ELSEVIER
Artificial Intelligence 87 (1996) 215-254
Artificial Intelligence

Improving accuracy by combining rule-based and case-based reasoning

Andrew R. Golding[a,*], Paul S. Rosenbloom[b]
[a] *Mitsubishi Electric Research Laboratories, 201 Broadway, 8th Floor, Cambridge, MA 02139, USA*
[b] *Information Sciences Institute and Computer Science Department, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292, USA*

Received October 1994; revised November 1995

Abstract

An architecture is presented for combining rule-based and case-based reasoning. The architecture is intended for domains that are understood reasonably well, but still imperfectly. It uses a set of rules, which are taken to be only approximately correct, to obtain a preliminary answer for a given problem; it then draws analogies from cases to handle exceptions to the rules. Having rules together with cases not only increases the architecture's domain coverage, it also allows innovative ways of doing case-based reasoning: the same rules that are used for rule-based reasoning are also used by the case-based component to do case indexing and case adaptation. The architecture was applied to the task of name pronunciation, and, with minimal knowledge engineering, was found to perform almost at the level of the best commercial systems. Moreover, its accuracy was found to exceed what it could have achieved with rules or cases alone, thus demonstrating the accuracy improvement afforded by combining rule-based and case-based reasoning.

1. Introduction

Domains vary in the degree to which they are understood, ranging from those that have been codified completely and correctly in terms of a set of rules of behavior, to those for which no such rules are known. This paper is concerned with domains that fall between these two extremes, but closer to the "well-understood" end—domains for which a set of rules is known, but the rules do not cover the full complexities of the domain. The rules must also be able to be run efficiently. In such domains, rules and cases both provide valuable knowledge. While the rules embody the understanding that

* Corresponding author. E-mail: golding@merl.com.

has been codified over the years by experts, cases contain knowledge of the domain in a more unprocessed form—illustrations of actual behaviors that occur in the domain, complete with idiosyncrasies and irregularities. Neither source subsumes the other—the codified knowledge in the rules is not necessarily well represented by any given set of cases, while the idiosyncratic knowledge in the cases is not necessarily captured by known rules. This observation is the basis for the architecture presented here for combining rule-based reasoning (RBR) [17] and case-based reasoning (CBR) [20,30]. The architecture uses a set of rules, which are taken to be only approximately correct, to obtain a preliminary answer for a given problem; it then draws analogies from cases to handle exceptions to the rules.

Having rules together with the cases not only allows the architecture to take advantage of more domain knowledge, it also allows innovations in CBR technology. The architecture incorporates two novel methods for CBR that are based on exploiting the rules of the RBR component. First, the rules are used to index the cases. The indexing scheme, termed *prediction-based indexing*, hangs cases directly off the rules, using the rule antecedents to supply appropriate cues for case retrieval. This avoids having to analyze the domain to identify a suitable vocabulary of direct and derived indexing features; instead, it takes advantage of the domain structure implicit in the rules, and hence already available. The second role of the rules in CBR is for case adaptation, via a strategy of "case adaptation by factoring". The rules are used to factor each source case into the individual steps that were applied within the case, through a process of *rational reconstruction*. The individual steps are then sufficiently fine grained that the relevant ones can be transferred verbatim from source to target, despite overall disparities between the cases. Factoring is thus a way of adapting the source case to enable transfer to globally dissimilar target cases.

To test the architecture for a real-world task, it was applied to the problem of name pronunciation. With minimal knowledge engineering, the resulting system, Anapron,[1] was found to perform almost at the level of the best commercial name-pronunciation systems, and substantially better than other machine-learning systems applied to this task (two versions of NETtalk). Moreover, Anapron's accuracy was found to exceed what it could have achieved with rules or cases alone, thus demonstrating the accuracy improvement afforded by combining rule-based and case-based reasoning.

The next section presents the architecture, independent of the domain of name pronunciation. The Anapron system, which instantiates the architecture for name pronunciation, is then described. A set of experiments on Anapron are presented, the key result being an empirical demonstration of the improvement obtained by combining rules and cases. The last two sections discuss related work, and conclude.

2. The architecture

The architecture is organized as a set of modules that can be configured according to the needs of the domain; see Fig. 1. The minimal configuration consists of four modules,

[1] Anapron stands for *Ana*logical *pron*unciation system.

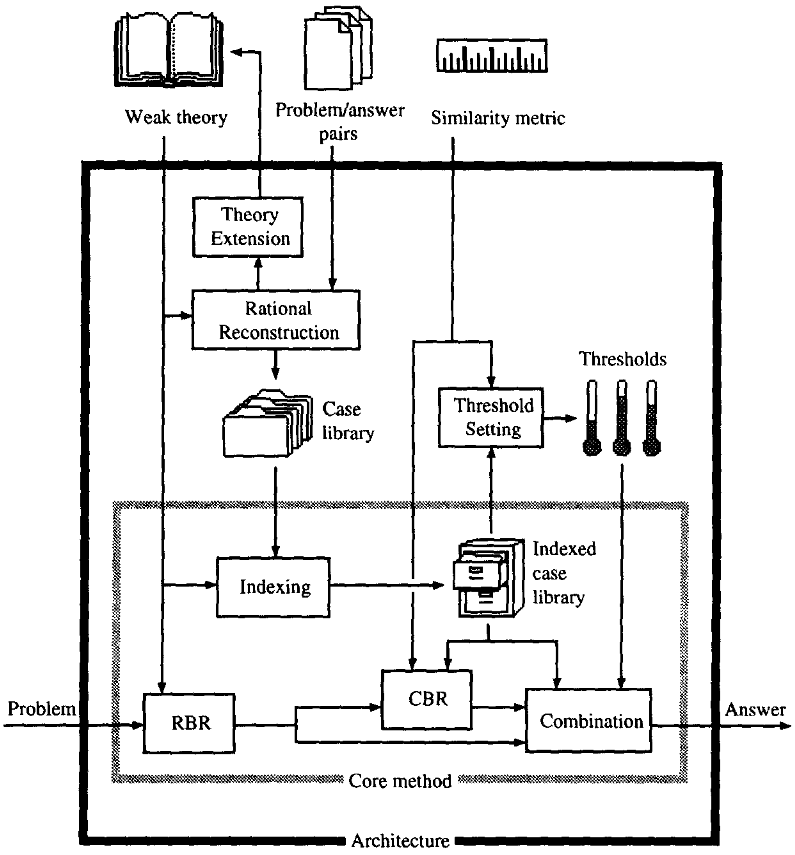


Fig. 1. Anatomy of the architecture. The black outer rectangle represents the overall architecture for combining RBR and CBR; the gray inner rectangle encloses the core method. Boxes represent modules of the architecture, and icons stand for knowledge structures. Links indicate dependencies between components.

collectively termed the *core method*, shown in the diagram enclosed in the gray inner rectangle. These four modules are the heart of the architecture—they implement the method for combining RBR and CBR. The remaining three modules, termed the *support modules*, perform various roles in acquiring the knowledge needed by the core method. Each of these modules may be included, on a domain-by-domain basis, as needed to make up the difference between the knowledge needed by the core method and the knowledge that is already available in the domain.

The sections below describe the core method of the architecture and the support modules, followed by a discussion of design issues. For ease of exposition, examples will be drawn from a toy (but implemented) domain; the instantiation to name pronunciation is deferred to Section 3.

```
Procedure RC-Hybrid(problem)
  Until problem is solved do:
    (a) RBR: Use the rules to select an operator to apply.
    (b) CBR: Look for analogies that contradict the operator suggested by RBR.
    (c) Combination: Decide between the operators suggested by RBR and CBR.
```

Fig. 2. Top-level procedure for combining rule-based and case-based reasoning.

2.1. The core method

The core method is the heart of the architecture; it is the part that solves problems by a combination of RBR and CBR. The central idea is to apply the rules to the target problem to get an approximate answer, and to draw analogies from cases to cover exceptions to the rules. This idea is expressed in the RC-Hybrid procedure of Fig. 2. The procedure treats problem solving as a process of applying operators to the target problem until it is solved. The procedure applies one operator on each iteration. It chooses the operator in three steps. First it selects an operator to apply via the rules. It then looks for analogies that contradict the rules and suggest alternative operators. In the combination step, it decides which operator to actually apply—the one suggested by RBR or one suggested by CBR. Underlying this strategy of starting with the rules and fine tuning with the cases is the assumption that a reasonably fast and accurate set of rules is available. If not, a different architecture may be called for, such as one that applies CBR and RBR in the opposite order.

The core method gets its domain knowledge from four sources: a weak theory of the domain, a case library, a similarity metric, and a set of thresholds. The weak theory is a method for solving problems in the domain using a set of rules. It has two components: the rules themselves, and a set of operators. The operators define the legal actions in the domain. Each operator may have an associated applicability condition that limits the set of states in which it can be applied. The rules provide search control, specifying exactly one operator to apply in every problem-solving state. A weak theory is *weak* in the sense that it does not always suggest the right operator to apply—if it did, there would be no reason to apply the architecture in the first place. In fact, even if the weak theory does not *contain* the right operator to apply, the architecture may be able to recover by applying theory extension, which detects such missing operators by noticing failures of the weak theory to account for examples in the case library, and which invents new operators (and rules) to correct the failures (see Section 2.2.2).

As an example of a weak theory, consider a toy version of a problem in auto insurance: to assess the risk of insuring a new client. Solving a problem in this domain consists of three steps: determining whether the client is an attentive driver, determining whether he[2] is in a hostile driving environment, and, based on the previous inferences, assessing his level of risk. Each step is implemented by applying one of a corresponding set of operators. For example, the first step—determining whether the client, c, is an attentive driver—is implemented by applying either the `attentive` or `inattentive` operator.

[2] Masculine pronouns are intended in the generic sense unless the context indicates otherwise.

Operators

```
                                      high-risk
 attentive       endangered
                <             <     medium-risk
inattentive        neutral
                                      low-risk
```

Rules

```
If occupation(c) = student then attentive                        ; 'Student' rule
elseif sex(c) = M and age(c) < 30 then inattentive               ; 'Young driver' rule
elseif age(c) ⩾ 65 then inattentive                              ; 'Old driver' rule
else attentive                                                   ; 'Default' rule

If address2(c) = New York, NY
   or address2(c) = Los Angeles, CA then endangered              ; 'Hostile traffic' rule
else neutral                                                     ; 'Normal traffic' rule

If inattentive(c) and endangered(c) then high-risk               ; 'High' rule
elseif inattentive(c) or endangered(c) then medium-risk          ; 'Medium' rule
else low-risk                                                    ; 'Low' rule
```

Fig. 3. Weak theory for the toy auto-insurance example. The less-than signs (<) between operators represent applicability conditions that control the order in which the operators are applied. The letter c in the rules stands for a client.

The former adds the assertion attentive(c) to the state, while the latter adds the opposite assertion. Applicability conditions control the order in which operators are applied; for example, `attentive` and `inattentive` are constrained to apply only in the initial state. The weak theory is summarized in Fig. 3.

The next knowledge source needed by the core method is the case library. It is a collection of cases, where a case consists of a problem, its answer, and the chain of operators by which the answer was derived. In the insurance domain, a case translates into a client, the client's level of risk, and the operators that derived that level of risk for that client. The client is represented as a feature vector. Fig. 4 gives an example of

	Target	Case # 1	Case # 6
Name	Smith	Johnson	Davis
Address1	Sigma Chi House	Sigma Chi House	Toyon Hall
Address2	Stanford, CA	Stanford, CA	Stanford, CA
Sex	M	M	F
Age	21	19	22
Occupation	student	student	student
Car-make	Chevrolet	BMW	Toyota
Car-value	2,500	30,000	3,000
Answer		medium-risk(Johnson)	low-risk(Davis)
Operators		`inattentive`, `neutral`, `medium-risk`	`attentive`, `neutral`, `low-risk`

Fig. 4. A target problem and selected cases from the insurance example.

```
Procedure Index(case)
    Until case is solved do:
        (a) Use the rules to predict which operator $o_p$ should apply to case. Let $r$ be the rule that made
            the prediction.
        (b) Compare $o_p$ with the operator $o_o$ that is observed to have been applied to case.
        (c) If the two operators are the same, store the case as a positive exemplar of rule $r$, else as a
            negative exemplar.
        (d) Proceed to apply operator $o_o$ to case.
```

Fig. 5. Procedure for indexing a case.

a problem and two cases in this domain. The full case library in this toy example has 30 cases.

The last two knowledge structures required by the core method are the similarity metric and thresholds. The similarity metric estimates how similar two problems are with respect to the application of a particular operator. The thresholds are used by the combination module in deciding whether an analogy should override the rules. These are both discussed further below.

