(s) that generated it, then we could report the `BEST` worlds as the union of the actions that generated the `BEST` worlds.

Once generated, the `BEST` planning worlds could be passed to a *monitoring* system. As new information comes to light, some of these assumptions made by our planner will prove to be invalid. Hence, some of our worlds (a.k.a. plans) will also be invalid. The remaining plans represent the space of possible ways to achieve the desired goals in the current situation.

2.2.7 Validation

Validation: For example, let the *cover* of a world be its overlap with the `OUT` set. The cover of `W[1]` and `OUT` in Figure 2 is {`dUp,eUp,fDown`} and the cover `W[2]` and `OUT` is {`dUp,fDown`}; i.e. `W[1].cover`=3=100% and `W[2].cover`=2=67%. The maximum cover is 100%; i.e. their exist a set of assumptions ({`cUp`}) which let us explain all of `OUT`. A worlds-level `BEST` operator that returns the world with maximum cover is a validation procedure which answers the following question: "how much of the known behaviour of X can be reproduced by out model of X?". This abductive validation procedure has faulted theories published in the international peer-reviewed literature. Interesting, we have found these faults using the data published to support those theories [36]. We have argued elsewhere [33,34] that this is the non-naive implementation of KBS validation since it handles three certain interesting cases:

1. If not all variables in the theory are measured, then this procedure can still validate the theory. The procedure will take the necessary assumptions to prove some member of `OUT` and mutually exclusive assumptions are managed in separate worlds.

2. If a theory is globally inconsistent, but contains local portions that are consistent and useful for explaining some behaviour, we can find those portions.

3. In the situation where no current theory explains all known behaviour, competing theories can be assessed by the extent to which they cover known behaviour. Theory X is definitely better than theory Y if theory X explains far more behaviour than theory Y.

2.2.8 Others

We believe that abduction provides a comprehensive picture of declarative knowledge-based systems (KBS) inference. Apart from the problem solving methods discussed here, we also believe that abduction is a useful framework for intelligent decision support systems [29], diagrammatic reasoning [35], single-user knowledge acquisition, and multiple-expert knowledge acquisition [31]. Further, abduction could model certain interesting features of human cognition [32]. Others argue elsewhere that abduction is also a framework for natural-language processing [40], visual pattern recognition [47], analogical reasoning [16], financial reasoning [21], machine learning [22] and case-based reasoning [25].

We have argued [34] that abduction is such a general procedure since it operationalises the *theory subset extraction* process that Breuker [2] and Clancey [5, 6] argue is at the core of expert systems. Clancey offers a two-layered extraction process (qualitative model to situation-specific model) while Breuker offers a four-layered view of the components of solutions in an expert system (generic domain model to case model to conclusion to argument structure). Our approach is more general than Clancey's since it makes explicit certain assumptions which are only tacit in Clancey's approach. For example, Clancey assumes that the best world uses the fewest number of `IN`s [5, p331]. We have shown above that this is not universally true. As to Breuker's proposal, his components of solutions sounds to us like three recursive calls to a single inference procedure. In his view, all expert system tasks use an *argument structure* which is extracted from a *conclusion* which is in turn extracted from a *case model* which is in turn extracted from a *generic domain model*. Note that, in all cases, each sub-component is generated by extracting a relevant subset of some background theory to generate a new theory (i.e. abduction).

2.3 `KL-A` and `KL-B`

This section discusses two variants on the `KL` approach: `KL-A` (e.g. SOAR) and `KL-B` (e.g. KADS).

Newell's own exploration of the `KL` lead to a general rule-based language called SOAR [56] which was the basis for the problem-space computational model (PSCM) [61]. Programming SOAR using the PSCM involves the consideration of multiple, nested problem spaces. Whenever a "don't know what to do" state is reached, a new problem space is forked to solve that problem. Newell concluded that the PSCM was the bridge between SOAR and true `KL` modeling [38, 39].

We distinguish between PSCM (which we term `KL-A`) and `KL-B`, a `KL`-modeling variant which groups together a set of authors who argue for basically the same technique; e.g. Clancey's model construction operators [6], Steels' components of expertise [58], Chandrasekaran's task analysis, SPARK/ BURN/ FIREFIGHTER [27] and KADS [60][3]. The fundamental premise of `KL-B` is that a knowledge base should be divided into domain-specific facts and domain-independent problem solving methods. Such problem-solving strategies are implicit in `KL-A`. The observation that a PSCM system is performing (e.g.) classification is a user-interpretation of a single inference procedure (operator selection over a problem space traversal) [61].

Like all single-world logic devices, SOAR `culls` possibilities at the local-choice level. By definition, local-choice systems cannot delay their decisions on options till later in the computation when more information is available. For example, 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 modeled control decisions, while the RAINBOW abductive inference engine generated possible designs [59]. In our approach, a `KL` agent has the facility to delay choice decisions till the path, paths, world, or worlds-level.

[3] See the *Related Work* section of [60] for a discussion of the differences in these techniques

Our multiple worlds device can explicitly represent all the options. Hence, a `KL` agent is free to reflect more fully over the choices.

Our abductive PSM approach is clearer nearer to `KL-A` than `KL-B`. Like SOAR, our framework uses a single inference procedure (abduction). The connection of our approach to `KL-B` in general and KADS in particular is explored elsewhere [28, 34]. In summary, KADS offers a variety of designs for a range of inference devices (e.g. [1]) for expert systems. We prefer the use of a single inference device (abduction). Our research goal is (i) the description of the minimal architecture necessary to `KL` abduction; and (ii) a maintenance framework for that minimal architecture.

2.4 A Note on Complexity

One drawback with abduction is that it is slow. Selman & Levesque show that even when only one abductive explanation is required and the theory is restricted to be acyclic, then abduction is NP-hard [57]. Bylander *et. al.* make a similar pessimistic conclusion [4]. Computationally tractable abductive inference algorithms (e.g. [4, 15]) typically make restrictive assumptions about the nature of the theory or the available data. Such techniques are not applicable to arbitrary theories. Therefore, it is reasonable to question the practicality of abduction for medium to large theories.

While the complexity of `BEST` is operator specific, we can make some general statements about the computational cost of `BEST`. At a procedural level, assessment operators at level `i` can be used to restrict the search space for operators at level `i+1`. *Vertex* or *proof-level* assessment reduce the complexity of the search space traversal (since not all paths are explored). *Worlds-level* assessment is a search through the entire space that could be relevant to a certain task. Hence, for fast runtimes, do not use worlds-level assessment. However, for some tasks (e.g. the validation task §2.2.7) worlds-level assessment is unavoidable. Elsewhere we have shown that level 5 (worlds-level) abduction is practical for the sizes of knowledge bases we see in contemporary practice [33]. Hence, we are confident that an abductive system that uses some lower level assessment operators will operate in acceptable runtimes.

3 Ripple-Down-Rules

We argued above (§2.2) that a variety of knowledge level tasks can be expressed a single abductive inference procedure, plus some customisable `BEST` inference assessment operators. We will argue below (§4) that we can maintain PSMs by maintaining the suite of `BEST` operators. That section will use a technique called ripple-down-rules (or `RDR`). This section describes `RDR`. We focus only on single classification `RDR` since that is all that is required for `RD-RA`. Multiple classification `RDR` is discussed elsewhere [24].

3.1 An Overview of Single Classification RDR

Standard software engineering and knowledge engineering typically assumes that prior to building a system, an extensive analysis stage develops a design for the system. Compton reports experiments with a completely reversed approach. In `RDR`, there is no analysis period. KA in an `RDR` system consists only of fixing faulty rules using an *unless* patch attached at the end of a rule condition. Patches are themselves rules which can be recursively patched. Experts can never re-organise the tree; they can only continue to patch their patches. If a new case motivates a new patch, that this case is stored with the new patch. Such cases are called *cornerstone cases.* Compton argues that these (`RDR`) trees model the context of knowledge acquisition. When a case is processed by an `RDR` tree, its context is the set of cases in the patches exercised by the new case. When looking for new patches, experts can only choose from the *difference* of the attributes in the current case and the attribute value tests (called *features*) exercised down to the current faulty rule.

For example, Figure 3 shows the rule `if a&b then x1` patched several times. At runtime, the final conclusion is the conclusion of the last satisfied rule. At maintenance time, when fixing deficient knowledge, an *unless* patch is added beneath the incorrectly last-satisfied rule. Only the *logic delta* is added in the new rule since the system cannot get to this rule without first satisfying the logic from the root to this rule. So, in Figure 3, if `x2` is the correct conclusion when `a&b&c` is true, but incorrect when `c` is false, we add the logic delta `c` to a patch rule

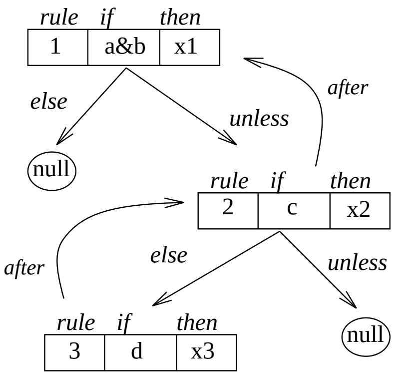


Figure 3: A RDR knowledge base

RDR rule	If	Then
1	a & b & not c	x1
2	a & b & c	x2
3	a & b & not c & d	x3

Figure 4: The flattened rule set of Figure 3.

in the *unless* branch beneath `rdr1`.

RDR applies the maxim *si fractum non sit, noli id reficere*[4] and "freezes" knowledge that has proven satisfactory in the past. Logic deltas are only added to new knowledge. This *frozen knowledge principle* simplifies maintenance. To see this, consider the *flattened rule set* we can generate from Figure 3 in Figure 4. In the flattened rule set, one rule is generated for each RDR path. Most rule-based expert systems would maintain their knowledge in flattened rule sets such as Figure 4. In such a conventional rule-based expert system, a patch can extend over many rules. Repeating our above example, the patch on the `x1` error requires an edit to one RDR rule `rdr1` and the creation of another RDR rule `rdr2`. Further, the new logic refers to `c` which is a new concept that must be propagated down to all related rules (e.g. `rdr3`). The more related rules, the more edits. As the knowledge base grows, so to does the number of related rules and the number of rules that may require editing after a change [20]. By contrast, in RDR, existing knowledge is frozen and we only *extend* the knowledge base.