The individual modules in the core method are presented in the following sections.

2.1.1. Indexing

The purpose of the indexing module is to organize the cases to make them accessible later for CBR. CBR will use the cases to critique RBR; cases are therefore viewed as evidence for or against the rules. A case constitutes evidence *for* a rule if it illustrates a place where the rule makes a correct prediction. It constitutes evidence *against* a rule if it illustrates a place where the rule makes an incorrect prediction. The indexing module stores the case as a *positive* or *negative exemplar* of the rule accordingly.

The complete indexing procedure for a case is shown in Fig. 5. It applies RBR to the case as if it were a new problem. Step (c) does the actual indexing: it stores the case as a positive or negative exemplar of each rule that was predicted to apply to it. Step (d) completes one iteration of the procedure by applying an operator to the case. It applies o_o, the *observed* operator, so that when the rules make their next prediction (in step (a)), it will be based on how the case was actually solved, not on how the rules would have solved it. Applying the predicted operator, o_p, would be incorrect, because then if the rules predict one wrong operator at the beginning, all of their subsequent predictions that are based on this initial wrong operator will be thrown off as well.

Once all of the cases have been indexed as described, they can be used as source exemplars for analogies. The indexed cases will also be used to help in judging analogical compellingness (see Section 2.1.4).

Given that the rules are not expected to be perfect, one may ask how this indexing scheme performs in the face of rule inaccuracies. Consider the situation where the architecture is presented with a target problem, and the case library contains a source problem that is similar to the target and has the same behavior. Will the indexing scheme be able to retrieve this source? The answer is yes, regardless of rule inaccuracies, as long as the rules fire the same way for the source and target problems. This is likely,

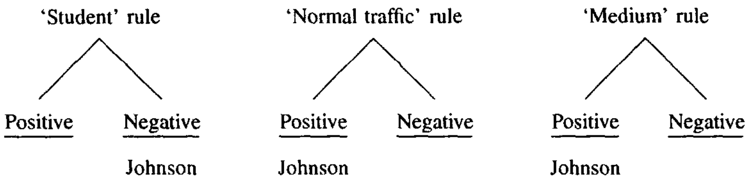


Fig. 6. Results of applying PBI to client Johnson. Johnson is stored as a positive or negative exemplar of each of the three rules that make a prediction for him.

given that the source and target are similar; the only way for the rules to differentiate between them is to test a property that is both irrelevant to their behavior and not shared between them. Thus if the rules handle the source correctly, they will file it under the right rules, and retrieve it when these rules fire again for the target problem. If the rules handle the source incorrectly, they will file it under the wrong rules, and retrieve it when these same wrong rules fire for the target problem. In effect, the rules act as a hashing function, distributing exemplars into behavior classes based on (weak) knowledge of the domain.

The indexing scheme just presented is termed *prediction-based indexing* (PBI) because it indexes cases by the rules that predicted which operators should apply—regardless of whether the predictions were correct. Equivalently, it can be thought of as indexing cases by whatever features the rules looked at in order to make their predictions. This is related to explanation-based indexing (EBI) [3]; however, there the rules are used to explain an observed outcome, rather than to make their own prediction of the outcome. This difference results in quite distinct modes of operation in the two schemes. EBI needs to account for any observed answer, and thus works backward from the answer using correct rule applications. It prefers a theory that is as *broad* as possible—one that can even account for multiple answers to the same problem (rendering the rules nondeterministic). PBI, on the other hand, applies the rules in the forward direction, allowing both correct and incorrect rule applications. It uses the same rules as for ordinary rule-based reasoning, rather than requiring one set for explaining answers and one set for performance. It prefers a theory that is as *accurate* as possible, to minimize the amount of work the architecture will have to do later to override wrong rule applications via CBR.

Example. As an illustration of PBI, consider again the toy auto-insurance example. PBI applies to the first client in the case library, Johnson, in three iterations. The results are shown in Fig. 6. On the first iteration, the rules are applied, and the 'student' rule fires. It predicts the `attentive` operator for Johnson. This differs from the operator specified in the case, `inattentive`. Johnson is therefore stored as a negative exemplar of the 'student' rule. The indexing procedure proceeds to apply the observed operator, `inattentive`, to Johnson. On the second iteration, the 'normal traffic' rule predicts `neutral`, which agrees with the operator given in the case. Johnson is made a positive exemplar of the 'normal traffic' rule. On the last iteration, the 'medium' rule correctly predicts the operator `medium-risk`, based on the assertions of the previous two operators. Johnson is therefore stored as a positive exemplar of the 'medium' rule.

2.1.2. RBR

Rule-based reasoning matches the rules against the target problem. Rules can match any attribute of the problem-solving state, including assertions added by previous problem solving. Rule matching corresponds to step (a) of procedure RC-Hybrid. The matching rule is not actually applied at this point, but rather is taken as a *provisional rule* that will only be applied later if not overridden by CBR.

Example. Continuing with the insurance example, suppose the target problem is to evaluate the risk of insuring client Smith (see Fig. 4). On the first iteration of procedure RC-Hybrid, the 'student' rule matches and is selected as the provisional rule.

2.1.3. CBR

The CBR module acts as a critic of the RBR module. It corresponds to step (b) of procedure RC-Hybrid. It tries to show that the target problem is an exception of the provisional rule by looking for an analogy between the target problem and a negative exemplar of the rule. The negative exemplars of the rule are available in a list hanging off the rule—this was arranged by the indexing scheme. The CBR module goes down this list, proposing analogies one at a time, until it runs out of exemplars, or the combination module judges one of the analogies to be compelling.[3]

The actual proposing of analogies is done by applying the similarity metric. The metric takes three arguments: a source problem, a target problem, and an operator-to-be-transferred from source to target. Here the source problem will be a negative exemplar, and the operator-to-be-transferred will be the operator that was applied to this exemplar. The operator establishes a context for comparing the problems. Given these three arguments, the metric returns two values: a numerical rating of the similarity, called the *similarity score*; and the implicit rule behind the analogy, called the *analogical rule* or *arule*. The implicit rule that any analogy makes is that a particular set of features that were found to be in common between the source and target problems determines the same outcome for the two problems. Accordingly, the left-hand side of the arule gives the features that were judged by the metric to be shared by the two problems, and the right-hand side gives the operator-to-be-transferred. The arule will be used for judging whether the analogy is compelling (see Section 2.1.4).

Example. Back to the insurance example, the RBR module has just proposed the 'student' rule as the provisional rule for Smith. The CBR module attempts to defeat this proposal by likening Smith to previous negative exemplars of the rule. As was shown above, Johnson is one such negative exemplar. The CBR module draws an analogy from Johnson to Smith, with respect to the `inattentive` operator, by applying the similarity metric. A similarity metric can, in general, be as simple or as complex as

[3] If there are multiple compelling analogies for different operators, this procedure will only find the first one. The rationale is that multiple compelling analogies simply indicate multiple acceptable answers; the choice among them is immaterial. In practice, the issue of choosing among multiple compelling analogies was found to be unimportant; the current procedure already draws very few incorrect analogies (see Section 4.1.3), let alone incorrect analogies where an alternative compelling analogy would have been better.

desired, ranging from simply counting identical features, to doing a relevance-weighted feature comparison, to applying a full expert system for measuring similarity. For the insurance domain, a metric at the simple end of the spectrum was chosen: it counts the number of fields that match in the two client structures (and it ignores the operator-to-be-transferred). Text fields are considered to match if they are identical. Numeric fields match if the two numbers fall within the same interval of a predefined set of intervals. For the analogy from Johnson to Smith, the metric yields the arule:

If address1(c) = Sigma Chi House **and** address2(c) = Stanford, CA
  and sex(c) = M **and** age(c) < 30 **and** occupation(c) = student
then `inattentive`.

This arule expresses the features shared by Johnson and Smith according to the metric. The metric returns a similarity score of 5, which is the number of fields that match between Johnson and Smith (and hence also the number of conditions in the arule).

2.1.4. Combination

The combination module implements step (c) of RC-Hybrid: it decides which of the other modules to listen to, RBR or CBR. It does this by evaluating the analogies proposed by CBR. If it deems one of them to be *compelling*, then CBR wins; else RBR wins. Decisions of compellingness are based in part on the similarity score of the analogy. The similarity score is the degree to which the source and target problems match on relevant attributes, and thus the degree to which the problems are expected to have the same answer, according to the similarity metric. Because the metric is only a heuristic, however, the combination module does not rely on it exclusively; it also subjects the analogy to an empirical verification. This is a test of how well the arule—the generalization behind the analogy—works for other exemplars in the case library. The test returns two results: the arule's accuracy, that is, the proportion of exemplars it got right; and the significance of the accuracy rating, which is 1 minus the probability of getting that high an accuracy merely by chance. The calculation of these results is explained in more detail below.

Compellingness can now be expressed essentially as a conjunction of the two factors discussed above: the analogy must have a high similarity score, and it must perform well in the empirical verification. The conjunction enables more robust judgements of compellingness. An analogy between two apparently similar problems will be rejected if the similarity turns out not to be predictive for other examples; and an analogy that works by spurious coincidence on the available examples will be rejected if there is not also a plausible similarity between its source and target. The compellingness of an analogy $\mathcal{A}$ is defined more precisely as follows:

Compelling-p($\mathcal{A}$) $\iff$
  similarity-score($\mathcal{A}$) $\geqslant SS_0$
  and accuracy($\mathcal{A}$) $\geqslant A_0$
  and (significance($\mathcal{A}$) $\geqslant S_0$ **or** similarity-score($\mathcal{A}$) $\geqslant SS_+$)

where SS_0, SS_+, A_0, and S_0 are thresholds for deciding when a value is high enough; they can be provided from the outside, or set by the threshold setting module (Section 2.2.3). This definition requires the analogy to be strong on all parameters—the score assigned by the similarity metric, and the accuracy and significance from the empirical verification. However, an escape clause—the disjunct involving SS_+—provides a way of accepting analogies between overwhelmingly similar problems, even if there are not enough data for a significant accuracy reading.

The calculation of the accuracy and significance of an analogy will now be explained in more detail. The first observation is that the arule may be viewed as a specialization—in particular, an exception class—of the provisional rule. It follows that the arule applies only to a subset of the exemplars (both negative and positive) of the provisional rule. It does not apply to the rest of the exemplars in the case library. When the arule does apply to an exemplar, it will suggest the application of the operator o on its right-hand side. This operator may or may not agree with the operator that is observed to have been applied to the exemplar. Several definitions can now be made. Let:

- m = number of exemplars that the arule applies to and that were observed to have had operator o applied,
- n = total number of exemplars that the arule applies to,
- M = number of exemplars of the provisional rule that were observed to have had operator o applied,
- N = total number of exemplars of the provisional rule.

The accuracy of the arule is then given by m/n. As mentioned above, the significance is one minus the probability, p, of getting that high an accuracy merely by chance. In calculating p, a slight correction is needed. The probability of getting m out of n exemplars right is influenced by the fact that the arule was *constructed* to be right for one of the exemplars—namely, the source case for the analogy. The calculation of p therefore pretends that this source case does not exist; it uses $m' = m - 1$ and $n' = n - 1$ in place of m and n. With this in mind, p can be calculated using Fisher's exact test [13, p. 25]:

$$\begin{aligned} p &= \text{prob(getting at least } m'/n' \text{ by chance)} \\ &= \sum_{m' \leqslant k \leqslant n'} \text{prob(getting exactly } k/n' \text{ by chance)} \\ &= \sum_{m' \leqslant k \leqslant n'} \frac{\binom{M}{k}\binom{N-M}{n'-k}}{\binom{N}{n'}}. \end{aligned} \qquad (1)$$

Unfortunately, Eq. (1) is computationally unpalatable for large values of N. An approximation is therefore used. It assumes that N is large compared to n', which is reasonable for nontrivial-sized case libraries. Under this assumption, the probabilities can be calculated as if the exemplars are being drawn from an infinite population. Thus the probability of getting one exemplar right by chance is just equal to the proportion of "right" exemplars in the overall population—namely, $r = M/N$. The probability of getting k exemplars right by chance will be r^k—each exemplar has the same probability, r, and the probabilities multiply. The revised derivation of p is then:

$$\begin{aligned} p &= \text{prob(getting at least } m'/n' \text{ by chance)} \\ &= \sum_{m' \leqslant k \leqslant n'} \text{prob(getting exactly } k/n' \text{ by chance)} \\ &\approx \sum_{m' \leqslant k \leqslant n'} \binom{n'}{k} r^k (1-r)^{n'-k}. \end{aligned} \tag{2}$$

Example. Consider again the analogy from Johnson to Smith. To decide whether this analogy is compelling, the combination module first runs an empirical verification. It tests the arule on the positive and negative exemplars of the provisional 'student' rule. It happens that the arule applies to four of these exemplars, Johnson and three others. Three of the four are listed as `inattentive`. Thus $m = 3$ and $n = 4$, giving an accuracy of 0.75. Turning to the 'student' rule as a whole, 4 out of 10 exemplars are listed as `inattentive`. So $M = 4$, $N = 10$, and $r = 0.4$. The significance of the accuracy rating then works out to be 0.648 by Eq. (2).[4] Also, as mentioned earlier, the similarity score of the analogy is 5. The thresholds in this domain were set to the values $SS_0 = 4$, $SS_+ = 6$, $A_0 = 0.66$, and $S_0 = 0.50$ (see Section 2.2.3). Thus the analogy is deemed compelling. The upshot is that Smith is determined to be inattentive, by analogy to a similar inattentive student from the same fraternity. Assuming that no compelling analogies are found in the balance of the problem solving, Smith will ultimately be assessed as medium risk, rather than low risk as the rules alone would have predicted.