[4] If it ain't broke, don't fix it.

```
RDR         ::= RDRnode | RDFnode
RDRnode     ::= rdr   _id
                [after RDR]
                if     Condition
                       then _conclusion
                       since Cornerstone
                       [unless RDR]
                [else  RDR]
Condition   ::= [not] Feature  {and Condition}+
               | [not] Feature  {or Condition}+
Feature     ::= Oracle says Comp than _value
Oracle      ::= NamedOracle(_argument {,_argument})
NamedOracle ::= _namedOracle
Function    ::= toCall  Oracle
                perform [cache] _body
Comp        ::= >=|>|<>|=|<|<=
Cornerstone ::= Feature {and Feature}
```

Figure 5: BNF for RDR. RDFnodes are discussed later (§4.2).

The BNF of an RDR is shown in Figure 5[5]. **After**, **unless** and **else** links are optional. Only patched RDRrules will get `else` and `unless` links. If an `RDRnode` points to another `RDRnode`, then we may optionally store a `after` link as a back pointer (for efficiency reasons). Note a `Function` is a pair of `<Oracle, body>` where the `body` is called as a side effect of passing `arguments` to a `NamedOracle`. If the keyword `cache` is included in the `Function` definition, then the results of calling the `body` the first time is simply returned if ever it is called again.

RDR trees are a very low-level representation. RDR rules cannot assert facts that other RDR rules can use. In no way can a RDR tree be called a model in a KL sense. Further, the RDR formalism makes no commitment to tree structures that are optimal. An RDR tree can contain repeat tests, redundant knowledge, and its sub-trees can overlap each other semantically. Despite these apparent drawbacks, RDR has produced large working expert systems in routine daily use. In practice the RDR trees are only

[5] In the BNF variant used here, words starting in lower case represent terminals. Terminals that are variables start with an underscore; e.g. `if` is a non-variable terminal while `_name` is a variable storing some string entered by the user. Words starting with upper case are nonterminals. Nonterminals are expanded with the form `NonTerm ::= Rhs`. A `Rhs` can use the following symbols. Square brackets `[]` surround optional items; a vertical line (`|`) separates alternatives; curly brackets {} surround items that can repeat many *zero* or more times; and curly brackets followed by a plus {}+ surround items that can repeat *one* or more times.

twice as big as the optimum tree [19] and runtimes have never been an issue. It may be somewhat misguided to attempt to optimise an RDR tree to (e.g.) remove the redundancies or separate the overlaps. The important feature of an RDR tree is that is it optimised for maintenance. Alternative representations may run faster, but incurs the penalty of more complicated maintenance.

In practice, RDR appears to work very well, at least for single classification problems. For example, the PIERS system at St. Vincent's Hospital, Sydney, models 20% of human biochemistry sufficiently well to make diagnoses that are 99% accurate [50]. RDR has succeeded in domains where previous attempts, based on much higher-level constructs, never made it out of the prototype stage [44]. Further, while large expert systems are notoriously hard to maintain [9], the no-model approach of RDR has never encountered maintenance problems. System development blends seamlessly with system maintenance since the only activity that the RDR interface permits is patching faulty rules in the context of the last error. For a 2000-rule RDR system, maintenance is very simple (a total of a few minutes each day). Compton argues that his process of "patching in the context of error" is a more realistic KA approach than assuming that a human analyst will behave in a perfectly rational way to create some initial correct design [7].

3.2 Limits to RDR

This section argues that, in terms of RD-RA, RDR has two significant limits: representing functions performing the feature extraction and representing PSMs.

Representing feature extractors: An RDR rule queries values extracted from the functions performing the feature extraction which summarise the environment. For example, an RDR rule might say `if age=old then...` where `age` is a `Function` that assigns labels such as `old` to continuous values such as `age`. The change control for these functions is not controlled by logic patches in a RDR tree; i.e. RDR does not address the issue of maintaining functions (though see Preston's work [49] for an interesting approach to customising feature extraction functions).

Representing PSMs: Before a RDR tree executes, the feature extractor `Functions` find all the features in a case. The delta logic is constructed via computing the difference between the case features and the features found in the rules along the path to the faulty rule. Consequently, RDR is focused on the details of the specific case at hand. PSMs may not be expressible with respect to the specific problem at hand. For example, consider an RDR tree maintaining taxi driver knowledge. Our student taxi driver may learn many tricks about navigating from Sydney to Canberra through specific streets. However, in an RDR framework, she may never learn the generalised PSM: "open the map, find your current location, find your destination, compute the shortest distance path from here to there". More generally, to date, the `Functions` called in `RDR Conditions` do not access the meta-level reasoning used in problem solving.

These representational problems are a particular drawback for RD-RA. The abductive PSMs described in §2.2 are not defined with respect to the particulars of the problem at hand. Rather, the BEST operators are defined with respect to meta-level reflection on the current status of the inference. To resolve these problems, we will need a library of feature extractor `Functions` that can access the internal data structures of the inference engine. Further, we will need to offer a maintenance strategy for those `Functions`. We return to this point below when we discuss ripple-down-function (§4.2).

4 An Architecture for Ripple-Down-Rationality

This section RD-RA, our marriage of abduction to ripples to KL modeling. After considering an extended example of our framework (§4.1), we will see that we need some tool for maintaining the rule conditions (§4.2).

4.1 RD-RA: An Example

Suppose we are maintaining some `version` of a knowledge base `KB` such as Figure 6. This knowledge base contains a set of statements, each tagged with their `author`.

```
KB          ::= _version contains {Statement}+
Statement   ::= _id : _author says _contents
```

```
sydneyDiseases contains 1,2.

1  : timm says
if      suburb='bondi' and
        temperature > 40 and
        pain.location = 'tummy' and
        observed = 'sand rash on tummy'
then    conclusion = 'sand allergy secondary to
                       too much surfing'.

2 : ashesh says
if      temperature > 40 and
        pain.location = 'tummy' and
        observed = 'sand rash on tummy'
then    conclusion = 'too much groveling'.
```

Figure 6: A knowledge base.

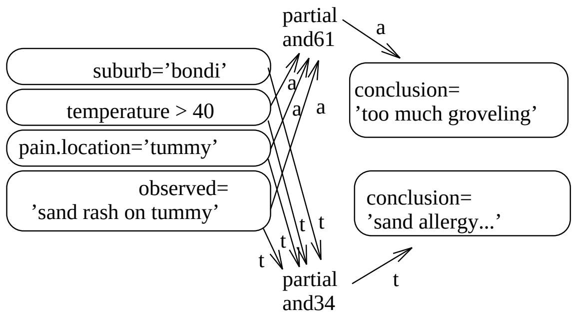


Figure 8: A search space generated from Figure 6. Edges are labeled with their authors; e.g. t=`timm` and a=`ashesh`.

```
Theory        ::= _version contains Graph
                  for {Goal}+ using PSM
PSM           ::= RDR
Goal          ::= input {Vertex}+ output {Vertex}+
Graph         ::= {Vertex}+ {Edge}
Edge          ::= from Vertex to Vertex
Vertex        ::= And | Or
Or            ::= from Vertex Feature
And           ::= from Vertex {and Vertex}
                  type AndType
AndType       ::= full | partial
```

Figure 7: BNF for `RD-RA`

We make no commitment to the form of the `contents`. However, we do require that some `compiler()` can convert that `KB` into a `Theory`[6] comprising:

- A directed graph connecting `Feature`s.
- A library of known/desired behaviour (called the `Goals`);
- `Prefer`ence criteria for deciding between different goals;

The BNF of a `Theory` is shown in Figure 7. The PSM is modeled as an `RDR` tree controlling the `BEST` operators. The `IN`put and `OUT`put sets are generated from the `Goals`. An `Or Vertex` can be traversed if any of its parents can be traversed. As we cross an `Or Vertex`, its `Feature` test is applied which, in turn, may executes a `Function body` (recall Figure 5). That is, procedural side-effects can be coded into `Or Vertex` traversal. An `And Vertex` can only be traversed if some its parents can all be traversed. There are two `AndTypes`. A `full And` requires all its parents to be traversable. A `partial And` will try all its parents but only strictly needs one parent to be traversable.

The BNF of Figure 7 is compatible with at least with two common styles of expert systems: frames and rules. Frame systems can be implemented using `Partial And`s connecting slots to frame names. That is, if we can match slot contents, then we can reason back to a frame. Rules can be compiled into an `and-or` graph connecting rule left-hand-sides to rule right-had-sides. For example, the rules of Figure 6 could be compiled into the graph of Figure 8. Note that:

- If the rule right-hand-sides are used in other rules, then these connect to other rule left-hand-sides; i.e. graphs like Figure 6 would be deeper that just 1 edge.
- Observe the use of `PartialAnd`s in Figure 6. If we have only some of the evidence contributing to (e.g.) `tooMuchGroveling`, then with `PartialAnd`s we can still explore this potential conclusion.

Figure 9 shows some `Prefer`ence criteria expressed in `RDR` format (recall Figure 5). The `Worlds,World,Proofs,Proof,Edge1,Edge2` variables are generated from the `Graph` search space

[6] i.e. `Theory=compile(KB)`

```
prefer(Worlds,World,Proofs,Proof,Edge1,Edge2);

rdr  0
if true
then true;

rdr 11 after 0
if      previouslySeenEdges(World1,X1) and
        previouslySeenEdges(World2,X2) and
        bigger1(X1,X2)
then    cull(World2);

rdr 34 after 0
if      seenBefore(Edge1) and
        not seenBefore(Edge2)
then    cull(Edge2);

bigger1(Big, Small) :- Big > Small.
```

Figure 9: `Prefer`ence criteria.

```
prefer(Worlds,World,Proofs,Proof,Edge1,Edge2);
...

rdr 67 after 34
if      timesSeen(Edge1,X1) and
        timesSeen(Edge2,X2) and
        bigger2(X1,X2)
then    cull(Edge1);

rdr 99 after 67
if      fromDoctor(Edge1) and
        fromMister(Edge2)
then    cull(Edge2).

bigger2(Big, Small) :- Big > 2*Small

% ashesh is slow at writing up his phd
mister(ashesh).
% timm teases ashesh about this
doctor(timm).
```