2.2. The support modules

The knowledge structures required by the core method may not be readily available in all domains. The role of the support modules is to help construct these knowledge structures. Each of the three support modules deals with a particular issue in the construction. The first, rational reconstruction, deals with the issue that while a set of problem/answer pairs may be available for the domain, the path by which each answer was derived may not be—but these paths are needed by the core method as part of the case library. Rational reconstruction uses the weak theory to infer the path of operators leading from each problem to its answer.

The second support module, theory extension, deals with the issue that the weak theory may have gaps that prevent the abovementioned reconstructions from going through. When such a gap is exposed, theory extension proposes plausible new operators or rules to add to the weak theory to bridge the gap.

The third support module, threshold setting, deals with the issue that there may not be predefined values to use for the four thresholds in the definition of analogical

[4] For comparison, Fisher's exact test in Eq. (1) would have given 0.667; the discrepancy is noticeable because N is so small in this toy example.

compellingness. Threshold setting chooses values via a learning procedure that generates training examples for itself from the case library.

The support modules effectively reduce the knowledge requirements of the architecture to three: a weak theory, a set of problem/answer pairs, and a similarity metric. Rational reconstruction and theory extension have hooks to incorporate supplemental domain knowledge if desired, as discussed below. The following sections briefly describe the three support modules. A more complete description can be found in Golding [14].

2.2.1. *Rational reconstruction*

Examples of problems and their answers are available in many domains—spellings and their phonetic transcriptions in pronunciation, patients and their diagnoses in medicine, theorems and their proofs in mathematics, etc. What tends to be not so widely available is the chain of reasoning by which each answer was derived—experts have trouble articulating how they pronounced a name, or arrived at a particular diagnosis, or came up with a proof. Unfortunately, without this information, an answer is of rather limited use; it can only be applied to new problems in toto. Any system that wants to transfer just part of the answer to a new problem needs some way of breaking down the answer into individual steps. The rational reconstruction module (RR) provides a way of doing this. Given a problem and an answer—and using a weak theory of the domain—RR infers an operator sequence that leads from the problem to the answer.

Rational reconstruction can be viewed as a problem of search for an operator sequence. The operators in the sequence are drawn from a weak theory of the domain. The sequence must satisfy two constraints. First, it must account for the given problem and answer:

> *Validity*: The operator sequence, when applied to the problem, must produce the answer.

It may happen that no operator sequence satisfies the validity constraint; this signals that the weak theory is missing one or more operators. In this case, RR calls theory extension to fill the gap. The opposite problem is when there are multiple valid operator sequences. Here, RR invokes the rules of the weak theory as a bias: it prefers the operator sequence that is closest to what the rules would have predicted. The idea is that even though the rules are not perfect, they are good enough to steer RR toward plausible derivations. This is expressed in a second constraint:

> *Minimality*: The operator sequence must deviate minimally from the sequence predicted by the rules of the weak theory.

This brings up a second opportunity for patching the weak theory: if RR cannot find a valid operator sequence with *zero* deviation, this means that the rules do not predict a valid operator sequence for the problem at issue. In such cases, RR may call theory extension to alter the rules such that they *do* predict a valid operator sequence. This option is rarely invoked, however; the primary approach of the architecture is not to fix imperfect rules, but to supplement them with CBR.

The two constraints above lead to two strategies for RR: the validity-first strategy, which generates valid operator sequences and selects the one with minimal deviation; and the minimality-first strategy, which generates operator sequences in order of increasing deviation, and selects the first valid one. The deviation of an operator sequence is the sum of the deviations of each of its operators, where a *deviation metric* measures the deviation of an operator. The default metric scores 0 if the operator agrees with the one predicted by the rules, and 1 otherwise. The scoring can be made more sophisticated by including domain-specific knowledge that penalizes deviations according to their severity. The validity-first and minimality-first strategies, and combinations thereof, define a space of possible strategies for doing RR. The particular strategy that is best for a given domain depends on a number of factors, including the accuracy of the rules in the domain. Pruning and ordering heuristics may be used in conjunction with either strategy to speed it up.

A couple of observations about RR can be made at this point. The first is that RR can be regarded as doing a form of credit assignment. In particular, suppose that RR is given a problem/answer pair that has several valid reconstructions, each of which violates a different rule. Then RR, in choosing among these reconstructions, is implicitly doing credit assignment, as it is deciding which rule violations hold. Moreover, it is doing the credit assignment by invoking a minimality bias—it selects the reconstruction with the smallest total deviation from the rules. Put another way, it assigns credit so as to minimize the total amount of blame.

The second observation concerns RR's effectiveness as a function of the *directness* of the operators in the weak theory. A *direct* operator affects the final answer of a problem by directly altering some part of it. An *indirect* operator does not manipulate the answer itself, but rather affects the choice of other operators. Indirect operators tend to be harder to reconstruct, because they are relatively unconstrained by the answer. RR must therefore rely more on its minimality bias in reconstructing them. This can be dangerous, as the minimality bias is only as accurate as the rules of the weak theory. By and large, therefore, the more direct the operators in the weak theory are, the more effective RR will be.

There is a variety of work related to RR, including learning apprentices, plan recognizers, student-modelling systems, and story-understanding systems. These systems all infer some kind of trace of the reasoning behind an agent's observed behavior. Such systems typically enforce the validity constraint; that is, they produce traces that are consistent with the observed behavior. The differences among systems lie in the bias they use for choosing among the valid traces. One simple bias is to return the first valid trace found; this is the approach taken in PAM [40], a story-understanding system. A widely used bias is to prefer the *simplest* valid trace—i.e., the one that makes the fewest assumptions—as in, for example, the plan-recognition work of Kautz and Allen [18]. The ACCEL system [26], which has also been used for plan recognition, instead prefers the most *coherent* trace. CELIA [29], a learning apprentice in the domain of automotive repair, prefers the valid trace that is thought to best capture the hierarchical goal structure underlying the linear sequence of the expert's actions. Other systems adopt the same bias as RR: they prefer the valid trace that is closest to what a theory would have predicted. Such systems can be classified according to where they fall in the space of strategies described above for RR. For instance, the BUGGY student-modelling system [5] uses a minimality-first strategy to infer a model of how a student does arithmetic.

Example. The toy insurance domain, as usual, will furnish an illustration of RR. A pure validity-first strategy will be used, with no pruning or ordering heuristics. Consider the reconstruction of the operator sequence for client Johnson (see Fig. 4). The validity-first strategy first generates all valid operator sequences that account for the answer of medium risk:

(1) `attentive, *endangered, medium-risk,`
(2) `attentive, neutral, *medium-risk,`
(3) `*inattentive, *endangered, *medium-risk,`
(4) `*inattentive, neutral, medium-risk.`

The deviation of each sequence is then measured. Operators that are found to disagree with the ones predicted by the rules have been marked above with an asterisk (*). If the default deviation metric is used, then sequences (1), (2), and (4) tie for first place with a score of 1.0; the choice among them can only be made arbitrarily. However, the deviation metric that was actually used for this toy domain treats violations of the 'student' rule as less serious than other violations. Thus operator sequence (4) wins, as reflected in the reconstruction information shown in Fig. 4.

2.2.2. *Theory extension*

In the process of reconstructing how an observed answer was derived, RR is bound to turn up inadequacies of the weak theory. In such cases, the theory extension module (TE) is invoked to patch the weak theory appropriately. The general problem of theory repair is quite difficult. Wilkins [41, Chapter 4] gives a good overview of various techniques that have been tried; there has been subsequent work under the rubric of abduction [25] for example. Because the general problem is so hard, TE takes a restricted approach to theory repair. It is geared to the two particular situations in which TE is invoked by RR.

The first situation is when RR cannot find a valid operator sequence for a given problem/answer pair. Viewing reconstruction as a search task, this means there was no complete path from the start state (containing the problem) to the goal state (containing the answer). TE tries to complete the path by proposing new operators to bridge gaps between previously unconnected states. In general, there will be multiple sets of operators that will do this. TE selects the minimal set, where minimality is defined by a cost metric. The cost metric typically uses domain-specific knowledge to evaluate the cost of inventing a new operator between a given pair of states.

The second situation in which TE may be invoked is when RR cannot find a valid operator sequence that has zero deviation from the rules. This presents an opportunity to alter the rules to bring their prediction into agreement with one of the valid operator sequences. However, TE will only attempt this under very constrained circumstances. In particular, if it is known that a certain class of rules in the domain fits a particular template, then TE can try proposing a new rule by instantiating the template for the current problem. If this improves the deviation score of the best valid operator sequence, then the new rule may be worth adding to the theory.

In the existing implementation, TE requests user approval before actually making any of its proposed changes. This provides a sanity check on whether the extensions appear to be reasonable.

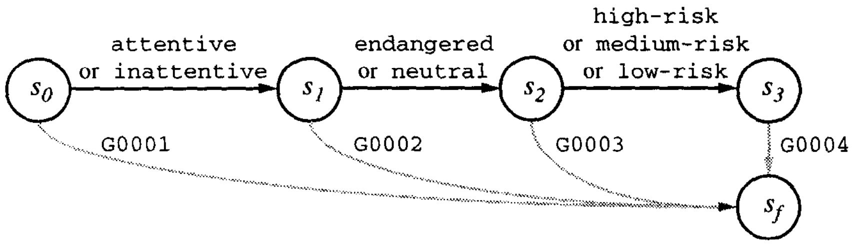


Fig. 7. RR's search space for client Davis, whose answer has been modified to be negligible(Davis). Black arcs indicate applications of existing operators. Gray arcs are for new operators being considered by TE.

Example. RR was able to reconstruct the 30 cases in the case library for the toy insurance domain without recourse to TE. For illustrative purposes, however, RR was run on a couple of additional examples and forced to call TE. In the first of these examples, client Davis (see Fig. 4) was modified to have an answer of negligible(Davis). RR could not find any operator sequence to explain this answer, and so it called TE. RR's search space is shown in Fig. 7. State s_0 is the start state, containing the information initially known about the case: name(Davis) = Davis, address1(Davis) = Toyon Hall, and so on. State s_f is the final state, containing the answer negligible(Davis). Arcs represent operator applications; for instance, an arc leaving state s_0 applies the `attentive` operator, resulting in a state s_1 containing the new assertion attentive(Davis). In the figure, the `attentive` arc and resulting state s_1 have been collapsed with the arc and resulting state for its sibling operator, `inattentive`. This collapsing was done throughout the diagram to hide distinctions irrelevant to theory extension.

RR's inability to reconstruct Davis can be seen from the unreachability of the final state s_f from the initial state s_0. TE's job is to fix this by adding one or more new operators. Four operators are considered, G0001 through G0004, shown by the gray arcs in the figure. To choose among these, TE applies a cost metric for the insurance domain. The metric is based on the number of *inferences* in a state, where an inference is an assertion added by an operator. The initial state has zero inferences. State s_2 has two inferences, because two assertions are needed to reach it; e.g., attentive(Davis) and neutral(Davis). State s_f has three inferences, because other final states in this domain have three inferences, and the number of inferences is taken to be a constant across final states. The metric then defines the cost of an operator to be the number of inferences it is implicitly making by going from one state to another. For instance, the cost of G0002 is 2, because it goes from a state with one inference to a state with three inferences. If an operator implicitly makes zero or a negative number of inferences, it is considered nonsensical, and is assigned an infinite cost. The effect of this metric is to connect the initial and final states using existing operators to account for as many inferences as possible, and new operators for as few as possible. This is a rudimentary example of how a cost metric can guide TE toward minimal extensions of the theory. By the metric, operators G0001 through G0004 have costs 3, 2, 1, and infinity, respectively; hence TE prefers G0003. This enables RR to infer the operator sequence `attentive`, `neutral`, G0003 for Davis.