Figure 10: Refined `Prefer`ence criteria. Figure 9 is included into line 2 of this figure.

and are passed in at runtime by the abductive inference engine when it has some choices to evaluate. Suppose we were implementing an explanation system (§2.2.4). `Rdr0` is the obligatory `RDR` root node where `RDR` inference begins. `Rdr11` prefers worlds containing edges which have appeared in previous explanations; i.e. this system seeks to explain `OUT`puts using concepts the user is already familiar with. Note that `rdr11` is a worlds-level `BEST` and can be optimised using `rdr34`, a vertex-level `BEST` (§2.4). `Rdr34` performs a local cull of edges which have never been seen before.

Suppose our knowledge engineer enters Figure 9 and Figure 6 into the `RD-RA` environment, ran some cases, and found the conclusions unacceptable. Patches were then made to the PSM to generate Figure 10. `Rdr34` was been patched by another vertex-level `BEST` in `rdr67`. In the case where we have seen *both* `Edge1` and `Edge2`, but have seen `Edge2` twice as often as `Edge1`, then we cull `Edge1`. Lastly, in `rdr99`, it was decided that if a PhD graduate wrote `Edge1`, then it must be true! Hence, in the case of all the conditions of `rule1` and `rule2` being true in Figure 6, then we would conclude that surfing was the problem.

The example of Figure 9 illustrates some interesting points. Firstly, the flattened rule set from Figure 9 and Figure 10 would contain conditions containing combinations of `BEST` operators at different levels (in the case above: worlds and vertex-level). At the lowest-level of an abductive inference engine, the above preference criteria must be executed at every inference step. Clearly, we need an optimiser lest our abductive inference crawls to a halt. While clever optimisations are possible, we are currently exploring the following simple optimisation strategy. In the case of flattened preference rules containing a mix of operators at different levels, then the `BEST` operator is assigned to the highest level. This optimisation strategy should work well in the special case where the operators are all at the same low-level. In our example above, we would have vertex-level `BEST`s appearing in separate flattened rules to the worlds-level `BEST`s. The optimiser would assign the vertex-level `BEST`s to the local propagation level of the abductive inference engine and the worlds-level `BEST`s to the subsequent worlds assessment.

Secondly, note the changing of the definition of `bigger/2`. `Rdr11` defines `bigger` to be a straight numerical comparison while `rdr67` defines `bigger` to be twice as big as the "big" defined in `rdr11`. Feature extractor `Functions` like `bigger/2` (and indeed, `previouslySeenEdges/2, seenBefore/2, timesSeen/2, fromDoctor/1, fromMister/1`) need to be maintained. `RDR` has no mechanism for maintaining functions used in conditions. Hence ripple-down-functions (§4.2).

4.2 Ripple-Down-Functions

This section describes ripple-down-functions (`RDF`) [30], an extension to `RDR`. `RDF` enforces the disciplined evolution of `Functions` that perform feature extractors. Using an `RDR`tree the scope of change of a `Function` is limited to the `RDR` rules written *after* the revision of a construct at a particular time. The definition of the `Function` in existing knowledge does not change.

If we adopt the `RDR` frozen knowledge principle (§3.1) for condition `Functions`, then if follows that the definition of a function should be split each time an `Function` refinement is made. In terms of `RDR`, we should add a `FunctionFrame` stack which stores the order in which new `RDR` rules were added to the `RDR` tree. Each entry is this list contains a `FunctionFrame` which stores the definitions of `Functions`. When an `RDR` rule needs a definition of a procedure, it finds its own reference in the function stack and searches back towards `frame0`. At each `FunctionFrame`, it checks for a definition of the required procedure. If found, the search stops and the found `Function` is executed.

We can optimise this representation. We need only add a new `FunctionFrame` when a `Function` definition changes. Whenever a new `RDR` rule is created, a pointer could be added from this `RDR` rule to the `FunctionFrame`. The lookup of all the `Function` definitions in the new `RDR` rule could be done at `RDR` rule creation time thus avoiding having to do this lookup at runtime.

Continuing the `RDR` example of Figure 3, suppose the `Function b` is found to be faulty in the case of `a&b&e`. A new `RDR` rule `rdr4` is added to our `RDR` tree. The resultant tree (and the pointers into the function stack) is shown in Figure 11.

In terms of BNF, `RDF` is only a small extension to `RDR`. An `RDFnode` is an `RDRnode`, plus a pointer into a `FunctionFrame`. Each `FunctionFrame` stores a pointer to a parent `FunctionFrame` (see Figure 12). A `FunctionFrame` stores `Oracles` (recall Figure 5). Within a `FunctionFrame`, no `Oracle name` is repeated. However, `Oracle names` can repeat in different `FunctionFrames`. For example, in Figure 11, `b/0` was defined in two `FunctionFrames`.

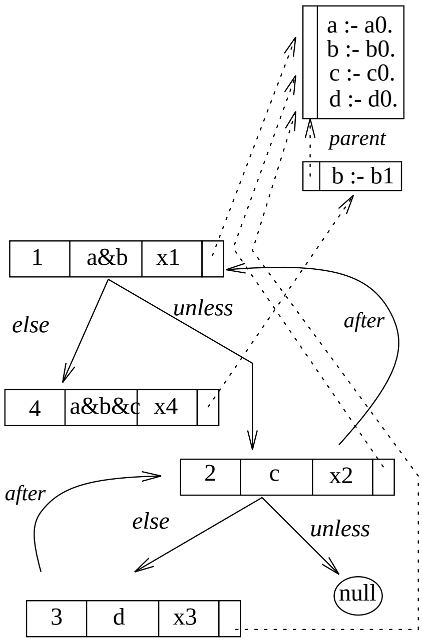


Figure 11: `RDF`= `RDR` plus a `FunctionFrame` stack (top right).

```
RDFnode       ::= RDRnode thatUses FunctionFrame
FunctionFrame ::= {parent FunctionFrame} {Feature}+
```

Figure 12: BNF for `RDF`.

5 Discussion

Our approach has several advantages over conventional knowledge engineering methodologies. Firstly, this framework is stronger on maintenance than other approaches. We have defined a maintenance regime which integrates into our single abductive `KL` device. Maintaining a range of different inference devices (e.g. like those used in KADS) is inherently more complicated.

Secondly, in our framework, we can perform comparative evaluations of different approaches. Consider two interesting comparisons:

- `RD-RA` vs `RDR`: Our framework lets knowledge engineers work at the level of specific data items or at a more general PSM level (contrast `rdr99` in Figure 10 with `rdr11` in Figure 9).

When offered that choice, if knowledge engineers only ever work at the specific example level (e.g. using BESTs like rdr99), then that would make us suspect that RDR is more useful than RD-RA.

- RD-RA vs KL-A: Our framework lets knowledge engineers work at the single-world level or the multiple world level (contrast rdr11 with rdr34 in Figure 9). When offered that choice, if knowledge engineers only ever work at the single-world level (e.g. using BESTs like rdr34), then that would make us suspect that KL-A is more useful than RD-RA.

Thirdly, RD-RA addresses a known problem with RDR. The RDR representation is optimised for maintenance, not human reflection. Consequently, they are hard to read. Various techniques have been explored for reverse engineering a browsable model from an RDR tree. Lee extracts casual models from RDRs [26]; Richards extracts exception graphs [55]; and Edwards explores reflection tools for RDR trees [14]. None of these approaches allow the knowledge engineer to initialise the RDR trees with existing knowledge. In our approach, knowledge engineers can write text files like Figure 6 and Figure 10 and then check them into the maintenance environment. Further, they can then check the knowledge out of the maintenance environment in which case new text files in the format of Figure 6 and Figure 10 are written. That is, while standard RDR is a replacement of existing knowledge engineering approaches, our approach can usefully augment existing knowledge engineering approaches.

Fourthly, the abductive approach described here can be extended from maintenance to conflict resolution in requirements analysis. Requirements modelling (RM) researchers such as Easterbrook [13], Finkelstein *et. al.* [18], and Nuseibeh [42] argue that we should routinely expect specifications to reflect different and inconsistent viewpoints. The software specification problem then becomes one of managing these different viewpoints. In current methodologies, handling and resolving different viewpoints is a time-consuming and costly process. If these different viewpoints are poorly managed, the specifications have to be repeatedly reworked or the runtime system has to be extensively modified.

Our approach could be used for conflict resolution. Let us reconsider Figure 1 and the author labeling on the edges of that graph. Recall the apparent conflict in the middle of Figure 1 on the left-hand-side: author 1 believes a $\overset{--}{\rightarrow}$ c while author 2 believes a $\overset{++}{\rightarrow}$ c. In terms of conflict resolution, the worlds in Figure 2 tell us several things. Firstly, author 1's edges can be found in two worlds; i.e. with respect to the task IN= {aUp, bUp} and OUT= {dUp, eUp, fDown}, a single author's opinions are inconsistent. Secondly, both author 1 and author 2's edges exist in the same consistent world (W[1]); i.e. the apparent conflict of authors 1 & 2 did not matter for this task. Thirdly, author 1 should review their opinion that a $\overset{--}{\rightarrow}$ c since that proved to explain less of the required behaviour that author 2's option that a $\overset{++}{\rightarrow}$ c.

This abductive approach has technical advantages over other conflict resolution approaches. One striking feature of other systems that support multiple-worlds (e.g. Cake [54],Telos [45]) is their implementation complexity[7] We have found that it easier to build efficient implementations [33, 34] using the above graph-theoretic approach than using purely logical approaches (e.g. [23]). Easterbrook's Synoptic tool only permits comparisons of two viewpoints [13, p113]. Abduction can compare N viewpoints. Also, Easterbrook [13] lets users enter their requirements into an explicitly labeled viewpoints which he assumes are internally consistent. We have no need for this restrictive (and possibly overly-optimistic) assumption. Abduction can handle inconsistencies within the opinions of a single user.