As an example of TE for rules, consider the rational reconstruction of client Johnson. As mentioned above, Johnson has four valid operator sequences:

(1) `attentive`, `*endangered`, `medium-risk`,
(2) `attentive`, `neutral`, `*medium-risk`,
(3) `*inattentive`, `*endangered`, `*medium-risk`,
(4) `*inattentive`, `neutral`, `medium-risk`.

Asterisks ($*$) in the list mark rule violations. The sequence with the smallest deviation from the rules was sequence (4) (as shown in Section 2.2.1), but it still has a positive deviation. This signals an opportunity for extending the rules. Now suppose that the following rule template is known:

If address2(c) = ____ **then** `endangered`.

TE may then instantiate the template for Johnson, yielding:

If address2(c) = Stanford, CA **then** `endangered`.

When this rule is inserted into the weak theory, it brings sequence (1) above into complete agreement with the rules—i.e., it gives it a deviation of zero. This is an improvement from the previous best deviation, which was for sequence (4), and was nonzero. TE therefore proposes this new rule as a possible extension to the theory.[5]

2.2.3. Threshold setting

The threshold setting module (Tset) provides a principled way of choosing values for the thresholds of the core method. The thresholds are used in determining when an analogy is compelling. The definition of compellingness is repeated here for convenience:

$$\begin{aligned}
&\text{Compelling-p}(\mathcal{A}) \iff \\
&\quad \text{similarity-score}(\mathcal{A}) \geqslant SS_0 \\
&\quad \textbf{and } \text{accuracy}(\mathcal{A}) \geqslant A_0 \\
&\quad \textbf{and } (\text{significance}(\mathcal{A}) \geqslant S_0 \textbf{ or } \text{similarity-score}(\mathcal{A}) \geqslant SS_+).
\end{aligned}$$

The point of compellingness is to enable the architecture to decide when it should listen to an analogy—i.e., when the analogy is right and the rules are wrong. The goal of Tset, consequently, is to pick values for the thresholds that will result in analogies being classified as compelling whenever they would correct wrong answers of the rules. Tset takes a machine-learning approach: it generates training analogies, and tries to pick the thresholds so as to do the right thing for these training analogies—that is, it should accept analogies that correct wrong answers of the rules, and, conversely, it should reject analogies that spoil right answers of the rules. The approach is semi-automatic in that the user has the final say of what values to pick, based on Tset's recommendations.

Tset generates its training analogies from the case library. It pretends that each exemplar in the case library in turn is a target problem, and it finds all analogies

[5] Although the new rule improves the system's account of this particular client, it may worsen its account of other clients. The system depends on the user to verify that the rules proposed by TE are in fact reasonable additions to the theory.

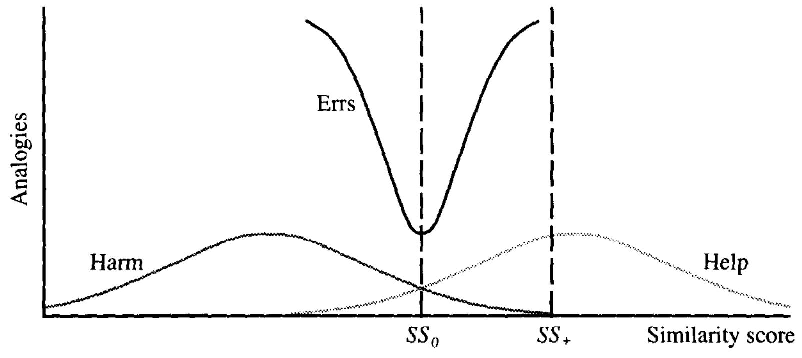


Fig. 8. Distributions of helpful, harmful, and misclassified analogies, as a function of similarity score. SS_0 and SS_+ mark the theoretically optimal settings for the similarity thresholds.

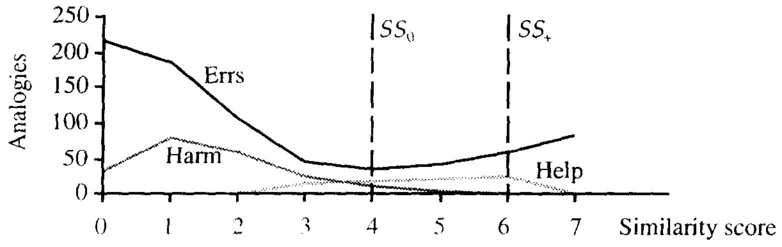
to it from other exemplars. A training analogy is classified as *helpful* if it suggests the right operator for a target problem where the rules would have suggested the wrong operator. An analogy is *harmful* if it suggests the wrong operator for a problem where the rules would have suggested the right operator. Tset's task may now be framed as one of selecting threshold values that minimize the number of misclassified analogies, where misclassified analogies are the helpful ones that are judged *un*compelling plus the harmful ones that are judged compelling.

While Tset could, in principle, search the 4-dimensional space of threshold settings for the one that minimizes the number of misclassified analogies, this turns out to be quite costly in practice. Instead of trying to set all 4 thresholds simultaneously, therefore, Tset sets them one at a time. This gives up the guarantee of finding the global minimum in exchange for tractable run time. The threshold-setting procedure has three steps. On the first step, Tset temporarily adopts a simplified definition of compellingness:

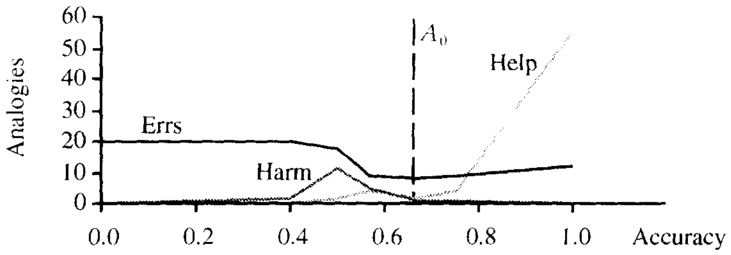
$$\text{Compelling-p}'(\mathcal{A}) \iff \text{similarity-score}(\mathcal{A}) \geqslant SS_0.$$

This definition requires setting only one threshold, SS_0, to minimize the number of misclassified analogies. Fig. 8 shows what prototypical distributions of helpful, harmful, and misclassified analogies would look like at this stage of the processing. The value of SS_0 that minimizes the number of misclassified analogies is also shown. Tset does not choose this value automatically, however, but rather displays the distributions to the user and lets him make the final decision. The SS_+ threshold is also set at this point, the natural choice being a value just high enough to exclude all harmful analogies.

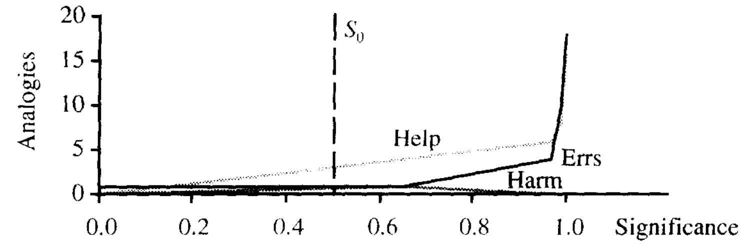
Once a value of SS_0 is selected, all training analogies whose similarity scores fall below this value can be discarded; they have already been classified as uncompelling, and so offer no information about how to set the rest of the thresholds. Each subsequent step of the threshold-setting procedure therefore has fewer training analogies to process—only the ones that are left unclassified by the previous steps. This makes the subsequent steps faster to run, although it also makes their conclusions less reliable due to the smaller number of examples on which they are based.



(a) Distributions of analogies by similarity score.



(b) Distributions of analogies by accuracy.



(c) Distributions of analogies by significance.

Fig. 9. Results of applying the threshold-setting procedure to the toy insurance problem. For each of the three steps of the procedure, the distributions of helpful, harmful, and misclassified training analogies are shown. The misclassified analogies are the helpful ones below a threshold value and the harmful ones at or above the value. Dashed lines indicate the values chosen for the thresholds.

The second and third steps of the threshold-setting procedure set the A_0 and S_0 thresholds, respectively. These steps are similar to the first (except that there is no analog to SS_+). Each step adopts a temporary definition of compellingness, adding one more conjunct of the true definition. At the end of the third step, all four thresholds—SS_0, SS_+, A_0, and S_0—will have been set.

Example. Tset was used to set the thresholds for the insurance problem. The results appear in Fig. 9. The distributions of analogies are shown for each of the three steps of the procedure. These curves do not quite have the ideal shape depicted in Fig. 8, being based on only 30 cases, the number available for the insurance problem. The threshold values that were selected for this task were $SS_0 = 4.0$, $SS_+ = 6.0$, $A_0 = 0.66$, and $S_0 = 0.50$. The values for SS_0, SS_+, and A_0 were selected at or near the optimum values. The choice of S_0 was less clear-cut, as the error curve was largely flat between 0.0 and 0.65 (this was because it was based on a lopsided 34 helpful analogies and 1 harmful analogy). Its value of 0.50 was chosen somewhat arbitrarily within this range.

2.3. Discussion

A number of "frequently asked questions" about the design of the architecture are discussed below. They are grouped by whether they concern the combination of RBR and CBR, just RBR, or just CBR.

2.3.1. Combination issues

Why is RBR applied before CBR? Rule-based and case-based reasoning can be combined in three main orders: RBR first, CBR first, or some interleaving of the two. The architecture presented here adopts the RBR-first strategy, using CBR merely to patch errors left by RBR. This strategy is appropriate when the rules are reasonably efficient and accurate to begin with. If the rules are deficient in some way, the CBR-first strategy may make more sense. If the rules and cases offer more balanced contributions to the problem solving, then an interleaving strategy may be best.

Can analogies be drawn from *positive* exemplars? While the architecture currently draws all of its analogies from negative exemplars, analogies from positive exemplars could also be useful in certain situations. One way to use them would be to decide between RBR and CBR by weighing the evidence from positive analogies against that from negative analogies. The drawback of this approach, however, is that it relies on the case library to illustrate not only the places where the rules are wrong, but also all situations where they apply correctly. Given that the rules are assumed to be fairly accurate to begin with, this could require a huge number of positive exemplars.

An alternative that relies less on having total analogical coverage of the domain is to use positive analogies to resolve nondeterminism in the rules—places where the rules suggest multiple operators. The operator with the greatest support from positive analogies is then selected. This approach is less sensitive to gaps in positive coverage because it compares positive analogies to other positive analogies, not to negative analogies. Gaps in positive coverage therefore tend to affect all operators in the comparison equally (especially if the evidence for each operator is averaged over a set of positive analogies). Such a scheme was implemented in Anapron, and is described below (see Section 3.3). This use of positive analogies may be regarded as a method for combining rules and cases to make nondeterministic answers unique. By contrast, the architecture presented here is a method for combining rules and cases to make deterministic answers more accurate. The two methods provide orthogonal functionality, and may be used separately or in combination. Anapron is an example of using them in combination.

Could rules and cases be converted into a uniform representation? An alternative to a truly hybrid system—one that works from multiple representations—is to convert all knowledge sources into a uniform representation, and work from that. Converting between rules and cases tends to be hazardous, however; the conversion tends to yield inefficient or unreliable representations. See also Golding and Rosenbloom [15].

2.3.2. RBR issues

What if the rules are not of an *if-then-else* form? The architecture assumes that exactly one rule will fire in any state to recommend the next operator to apply. This affords an easy way of assigning credit to the rules: a rule is held responsible for the operators it recommends. This credit assignment enables the architecture to improve the performance of the rules—it lets it associate past mistakes (negative exemplars) with particular rules, and later override similar incorrect behaviors by analogy to the past mistakes. In a more distributed, evidence-gathering model of problem solving, such as that of MYCIN [7], multiple rules can fire, and all contribute to each decision that is made. To accommodate such a rule formalism into the architecture, an analogous credit-assignment procedure would be needed—one that would ascribe some proportion of the credit for each decision to each of the rules that contributed to it.

2.3.3. CBR issues

What is the appropriate level of generality for arules? Each time the architecture draws an analogy, it extracts the generalization behind the analogy, and represents it as an explicit rule, the arule. It then uses the arule to do an empirical verification of the analogy. This involves trying out the arule on other cases in the case library. The purpose is to see how well the generalization holds up for other examples.