Further, our multiple-worlds reasoner is not the JTMS-style [12] approach used in other conflict recognition and management systems (e.g. [54]). A JTMS searches for a single set of beliefs. Hence, by definition, a JTMS can only represent a single viewpoint. Our approach is more like the ATMS [10] than a JTMS. An ATMS maintains all consistent belief sets. We believe that an ATMS approach is better suited to RM conflict management since the different belief sets are available for reflection.

[7] Rich & Waters especially comment on the complexity of their hetrogenous architecture [54].

6 Conclusion

We have characterised knowledge level modeling as an abductive process and PSMs as suites of BEST operators to operationalise the principle of rationality. We have further characterised PSM maintenance as the controlled evolution of the BEST operators via ripple-down-functions. That is:

$$\text{PSM maintenance} = \text{abduction} + \texttt{RDR} + \texttt{RDF}.$$

This abductive approach to PSMs is closer to the KL-A view of knowledge level modeling than the KL-B approach of (e.g.) KADS and, apart from its maintenance advantages, permits comparative analysis of expert systems approaches and conflict resolution in requirements modeling.

References

[1] V.R. Benjamins. Problem-Solving Methods for Diagnosis and their Role in Knowledge Acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93–120, 1995.

[2] J. Breuker. Components of Problem Solving and Types of Problems. In *8th European Knowledge Acquisition Workshop, EKAW '94*, pages 118–136, 1994.

[3] B.G. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[4] T. Bylander, D. Allemang, M.C. M.C. Tanner, and J.R. Josephson. The Computational Complexity of Abduction. *Artificial Intelligence*, 49:25–60, 1991.

[5] W. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[6] W.J. Clancey. Model Construction Operators. *Artificial Intelligence*, 53:1–115, 1992.

[7] P.J. Compton and R. Jansen. A Philosophical Basis for Knowledge Acquisition. *Knowledge Acquisition*, 2:241–257, 1990.

[8] L. Console and P. Torasso. A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7:133–141, 3 1991.

[9] A. Van de Brug, J. Bachant, and J. McDermott. The Taming of R1. *IEEE Expert*, pages 33–39, Fall 1986.

[10] J. DeKleer. An Assumption-Based TMS. *Artificial Intelligence*, 28:163–196, 1986.

[11] J. DeKleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32:97–130, 1 1987.

[12] J. Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12:231–272, 1979.

[13] S. Easterbrook. *Elicitation of Requirements from Multiple Perspectives*. PhD thesis, Imperial College of Science Technology and Medicine, University of London, 1991. Available from *http://research.ivv.nasa.gov/~steve/papers/index.html*.

[14] G. Edwards. Reflective Expert Systems in Pathology. Master's thesis, Computer Science & Engineering, University of NSW, 1996.

[15] K. Eshghi. A Tractable Class of Abductive Problems. In *IJCAI '93*, volume 1, pages 3–8, 1993.

[16] B Falkenhainer. Abduction as Similarity-Driven Explanation. In P. O'Rourke, editor, *Working Notes of the 1990 Spring Symposium on Automated Abduction*, pages 135–139, 1990.

[17] B. Feldman, P. Compton, and G. Smythe. Towards Hypothesis Testing: JUSTIN, Prototype System Using Justification in Context. In *Proceedings of the Joint Australian Conference on Artificial Intelligence, AI '89*, pages 319–331, 1989.

[18] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibe. Inconsistency Handling In Multi-Perspective Specification. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[19] B.R. Gaines and P. Compton. Induction of Ripple Down Rules. In *Proceedings, Australian AI '92*, pages 349–354. World Scientific, 1992.

[20] C. Grossner, A. Preece, P. Gokul Chander, T. Radhakrishnan, and C. Suen. Exploring the Structure of Rule Based Systems. In *Proc. 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 704–709. MIT Press, 1993.

[21] W. Hamscher. Explaining Unexpected Financial Results. In P. O'Rourke, editor, *AAAI Spring Symposium on Automated Abduction*, pages 96–100, 1990.

[22] K. Hirata. A Classification of Abduction: Abduction for Logic Programming. In *Proceedings of the Fourteenth International Machine Learning Workshop, ML-14*, page 16, 1994. Also in *Machine Intelligence* 14 (to appear).

[23] A. Hunter and B. Nuseibeh. Analysing Inconsistent Specifications. In *International Symposium on Requirements Engineering*, pages 78–86, 1997.

[24] B.H. Kang, P. Compton, and P. Preston. Multiple Classification Ripple Down Rules: Evaluation and Possibilities. In *Proceedings 9th Banff Workshop on Knowledge Acquisition*, 1995.

[25] D.B. Leake. Focusing Construction and Selection of Abductive Hypotheses. In *IJCAI '93*, pages 24–29, 1993.

[26] M. Lee and P. Compton. From Heuristic to Causality. In *Proceedings of the 3rd World Congress on Expert System (WcES'96)*, 1996.

[27] D. Marques, G. Dallemagne, G. Kliner, J. McDermott, and D. Tung. Easy Programming: Empowering People to Build Their Own Applications. *IEEE Expert*, pages 16–29, June 1992.

[28] T. Menzies and D. Fensel. 42 Kinds of Knowledge Mainteance, 1997. In preperation.