An arule is not uniquely determined; the only constraint is that it must be a generalization of the source and target problems. Currently, the architecture makes the arule maximally specific—it includes in the arule *all* conditions shared by the source and target (according to the similarity metric). The idea is to minimize the inductive leap; this reduces the risk of overgeneralization. However, there is an argument for generalizing more liberally: the more general the arule is, the easier it will be to find cases in the case library to which the arule applies, and thus the more informed the empirical verification will be. Striking an appropriate balance between this greater ease of empirical verification and the risk of overgeneralization is an area for future work.

Could the architecture save its arules? Given that the architecture already goes to the trouble of extracting the generalizations behind its analogies (as arules), it certainly could store them. Incorporating the arule into the existing rule set is straightforward: the arule represents an exception class of the rule from which it originated. Thus the arule would be stored so as to always override the original rule. If the rules are of an *if-then-else* form, this means ordering the arule just ahead of the original rule.

Saving arules in this way would gradually "compile" the cases into rules, thereby shifting the burden of problem solving from CBR to RBR. Whether this should be done is basically a store-versus-compute tradeoff. The architecture can store its arules, in which case it saves the time of rederiving the analogies; or it can compute the arules, in which case it saves the storage cost of keeping around all past arules. One could imagine resolving this tradeoff in either direction. The decision in the architecture to (re-)compute, rather than store, was based on the reasoning that, because the arules are constructed to be maximally specific (as discussed above), a policy of storing would end up keeping a large number of rules that hardly ever fired; moreover, these rules could easily be rederived if needed.

How does the architecture do case adaptation? Case adaptation is the process of transforming a source case to make it applicable to a (disparate) target case. Traditional techniques for case adaptation involve retrieving the entire source solution, and doing localized problem solving to patch the parts that are incompatible with the target case. The architecture presented here takes a different tack: using RR, it factors the source case into individual operator applications. It then draws analogies from these individual operator applications to the target problem. The individual operator applications are sufficiently fine grained that they can generally be transferred to the target problem verbatim. Thus the architecture employs a strategy of "case adaptation by factoring". This strategy is related to the idea of decomposing a source case into smaller parts, or *snippets*, each of which addresses one subgoal of the case [21].

Can the architecture learn new cases? Classical CBR [30] involves a learning step: after a target case is solved, it is stored back into the case library to enrich the system's bank of experience. Such a learning step could be incorporated into the architecture presented here by having the architecture ask, after each problem it solves, whether its solution is correct. If the answer is affirmative, the case may be added to the case library directly. If not, further dialogue would be needed to debug the answer before it is stored away. This procedure is not currently implemented because it would create a need for run-time feedback from the user. However, additional cases can always be added to the case library off-line if desired.

How is noise in the case library handled? The architecture protects itself against inaccurate cases through empirical verification. A bad case may lead to a bad analogy being proposed; but the analogy will be rejected unless there is a significant number of supporting cases. The one exception is if the bad analogy has a high enough similarity score to exceed the SS_+ threshold; in that event, it will be accepted even without other supporting cases. This is highly unlikely, however, as the SS_+ threshold should have been set high enough to avoid such spurious analogies.

3. Anapron

The architecture presented here was applied to name pronunciation, resulting in the Anapron system. Names, because of their varied etymology, are a notorious stumbling block for pronunciation systems; this has made name pronunciation an important problem in text-to-speech synthesis. The domain is well suited to application of the architecture, as reasonably accurate and efficient rules of pronunciation are known, yet the domain is sufficiently complex that perfect rules have never been devised—rules inevitably have exceptions. This suggests application of the architecture presented here. The architecture can take advantage of the existing rules, imperfect though they may be, while also tapping into an alternative knowledge source, namely examples of names and their pronunciations. By assimilating knowledge from both sources, the architecture has the potential to outperform existing systems, which are either rule-based or case-based, but not true hybrids.

The sections below start by introducing the domain of name pronunciation. The application of the architecture to this domain is then described; and an additional analogical mechanism is presented that was incorporated to deal with nondeterminism in the rules. The descriptions are at a high level, to give the basic idea of Anapron's operation and a sense of the task of name pronunciation, without getting deep into the intricacies of the domain. For full details, see Golding [14].

3.1. Name pronunciation

Name pronunciation is a problem of practical interest. It comes up in almost any text-to-speech application, but especially name-intensive applications, such as telephone-based credit validation, voice mail, and generic reverse directory assistance (i.e., number to name) [38]. The most important property that makes names unique, as compared to regular words, is their varied etymology. Not only must a pronunciation system infer the language of origin of a name, it must then decide how to treat foreign names. Strict adherence to the native pronunciations can come out sounding stilted at best, and unintelligible at worst. What is needed, assuming an American user population, is some appropriately anglicized interpretation of the foreign languages. These etymology-related difficulties make names problematic for pronunciation systems. The brute-force solution would be to construct a giant pronouncing dictionary of all names the system is apt to encounter. The problem is that the set of names is in general open-ended. A system reading stories off the AP newswire, for instance, has to contend with a constantly changing set of newsworthy individuals. In the end, one can only expend a finite amount of resources building a name dictionary; pronunciation systems will always have to deal with the problem of unfamiliar names.

Until fairly recently, the solution was simply to pronounce names badly; pronunciation of proper names was acknowledged to be an open problem [19]. In recent years, however, a substantial effort has been devoted to the problem, resulting in several high-quality commercial systems. The predominant approach has been to develop rules tailored specifically to names, as in, for example, the Orator$^{\text{TM}}$ [6] system [34] and DECvoice [37]. While these systems have achieved among the best performance yet demonstrated, they have also shown the extreme difficulty of writing rules to cover every contingency. No matter how many rules are written, there always seem to be exceptions. This observation is the basis for an alternative approach to the problem,

[6] Orator is a trademark of Bellcore.

Table 1
Illustration of Anapron's pronunciation modules for KEIDEL; the output of the modules has been abridged for clarity

Module	Function	Application to KEIDEL
Language	Determine language	Generic or German
Morphology	Identify prefix, root, and suffix morphemes	KEIDEL = a single root morpheme
Transcription	Map letters to phones	`kiydehl` if Generic; `kaydehl` if German
Syllable structure	Break into syllables	`kiy-dehl` if Generic; `kay-dehl` if German
Stress assignment	Assign level of stress to each syllable	`k'iydehl` if Generic; `` k`ayd'ehl `` if German
Selection	Pick best language/morphology analysis	`` k`ayd'ehl `` (German)

which views name pronunciation more as a huge bag of idiosyncratic behaviors than as a rule-governed process. The approach, embodied in the TTS system [9], is essentially case based, starting from a large dictionary of names and their pronunciations, and pronouncing a new name by retrieving a relevant source name from the dictionary, and performing one of a number of prespecified transformations, such as suffix exchange (e.g., AGNANO = AGNELLI − ELLI + ANO). TTS performs well—comparably to the rule-based systems mentioned above—but like those other systems, still leaves room for improvement. The good, but imperfect performance of both the rule-based and case-based approaches to name pronunciation suggests combining the two; however, no previous, practical system has taken a true hybrid approach. Sullivan and Damper [35] have combined rules and cases in a model of human pronunciation, but their model generates either a pure rule-based or a pure case-based solution—it does not intermix RBR and CBR within a solution as Anapron does.

Name pronunciation is defined here as the task of converting an input spelling, e.g., KEIDEL, into an output pronunciation, e.g., `` k`ayd'ehl `` (rhymes with MY BELL).[7] The pronunciation is a written specification of how to pronounce the name; it could be fed through a speech synthesizer to produce an actual spoken rendition. A pronunciation includes the sequence of phones or sounds in the name, as well as the level of stress to place on each syllable. Here, the phones are `kaydehl`, while ` and ' are stress marks. The ` says to put secondary stress on `kay`. The ' means primary stress on `dehl`. The notation is taken from DECtalk™,[8] but is unimportant for purposes of this paper.

In Anapron, the task of name pronunciation is divided among six principle modules. Table 1 gives a brief account of what each module does, by way of illustration for KEIDEL. Transcription and stress assignment are the top-level modules: they contribute to the output pronunciation directly. The other modules are in service of transcription and stress. The language and morphology modules produce nondeterministic answers. Here, the language module generates two possible language classifications of the name—

[7] This is an example of an appropriately anglicized pronunciation. The native German pronunciation more nearly rhymes with IDLE [Stefanie Brüninghaus, personal communication, 1995].

[8] DECtalk is a trademark of Digital Equipment Corporation.

"Generic" or German. This nondeterminism is carried through the other modules until the selection module resolves it by choosing the German analysis. The way the selection module makes its choice is discussed below (see Section 3.3).

3.2. Application of the architecture

The architecture was applied not to the task of name pronunciation as a whole, but to its two top-level subtasks: transcription and stress assignment. This section sketches the application of the architecture to each of these subtasks. The knowledge sources are briefly described, followed by an illustration of the architecture's operation using them.

The architecture works from three knowledge sources for each task: a weak theory, a set of problem/answer pairs, and a similarity metric. The weak theory for transcription was based on the rules of the MITalk text-to-speech system [1], as well as introductory grammar texts for French, German, Italian, and Spanish. Each operator in the weak theory says how to map a letter or letter cluster into a string of phones. For instance, the operator C:s says to map the letter C to the sound s, i.e., a soft C, as in C̲ENT. The basic operation of the theory is then to work through the name, applying operators to transcribe each letter or letter cluster. The rules for choosing which operator to apply can test the letters on either side of the cluster being transcribed, the language of the name, and its morphological structure. For example, one rule recommends the C:s operator if the following letter is I, E, or Y, and the name is of Latinate origin—this is the familiar "C softening" rule. A rule can also test how surrounding letters were *transcribed*; this imposes the constraint that the surrounding letters be transcribed before the rule at issue is matched. Such constraints restrict the possible orders in which the letters of a name may be processed. Occasionally a circular dependency may arise, in which case a deadlock-resolution strategy is invoked. The weak theory for transcription has a total of 295 operators and 619 rules.

The weak theory for stress assignment is based on MITalk, the grammar texts mentioned above, and the stress theory of Liberman and Prince [24]. The goal of stress assignment is to assign a level of stress—primary, secondary, or zero (i.e., no stress)—to each syllable in the name. The weak theory starts by assigning stress to each morpheme in the name individually. This is done in two backward passes of the morpheme: the first pass makes a binary decision as to whether each syllable has zero or nonzero stress; the second pass refines these binary stress levels into a proper three-valued stress pattern. The stress patterns for the individual morphemes are then combined into a stress pattern for the whole name, based on imposing a hierarchical structure on the morphemes. The operators of the weak theory provide primitives for implementing the above procedure. For instance, two operators implement the first backward pass of assigning zero or nonzero stress to each syllable of a morpheme: `MSR`, which identifies the last syllable with nonzero stress; and `propagate`, which repeatedly jumps backward to the next syllable with nonzero stress. The rules of the weak theory control which operator is applied and how it is instantiated—e.g., how many syllables the `propagate` operator should jump back each time. The rules can test the spelling of the name, its language, morphological structure, transcription, and syllable structure. The weak theory for stress has 7 operators (not including instantiations thereof) and 29 rules.

The second knowledge source of the architecture, the set of problem/answer pairs, was derived, in the case of both transcription and stress assignment, from a pronouncing dictionary of 5000 surnames.[9] The dictionary includes the 2500 most frequent names in the US, 1250 sampled randomly from ranks 2500 through 10,000, and 1250 from ranks 10,000 to 60,000. The utility of these last two groups is to illustrate patterns that are important but that may not appear in the very common names.

The similarity metrics used in Anapron are based on broad, approximate knowledge about which factors determine a given aspect of a name's pronunciation. For transcription, there are two factors. First, the letters in the immediate vicinity of the cluster-to-be-transcribed affect the cluster's pronunciation. This is due to assimilation effects: while the mouth is pronouncing the cluster, it is anticipating the next sounds, as well as retaining aspects of the previous ones. The second factor affecting the transcription of the cluster is the overall "shape" of the name—essentially, its pattern of (orthographic) consonants and vowels. The shape affects the pronunciation in that it reflects global influences such as language and morphology. The transcription metric takes the two preceding factors into account by combining a detailed comparison of the two names immediately around the letter cluster being transcribed with a rough global comparison of the names.

The stress metric is analogous to the transcription metric: it does a careful comparison of the two names in the region that is most critical for the particular stress operator at issue, as well as a rough global comparison, to pick up on effects of language and morphology. More detailed specifications of the metrics can be found in Golding [14].

The remainder of this section illustrates how Anapron, given the abovementioned knowledge sources, pronounces names. The illustration will be for the transcription of the KEIDEL example of Table 1. Transcription of KEIDEL is actually performed twice, once assuming the name is Generic, and once assuming it is German. This example is for the German case. As mentioned above, transcription involves applying operators to the name, in some order, to map letter clusters to strings of phones. Anapron selects the operators via the RC-Hybrid procedure. Table 9 summarizes the results, disregarding the order in which the letters are actually processed. For the first letter of the name, K, RBR is invoked first, and suggests the K:k operator, which maps the letter K to the phone k (as in KITE). CBR is then invoked to propose analogies contradicting this choice of operator, but no such analogy is found. The combination module therefore applies the operator suggested by the rules, K:k. Application of the next two operators in the table, EI:ay and D:d, is similarly uneventful.

For the E, things get more interesting. The rules suggest E:ey, the default pronunciation of E in German (as in FREGE). However, CBR finds an analogy from VOGEL which suggests the E:eh operator instead. This analogy has a similarity score of 0.73. Empirical verification reveals that the generalization behind the analogy—which says to apply E:eh in German names in a particular context—applies to 7 cases in the case library: EDELBROCK, FOGEL, GEIBEL, LOGEL, SCHNABEL, SPEIDEL, and of course VOGEL. All 7 have E:eh applied. Thus the accuracy of the analogy is 7/7 = 1.00. The significance works out to be 0.71. The way the thresholds were set, the analogy is

[9] This dictionary was kindly provided by Bellcore for purposes of this research.

Table 2
Transcription of KEIDEL under the German analysis; the table shows the transcription operators recommended for each letter of the name by RBR, CBR, and combination

	RBR	CBR	Combination
K	K:k	-	K:k
E I	EI:ay	-	EI:ay
D	D:d	-	D:d
E	E:ey	E:eh	E:eh
L	L:l	-	L:l

deemed compelling. Thus the combination module selects E:eh, overriding the rules by the analogy with VOGEL.

For the final L of the name, the rules suggest L:l, which again goes unchallenged. Thus the output of the transcription module for the German analysis of KEIDEL is kaydehl—as opposed to kaydeyl, which is what the rules alone would have said.

3.3. *Positive analogies*

As mentioned above, the language and morphology rules in Anapron return multiple answers, due to the difficulty of uniquely analyzing these aspects of a name. This rule nondeterminism is resolved by the selection module, through the method of *positive analogies*. The idea of the method is to use the positive exemplars in the case base to reinforce correct rule applications, just as negative exemplars were used to detect incorrect rule applications. The method starts with multiple candidate pronunciations of a name, corresponding to the different ways of applying the language and morphology rules. It evaluates each candidate pronunciation by seeing if the operators that were applied in deriving that pronunciation seem to have been applied correctly. It does this by drawing analogies between each operator application in the pronunciation and the positive exemplars of the rule that recommended that operator. The closer the operator application is to a previously seen, correct application, the more favorable the system's evaluation of that operator will be. Specifically, the score for an operator is the similarity score of the best analogy found. The overall score for a pronunciation is the average of the scores of its transcription and stress operators.[10]

Table 3 shows how positive analogies were used in the KEIDEL example. Only the analogies for transcription operators are shown. On these, the German analysis outscored the Generic analysis; the same turns out to be true of the overall scores, which is why the German analysis was ultimately selected. The main reason the German analysis did

[10] In addition, a pronunciation may receive bonuses or penalties assigned by the rules. The most common type of bonus is when a name contains a prefix or suffix characteristic of a particular language. For instance, the name ROCHAMBEAU has the characteristic French ending -EAU; the French analysis of this name therefore receives a bonus. These rule-based scores complement the analogy-based scores, and enable the system to decide among competing pronunciations of a name even in the absence of a case library, albeit in a less informed way.

Table 3
Positive analogies for transcription operators of the German and Generic analyses of KEIDEL

(a) German analysis			(b) Generic analysis		
Op	Analogy	Score	Op	Analogy	Score
K:k	KESLER → KEIDEL	0.67	K:k	KEELER → KEIDEL	0.88
EI:ay	SPEIDEL → KEIDEL	0.67	EI:iy	REID → KEIDEL	0.00
D:d	SPEIDEL → KEIDEL	0.91	D:d	RIEDEL → KEIDEL	0.71
E:eh	VOGEL → KEIDEL	0.73	E:e	RIEDEL → KEIDEL	0.86
L:l	GEIBEL → KEIDEL	0.91	L:l	RIEDEL → KEIDEL	0.88
	Transcription score:	0.76		Transcription score:	0.56

better on transcription operators is that the Generic analysis had little support for its EI:iy operator; REID was used, but scored poorly due to its global dissimilarity from KEIDEL. The name RIEDEL, while valuable elsewhere in the Generic analysis, could not help with EI:iy, because its I and E are in the wrong order. One other point concerns the E:eh operator in the German analysis: this operator was applied by analogy, not by a rule; thus there are no positive exemplars on which to base its score. Instead, the score of such an operator is taken to be the similarity score of the analogy that suggested it—in this case, the VOGEL/KEIDEL analogy.

4. Evaluation

Anapron's performance was evaluated in three phases. The goal of the first phase was to gain a quantitative understanding of system performance: a profile was taken of how active each part of the system was in practice, and any deviations from the expected performance were analyzed. The second phase stepped back from this internal analysis of the system and looked at the "bottom line": how does the performance of the rule/case hybrid approach, as embodied in Anapron, compare to that of other approaches? Commercial systems, other research systems, and humans were included in the comparison. Once the overall performance of Anapron was ascertained, the third phase was to understand how it achieved this performance, by evaluating the contribution of each of its components. This involved systematically modifying each component, and measuring the impact on system performance. The sections below discuss the three phases.

4.1. Exploratory measurements

Exploratory measurements of Anapron were taken to get a quantitative picture of its operation, and to detect any deviations from the expected behavior. Two main findings emerged: (1) the system found fewer *strong* analogies for rare names than for common names, although the total number of analogies, strong or weak, remained constant; and (2) the system's criterion for analogical compellingness was too strict. The sections below present the test set that the exploratory measurements were based on, and the measurements that were made, together with the resulting findings. The measurements are grouped by whether they were purely objective, or included a subjective component.

4.1.1. *Test set*

The test set for this and the other experiments was drawn from the Donnelley corpus, a database of over 1.5 million distinct surnames covering 72 million households in the US. Names in Donnelley range from extremely common (e.g., SMITH, which occurs in over 670,000 households) to extremely rare (e.g., BOURIMAVONG, which occurs in 1 household). The number of households that have a particular name will be referred to as the *frequency* (of occurrence) of the name.

Test sets were constructed from Donnelley by selecting points of interest along the frequency spectrum, and randomly sampling an appropriate number of names at each point. If Donnelley had fewer than the desired number of names at some frequency f, then the names were selected randomly from the narrowest symmetric frequency band around f that was big enough. The test set for the objective measurements contains 13 exponentially distributed frequencies: $1, 2, 4, 8, \ldots, 4096$. The frequencies were distributed exponentially because this yields evenly spaced measurements of Anapron's behavior—this was determined in a pilot study, which showed that Anapron's percentage of acceptable pronunciations drops linearly as frequency is decreased exponentially. The test set has a total of 10,000 names, with between 250 and 1000 at each frequency. These numbers represent a tradeoff between the cost of running the test, and the size of the confidence intervals in the resulting measurements. The names were chosen to be disjoint from Anapron's dictionary, since names pronounceable by rote lookup are unrepresentative of system behavior.

4.1.2. *Objective measurements*

Objective measurements were made for both the rule-based and case-based parts of the system. The rule-based measurements counted how many operators were applied by each module—language, morphology, transcription, syllable structure, and stress assignment. The case-based measurements counted how many analogies were proposed, accepted, and rejected, and for what reason, where the reason corresponds to the way the analogy satisfied or failed to satisfy the compellingness predicate. All measurements were broken down by name frequency, to see how the system's behavior changes as the names get rarer and thus more difficult to pronounce.

The main finding from the objective measurements was an effect termed the *analogical decline*. It says that as name frequency decreases, the number of *highly plausible* analogies to the name decreases, where a highly plausible analogy is one with a very high similarity score (this notion will be made more precise below). Fig. 10 shows the transcription data on which the analogical decline is based. It plots the number of transcription analogies as a function of name frequency. It is split into two graphs—one for accepted analogies, and one for rejected analogies. The accepted analogies in turn are broken down into two reasons for acceptance, denoted *significant*, and *highly plausible*. These correspond to which of the two disjuncts the analogy satisfies in the last clause of the definition of compellingness.[11] The definition of compellingness is repeated here for convenience:

$$\begin{aligned}
&\text{Compelling-p}(\mathcal{A}) \iff \\
&\quad \text{similarity-score}(\mathcal{A}) \geqslant SS_0 \\
&\quad \textbf{and } \text{accuracy}(\mathcal{A}) \geqslant A_0 \\
&\quad \textbf{and } (\text{significance}(\mathcal{A}) \geqslant S_0 \textbf{ or } \text{similarity-score}(\mathcal{A}) \geqslant SS_+).
\end{aligned}$$

Since highly plausible analogies match the last disjunct of this definition, they can now be seen to be those compelling analogies whose similarity score is SS_+ or greater. Rejected analogies, like the accepted analogies, are broken down into two groups, this time for the two reasons for rejection: *inaccurate*, and *unsupported*. These correspond to whether the analogy failed to satisfy the second or third conjunct of compellingness.[12]

To test for upward or downward trends in these curves, a Spearman rank-correlation test [10] was run on each. The results were that the curve of highly plausible analogies was found to decrease ($p(r_s) < 0.01$, $p(D) < 0.01$), but no other significant trend was found. This means that the system found fewer highly plausible analogies for rare names. Note, however, that this does not mean that case-based reasoning is useless for rare names—it is merely less effective at finding highly plausible analogies. In fact, the number of "normal plausibility" analogies does not decrease significantly, as demonstrated by the absence of a decreasing trend in the curve of significant analogies, which counts all accepted analogies other than highly plausible ones. A further investigation of the analogical decline can be found in Golding [14].

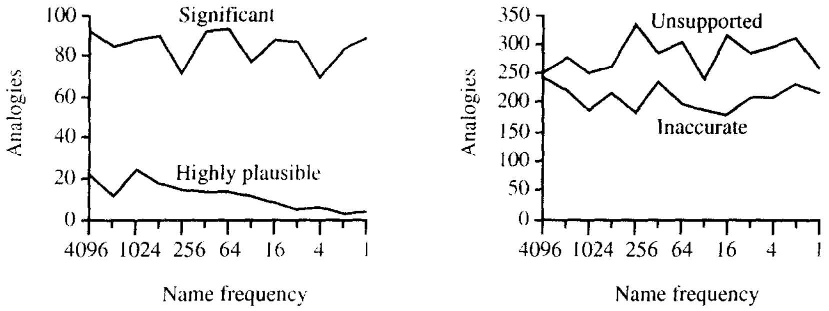
4.1.3. Subjective measurements

Subjective measurements of the system's behavior were made not on the 10,000-name test set described above, but on a scaled-down 1000-name version. This was necessary to make it feasible to obtain human judgements. The 1000-name test set had 250 names at each of four (roughly) exponentially distributed frequencies: 1, 32, 256, and 2048.

The subjective measurements consisted of judgements, for each name, about the acceptability of the following: the overall pronunciation, the individual transcription and stress operators applied, the choice of language/morphology analysis, and the analogies proposed (whether accepted or rejected). The judgements were made by the first author. To facilitate this rather laborious process, a *judgement editor* was used, which provided a graphical user interface for entering or changing judgements about a name. The editor also verified that the judgements for a name were complete and consistent.

[11] Analogies matching both disjuncts are counted as highly plausible—this reflects the system's processing of such analogies. After looking at their similarity score and accuracy, the system declares them compelling for reason of high plausibility. It has no reason to check further whether they are also significant.

[12] Analogies that fail to satisfy both conjuncts are counted as inaccurate, again because this can be determined from the similarity score and accuracy, without having to test whether they are unsupported. Also, there is technically a third reason for rejection, *implausible*, for analogies that have a similarity score less than SS_0, and thus fail to satisfy the first conjunct of compellingness. Most implausible analogies are never generated by Anapron; the system has been optimized to not retrieve the very distant analogs that would give rise to such analogies. Consequently, implausible analogies cannot be accurately counted, and are omitted from Fig. 10.



(a) Accepted analogies. (b) Rejected analogies.

Fig. 10. Number of transcription analogies as a function of name frequency. Each of the four curves plots the number of analogies that were accepted or rejected for a particular reason. The number of names at each frequency has been scaled to 1000.