[29] T. J. Menzies. Applications of Abduction: Intelligent Decision Support Systems. In *Proceedings of the Melbourne Workshop on Intelligent Decision Support*. Department of Information Systems, Monash University, Melbourne, 1996. Also, TR95-16, Department of Software Development, Monash University.

[30] T.J. Menzies. Maintaining Procedural Knowledge: Ripple-Down-Functions. In *Proceedings of AI '92, Australia*, 1992.

[31] T.J. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995.

[32] T.J. Menzies. Situated Semantics is a Side-Effect of the Computational Complexity of Abduction. In *Australian Cognitive Science Society, 3rd Conference*, 1995.

[33] T.J. Menzies. On the Practicality of Abductive Validation. In *ECAI '96*, 1996.

[34] T.J. Menzies. Applications of Abduction: Knowledge Level Modeling. *International Journal of Human Computer Studies*, 45:305–355, September, 1996.

[35] T.J. Menzies and P. Compton. *A Precise Semantics for Vague Diagrams*, pages 149–156. World Scientific, 1994.

[36] T.J. Menzies and P. Compton. Applications of Abduction: Hypothesis Testing of Neuroendocrinological Qualitative Compartmental Models. *Artificial Intelligence in Medicine*, 1997. To appear.

[37] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.

[38] A. Newell. Reflections on the Knowledge Level. *Artificial Intelligence*, 59:31–38, Feburary 1993.

[39] A. Newell, G.R. Yost, J.E Laird, P.S. Rosenbloom, and E. Altmann. Formulating the Problem Space Computational Model. In P.S. Rosenbloom, J.E. Laird, and A. Newell, editors, *The Soar Papers*, volume 2, pages 1321–1359. MIT Press, 1991.

[40] H.T. Ng 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*, volume TR 90-32, pages 13–17, 1990.

[41] P. Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992.

[42] B. Nuseibeh. To Be *and* Not to Be: On Managing Inconsistency in Software Development. In *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, pages 164–169. IEEE CS Press., 1997.

[43] C.L. Paris. The Use of Explicit User Models in a Generation System for Tailoring Answers to the User's Level of Expertise. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems*, pages 200–232. Springer-Verlag, 1989.

[44] R.S. Patil, P. Szolovitis, and W.B Schwartz. Causal Understanding of Patient Illness in Medical Diagnosis. In *IJCAI '81*, pages 893–899, 1981.

[45] D. Plexousakis. Semantical and Ontological Considerations in Telos: a Language for Knowledge Representation. *Computational Intelligence*, 9(1), February 1993.

[46] D. Poole. On the Comparison of Theories: Preferring the Most Specific Explanation. In *IJCAI '85*, pages 144–147, 1985.

[47] D. Poole. A Methodology for Using a Default and Abductive Reasoning System. *International Journal of Intelligent Systems*, 5:521–548, 1990.

[48] H.E. Pople. On the mechanization of abductive logic. In *IJCAI '73*, pages 147–152, 1973.

[49] P. Preston. Expert Systems, Knowledge Aquisition and Knowledge Modification, 1992. Honours thesis, School of Electrical Engineering, University of NSW.

[50] P. Preston, G. Edwards, and P. Compton. A 1600 Rule Expert System Without Knowledge Engineers. In J. Leibowitz, editor, *Second World Congress on Expert Systems*, 1993.

[51] J. Reggia, D.S. Nau, and P.Y Wang. Diagnostic Expert Systems Based on a Set Covering Model. *Int. J. of Man-Machine Studies*, 19(5):437–460, 1983.

[52] J.A. Reggia. Abductive Inference. In *Proceedigns of the Expert Systems in Government Symposium*, pages 484–489, 1985.

[53] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–96, 1 1987.

[54] C. Rich and Y.A. Feldman. Seven Layers of Knowledge Represeentation and Reasoning in Support of Software Development. *IEEE Transactions on Software Engineering*, 18(6):451–469, June 1992.

[55] D. Richards and P. Compton. Combining Formal Concept Analysis and Ripple Down Rules to Support the Reuse of Knowledge. In *SEKE '97: Proceedings of 1997 Conf. on Software Eng. & Knowledge Eng, Madrid*, 1997.

[56] P.S. Rosenbloom, J.E. Laird, and A. Newell. *The SOAR Papers*. The MIT Press, 1993.

[57] B. Selman and H.J. Levesque. Abductive and Default Reasoning: a Computational Core. In *AAAI '90*, pages 343–348, 1990.

[58] L. Steels. Components of Expertise. *AI Magazine*, 11:29–49, 2 1990.

[59] D.M. Steier. CYPRESS-SOAR: A Case Study in Search and Learning in Algorithm Design. In P.S. Rosenbloom, J.E. Laird, and A. Newell, editors, *The SOAR Papers*, volume 1, pages 533–536. MIT Press, 1993.

[60] B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. KADS: a Modeling Approach to Knowledge Engineering. *Knowledge Acquisition*, 4:1–162, 1 1992.

[61] G.R. Yost and A. Newell. A Problem Space Approach to Expert System Specification. In *IJCAI '89*, pages 621–627, 1989.

Some of the Menzies papers can be found at *http:// www.cse.unsw.edu.au/ ~timm/pub/ docs/papersonly.html.*