The main result of the subjective measurements was that errors of analogical omission (helpful analogies that were missed) were found to greatly outnumber errors of analogical commission (harmful analogies drawn). This suggests that the system's analogical compellingness criterion may have been too strict. This could be fixed by lowering the system's thresholds,[13] thereby relaxing the compellingness criterion, or by re-working the similarity metrics to allow better discrimination between good and bad analogies.

4.2. *System comparison*

In the second phase of the evaluation, Anapron was compared with a variety of other name-pronunciation systems to see how the performance of the rule/case hybrid method compares with that of alternative approaches. Seven other systems were used in the comparison: three state-of-the-art commercial systems, two versions of a machine-learning system (NETtalk), and two humans. The commercial systems are the same ones mentioned earlier (see the beginning of Section 3): the Orator™ system from Bellcore and DECvoice from DEC, both of which are rule based, and TTS from Bell Labs, which is case based. The two versions of NETtalk are BP-legal, which is the vanilla version of NETtalk [33], and BP-block, which is NETtalk enhanced with a block-decoding postprocessor [11]. The sections below sketch the test set, design, and results of the experiment. A more complete presentation can be found in Golding and Rosenbloom [16].

4.2.1. *Test set*

The test set for the system comparison was similar to that used in the subjective measurements, except that: (1) only 100 names (not 250) were chosen at each frequency,

[13] The results of the threshold modification study suggest that the most effective threshold to lower would be SS_0; see Section 4.3.

to reduce the burden on the human test subjects; and (2) the test set was no longer constrained to be disjoint from Anapron's dictionary, as an unbiased measurement of system performance includes names both in and out of the dictionary.

4.2.2. *Design*

Because there is no objective criterion of correctness for name pronunciations, a pronunciation was evaluated by asking human test subjects whether they found it acceptable. Each system was run on the 400-name test set described above. The output of the computer systems was collected in the form of written pronunciations; the output of the human pronouncers was tape-recorded and transcribed as written pronunciations. The two versions of NETtalk were trained on Anapron's 5000-name pronouncing dictionary.

A cassette tape was made of the pronunciations generated by all systems. This involved, for each name, eliminating duplicate pronunciations, and permuting the remaining pronunciations randomly. The order of names in the test set was permuted randomly as well. To hide the identities of the systems, all pronunciations were read by the DECtalk speech synthesizer. A panel of 14 human test subjects listened to the cassette tape and rated the acceptability of each pronunciation.

4.2.3. *Results*

The main results of the system comparison appear in Fig. 11. The names of the commercial systems and humans have been omitted as their identities are not relevant here. The figure gives the percentage of acceptable scores, out of a total of 5600, awarded to each system ($5600 = 14$ judges $\times$ 400 pronunciations). The scores are broken down by name frequency. The figure includes an imaginary ninth system, labelled Ubound, which generates for each name the pronunciation that received the greatest number of acceptable votes from the judges. It measures the degree to which all judges can be pleased simultaneously, using just the pronunciations available from the eight systems tested.

Fig. 11 shows that Anapron performs almost at the level of the commercial systems, and substantially better than the two versions of NETtalk. Also, although the eight systems seem to hit a performance asymptote at 93%, the Ubound system demonstrates that it is possible to score at least 97%. This suggests that there is room for improvement in all systems.

To detect whether the differences between Anapron and the other systems were statistically significant, an ANOVA was run, followed up by a Bonferroni multiple comparison procedure. The results are shown in Fig. 11 as annotations on the scores in the table. Overall, Anapron outperformed the two versions of NETtalk, but the commercial systems, humans, and Ubound did better than Anapron. However, in some frequency ranges, a significant difference between Anapron and certain commercial systems could not be detected.

Given that Anapron is able to exploit two knowledge sources, while the other computer systems use just one, it may be surprising that Anapron did not outperform the commercial systems. It should be borne in mind, however, that Anapron's knowledge sources were put together as rapidly as possible from whatever rules and cases could

System	Name frequency				Overall
	2048	256	32	1	
Ubound	98 +	98 +	98 +	96 +	97 +
Human1	97 +	93 +	93 +	88 +	93 +
Human2	98 +	94 +	94 +	86 +	93 +
Com1	97 +	95 +	93 +	90 +	93 +
Com2	96 +	90 +?	87 +?	86 +	90 +
Com3	96 +	94 +	89 +	78 –?	89 +
Anapron	**91**	**88**	**85**	**80**	**86**
BP-block	84 –	83 –	77 –	69 –	78 –
BP-legal	78 –	72 –	66 –	52 –	67 –

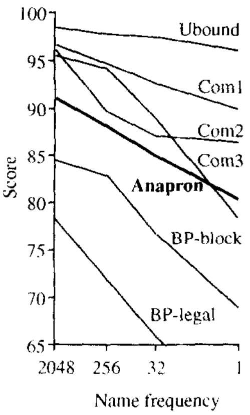


Fig. 11. Percentage of acceptable scores for each system, broken down by name frequency. The data are shown as a table and as a graph. Scores in the table have a plus sign (+) if higher than Anapron's score, or a minus sign (–) if lower. All differences are significant at the 0.01 level, except those marked with a question mark (?), which are not significant even at the 0.10 level. The humans were omitted from the graph to avoid clutter; their curves would lie between those of Ubound and Com2. The curve for BP-legal was truncated when it ran off the bottom of the graph.

be obtained—basically the MITalk rules and a 5000-name pronouncing dictionary. The commercial systems, in contrast, use extremely high-quality, and unfortunately proprietary, knowledge sources—carefully tuned rule sets for the rule-based systems, and a dictionary of over 40,000 names for the case-based system. Anapron was in fact found to improve on the performance of its rules or cases alone (see Section 4.3); it would appear, however, that in the system comparison, this improvement was outweighed by the mediocre quality of the rules and cases used. Thus while Anapron provides a proof of concept of the architecture—a demonstration that combining rules and cases improves performance—actually using this improvement to outperform commercial systems must wait until such time as commercial-quality knowledge sources can be obtained for testing.

4.3. Modification studies

To gauge the contribution of Anapron's components to its overall performance, a set of experiments was performed in which various components were modified, and the effects on system performance were observed. Five such studies were run, modifying: rules and cases, thresholds, language knowledge, morphology knowledge, and syllable-structure knowledge. The first study—on rules and cases—directly investigated the effects of combining rule-based and case-based reasoning. It provided the key result that the system achieved higher accuracy by combining rules and cases than it could have achieved with either one alone. The threshold study tested how sensitive the system's performance was to the threshold settings used in the definition of analogical compellingness—i.e., SS_0, SS_+, A_0, and S_0. Extreme raising or lowering of any one threshold was generally found to hurt accuracy, although lowering SS_0 sometimes improved accuracy at the expense of increasing run time. The remaining three studies concerned the system's support knowledge—i.e., knowledge needed in support of the two top-level tasks, transcription and stress. Degrading the language or morphology knowledge sufficiently was found to have a substantial negative impact on system accuracy, while degrading syllable-structure knowledge had a relatively minor effect. These studies are described more fully in Golding [14].

The rule/case modification study [15] is the subject of the rest of this section. The test set, design, and results are discussed below.

4.3.1. Test set

Like the system comparison, the rule/case experiment required a great deal of human effort in the evaluation. The test set was therefore made the same size as in the system comparison—100 names at each of four frequencies. The only difference was that, as in the exploratory measurements, the test set was constrained to be disjoint from Anapron's dictionary, since again rote lookup behaviors were not of interest.

4.3.2. Design

The rule/case study involved independently varying the strength of the system's rules and cases. For each combination of rule strength and case strength, the system was run on the 400-name test set, and its accuracy and run time were recorded. Accuracy was measured as the proportion of acceptable pronunciations generated by the system, where acceptability was judged by the first author.[14] All judgements were cached and re-used if a pronunciation recurred, to help enforce consistency across trials. Run time was the average time, in seconds, for the system to pronounce a name in the test set. The system, written in CommonLisp, was run on a Texas Instruments Microexplorer with 8M memory.

The rules were set to four different strengths: 0, 1/3, 2/3, and 1. A strength of 1 means all transcription and stress rules were retained in the system. Strength 0 means that all rules were deleted except *default* rules. The default rules transcribe a letter or assign stress if no other more specific rule matches. The default rules cannot be deleted, otherwise the system would be unable to generate a complete pronunciation for some names. Retaining the default rules corresponds to keeping 137 of the 619 transcription rules and 16 of the 29 stress rules. Rule strengths between 0 and 1 correspond to retaining a proportional number of nondefault rules in the system. Each strength is obtained by deleting a random subset of the nondefault rules from the next higher strength.

The cases were set to six strengths: 0, 1000, 2000, 3000, 4000, and 5000. The strength is just the number of names that were kept in the case library. Again, each weakening of the case library produces an arbitrary subset of the previous case library.

4.3.3. Accuracy results

Fig. 12 shows system accuracy as a function of both rule strength and case strength. The main result is that accuracy improves monotonically as rule or case strength in-

[14] The first author was an unusually harsh judge, thus the scores here are not directly comparable to those of the system comparison.

Rule	Case strength (× 1000)					
strength	0	1	2	3	4	5
1	56	65	65	67	67	68
2/3	46	54	56	56	57	59
1/3	33	39	43	44	46	47
0	19	27	32	34	35	36

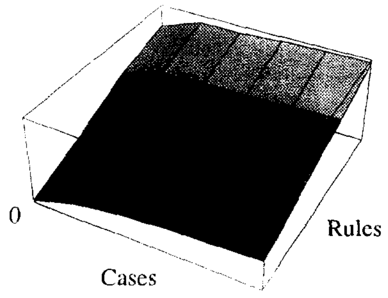


Fig. 12. System accuracy, shown as a table and as a 3D graph. Each value is the percentage of names in the test set for which the system produced an acceptable pronunciation.

Rule	Case strength (× 1000)					
strength	0	1	2	3	4	5
1	1.3	3.0	4.1	5.2	6.3	7.2
2/3	1.2	3.1	4.4	5.8	7.2	8.5
1/3	1.1	3.0	4.4	5.9	7.6	9.3
0	0.9	3.0	4.5	6.3	8.2	10.2

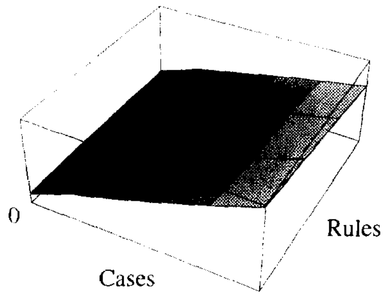


Fig. 13. System run time, shown as a table and as a 3D graph. Each value is the average time (in seconds) for the system to pronounce a name in the test set.

creases. The total improvement in accuracy due to adding rules is between 32% and 38% of the test set (depending on case strength). For cases it is between 12% and 17% (depending on rule strength). This shows that rules and cases each contribute to the system's overall accuracy. It is only by having both knowledge sources that the system is able to achieve its best performance.

4.3.4. Run-time results

Fig. 13 gives the results on run time. The interesting point here is that when the case library is large, adding rules to the system actually decreases run time. For example, with the case library at size 5000, increasing the rules from strength 0 to 1 lowers run time from 10.2 to 7.2 seconds per name. The reason is that adding rules to the system improves the overall accuracy of the rules, barring sociopathic effects. When the rules are more accurate, they will have fewer exceptions. This translates into fewer negative exemplars, and thus fewer opportunities to draw analogies. The foregone analogies result in a corresponding savings in run time. In short, adding rules to the system reduces the load on the CBR component. This demonstrates that RBR and CBR do not merely exist side by side in the architecture; they interact beneficially.

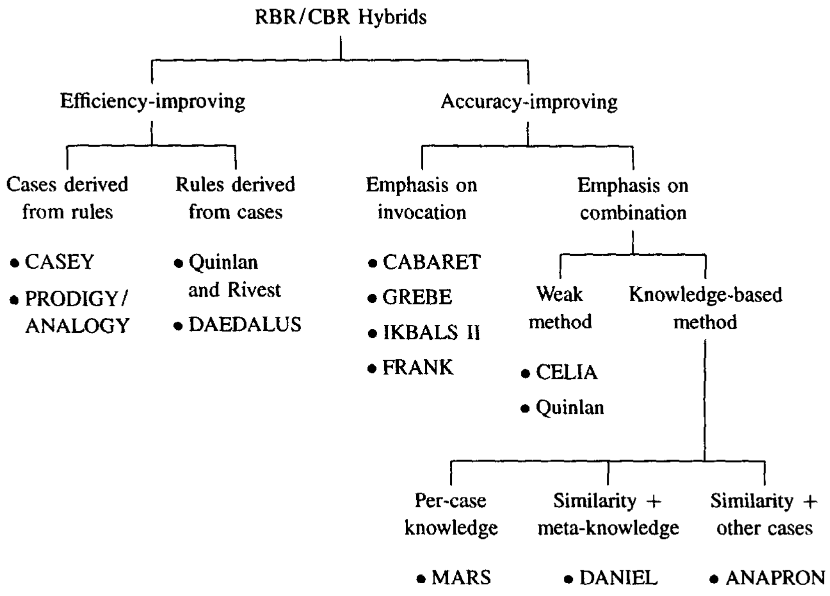


Fig. 14. Taxonomy of methods for combining RBR and CBR. Examples of systems that use each method are listed at the leaf nodes.

5. Related work

A number of other methods have been proposed for combining RBR and CBR. Each method is designed around a particular set of knowledge requirements; for instance, some methods expect independent rules and cases, while others start with just one knowledge source and derive the other from it. Methods also differ in their approach to integrating rules and cases; some focus on how and when RBR and CBR can each be profitably invoked, while others concentrate on how to reconcile conflicting results of RBR and CBR. Fig. 14 organizes the methods into a hierarchy according to these differences.

The first branching point tests whether the rules and cases used by the method are dependent or independent. If the rules and cases are dependent, it means that one was derived from the other. Such methods are labelled as efficiency-improving; their primary motivation is to express their knowledge in whatever form will make problem solving most efficient. The methods with independent rules and cases are labelled as accuracy-improving; the primary motivation here is to maximize problem-solving accuracy by exploiting multiple knowledge sources.

The efficiency-improving methods can be further broken down according to which of their knowledge sources was derived from which. Most CBR systems that include a rule component have cases that are derived from their rules. The cases are records of how the rules were applied to particular examples encountered previously. By reasoning from cases, the systems bypass the potentially lengthy process of solving a new problem from scratch via the rules. For example, CASEY [22] works in the domain of heartfailure diagnosis. It has a complete but slow set of rules—in the form of a causal model—for diagnosing heart failures. When given a new case to diagnose, it tries to relate the case to a similar case diagnosed previously. When it can find such a case, its answer usually agrees with what the causal model would have said, but is obtained an average of two orders of magnitude faster. PRODIGY/ANALOGY [36] can be regarded as a general architecture for combining RBR and CBR to improve efficiency. PRODIGY/ANALOGY's equivalent of rule-based reasoning is problem solving via search; its version of case-based reasoning is derivational analogy [8].

The systems whose rules are derived from their cases extract the rules by some generalization procedure. The systems must still keep the cases around, because their rules do not encode all of the knowledge in the cases. The rules in these systems can serve various purposes, such as enabling a more compact representation of the data, as in Quinlan and Rivest [28], or providing more efficient access to the cases, as in Daedalus [2].

Systems utilizing independent rules and cases are much closer in spirit to Anapron. Again the systems fall into two groups. The first group emphasizes how and when to *invoke* the RBR and CBR components; the second group emphasizes how to *combine* the results once the components have been invoked.

Systems that emphasize invocation include CABARET [32], GREBE [4], IKBALS II [39], and FRANK [31]. These systems are designed to gather evidence to support a user's position. CABARET, GREBE, and IKBALS II work in the domain of legal reasoning, finding support for one side or the other in a legal case. FRANK has been applied to the task of diagnosing back injuries, and generates medical reports reflecting the user's expository goals (e.g., downplay the seriousness of the injury; or give a balanced account of the evidence). Because these systems are not intended to make an actual decision about whether the user's position is right or wrong, they do not have to resolve conflicts between RBR and CBR; they merely report all of the evidence. The effort in these systems therefore goes not into combining the results of RBR and CBR, but into determining when RBR and CBR can each be profitably invoked to contribute to the target problem. CABARET uses a set of heuristics for this purpose, such as "If a rule fires with an undesired conclusion, invoke CBR to find cases that discredit the rule". GREBE uses a control strategy that calls on CBR to operationalize abstract rule antecedents, and calls on RBR to establish and elaborate matches between a source and target case. IKBALS II starts with the rules, only invoking CBR when it encounters an open-textured term that cannot be interpreted by further rule chaining. FRANK uses blackboard-based opportunistic control to select the most appropriate reasoning method to apply to a particular subgoal.

In the second group of systems, the focus is on reconciling the conclusions of RBR and CBR. The reconciliation can be done either by a weak method—i.e., a general-purpose method that does not require knowledge of the domain—or a knowledge-based method. CELIA [29] and Quinlan's method [27] are two examples of using a weak method. CELIA is a learning apprentice in the domain of automobile repair. A central part of the system's function as a learning apprentice is to watch an expert mechanic and predict the expert's next step. Prediction is done using two knowledge sources: abstract general knowledge, and cases. The abstract general knowledge is that of a novice mechanic, and is thus assumed incomplete and buggy. The cases, in contrast, represent actual troubleshooting sequences by an expert, and are considered highly reliable. To predict the expert's next step, CELIA applies its buggy model, and, independently, looks for an analogous case on which to base its prediction. If it is able to come up with a prediction based on a case, it listens to it, else it falls back on the rule-based prediction. This illustrates a way of integrating abstract general knowledge and cases under the assumption of incomplete, buggy general knowledge. It is a weak method because it does not use domain knowledge to decide between rules and cases, but rather simply prefers cases whenever they are applicable.

Quinlan's method [27] applies to tasks whose answer is a numeric quantity. It uses an instance-based scheme to generate an initial answer; this step corresponds to CBR. The simplest instance-based scheme is to retrieve the source case that is closest (by some metric) to the target, and just copy its answer. Quinlan's method improves on this answer by using a model, M, to add a correction term; this step corresponds to RBR. Let:

- T = the target problem,
- S = the source case retrieved by the instance-based scheme,
- $A(S)$ = the answer given by the source case,
- $M(T)$ = the answer obtained by applying the model to the target,
- $M(S)$ = the answer obtained by applying the model to the source.

The pure instance-based scheme would give the answer $A(S)$. But Quinlan's combined method gives $A(S) + (M(T) - M(S))$. The parenthesized correction term helps account for differences between the source and target problems. This answer is thus obtained numerically from the results of RBR and CBR; no domain knowledge is needed.

Systems that take a knowledge-based approach to combining the results of RBR and CBR include Anapron, MARS [12], and DANIEL [6]. The main distinction among these systems is in the type of knowledge they use to do the combination. MARS combines evidence from multiple rules and cases using possibilistic reasoning. This requires that all of its knowledge be represented as possibilistic rules; thus MARS's first step is to convert its cases into this form. The conversion requires certain knowledge about each case: the features of the case that are relevant to its outcome, and the necessity and sufficiency with which this outcome is implied. This per-case knowledge enables MARS to represent each case as a rule and subsequently aggregate evidence from rules and cases via possibilistic reasoning. In MARS's domain of mergers and acquisitions, the per-case knowledge is acquired via natural-language processing of a document that explains the judge's ruling on each case.

DANIEL combines CBR and RBR for legal interpretation. DANIEL explicitly addresses conflicts between rules and cases by invoking a *rule-based coordination component*. This component decides between CBR and RBR using two sources: domain meta-knowledge—in particular, the legal binding force of the rule-based and case-based arguments, and the degree of open-texturedness of the predicates involved—and the similarity between the source and target cases.

In Anapron, decisions between RBR and CBR are based on the compellingness of the analogy. Compellingness depends on two factors: the similarity between source and target, and an empirical verification, which tests the generalization behind the analogy on other cases in the case library.

It can be seen that MARS, DANIEL, and Anapron each depend on different kinds of knowledge to arbitrate between RBR and CBR. The systems are therefore applicable in different situations: when it is practical to do the knowledge engineering of cases that MARS requires, MARS is appropriate. When domain meta-knowledge is available for evaluating the strength of a case-based or rule-based argument, and when it is practical to specify a similarity metric, DANIEL is appropriate. When a large supply of cases is available for testing out an analogy, and again when a similarity metric can be specified, Anapron is appropriate.

6. Conclusion

An architecture was presented for improving system accuracy by bringing together knowledge in two forms: rules and cases. The architecture is intended for domains that are understood well but not perfectly. The idea is that in such domains, expert knowledge in the form of rules can be used to provide a skeletal method for solving problems; cases are then used to flesh out the method by covering idiosyncrasies and special cases that were not anticipated by the rules. In addition to a reasonably accurate and efficient set of rules to serve as a starting point for problem solving, the architecture also needs knowledge in support of CBR—namely, a set of cases and a similarity metric. The set of cases should be extensive enough to illustrate the errors in the rules; any unillustrated problems cannot be corrected.

The architecture was applied to the task of name pronunciation. With minimal knowledge engineering, it was found to perform almost at the level of state-of-the-art commercial systems. More importantly, a modification experiment showed that its performance was higher than what it could have achieved with its rules or cases alone. This demonstrates the capacity of the architecture to improve upon a pure rule-based or case-based system. In addition to the accuracy benefits, having rules together with the cases allowed two innovations in CBR technology: first, the rules provided a natural way to index the cases (prediction-based indexing); and second, they provided a method of doing case adaptation, termed "case adaptation by factoring".

The architecture presented here is one datapoint in a hierarchy of possible hybrid approaches. One way to abstract away from its design is to keep the same reasoning components (RBR and CBR), but to combine them differently. The method of combination could be tailored to whatever knowledge is available in the domain, whether analytic (e.g., heuristics about when to believe RBR versus CBR) or empirical (e.g., examples of previous decisions combining RBR and CBR). Another way to abstract away from the architecture is to replace its RBR component with some other reasoning method. CBR then becomes a postprocessor to improve an approximate answer obtained by any method of choice. The downside, however, is that the benefits of having rules together with cases would be lost—alternative methods of case indexing and case adaptation would be needed. A final level of abstraction, and the one that is in fact the essence of the work presented here, is simply to combine multiple independent knowledge sources to achieve higher accuracy.

Acknowledgements

The research reported here has its roots in the Soar architecture [23]. Soar provided both a model of problem solving and a vehicle for the initial implementations. The other important influence on the work is due to Ken Church and Mark Liberman, who suggested the problem of name pronunciation and proposed the approach of pronunciation by analogy. Murray Spiegel also provided generous assistance on the pronunciation aspects of the work, including arranging for us to borrow a pronouncing dictionary of names from Bellcore. We are grateful to the many people who helped with the experiment in which Anapron was compared to other name-pronunciation systems: Mark Liberman, Cecil Coker, Murray Spiegel, Tony Vitale, Tom Dietterich, John Laird, Connie Burton, and David Shapiro. Finally, we would like to thank the *Artificial Intelligence* reviewers and Bill Freeman for their helpful comments on drafts of this article.

This research was sponsored by NASA under cooperative agreement number NCC 2-538, and by a Bell Laboratories PhD fellowship to the first author. Computer facilities were partially provided by NIH grant LM05208. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA, the US Government, Bell Laboratories, or the National Institute of Health.

References

[1] J. Allen, M.S. Hunnicutt and D. Klatt, *From Text to Speech: The MITalk System* (Cambridge University Press, Cambridge, 1987).

[2] J.A. Allen and P. Langley, A unified framework for planning and learning, in: *Proceedings Workshop on Innovative Approaches to Planning, Scheduling, and Control*, San Diego, CA (1990).

[3] R. Barletta and W. Mark, Explanation-based indexing of cases, in: *Proceedings CBR Workshop*, Clearwater Beach, FL (1988).

[4] L.K. Branting, Building explanations from rules and structured cases, *Int. J. Man-Mach. Stud.* **34** (6) (1991).

[5] J.S. Brown and R.B. Burton, Diagnostic models for procedural bugs in basic mathematical skills, *Cognit. Sci.* **2** (1978).

[6] S. Brüninghaus, DANIEL: integrating case-based and rule-based reasoning in law, in: *Proceedings AAAI-94*, Seattle, WA (1994).

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

[8] J.G. Carbonell, Derivational analogy: a theory of reconstructive problem solving and expertise acquisition, in: *Machine Learning: An Artificial Intelligence Approach* **2** (Morgan Kaufmann, Los Altos, CA, 1986) 371–392.

[9] C.H. Coker, K.W. Church and M.Y. Liberman, Morphology and rhyming: two powerful alternatives to letter-to-sound rules for speech synthesis, in: *Conference on Speech Synthesis* (European Speech Communication Association, 1990).

[10] W. Daniel, *Applied Nonparametric Statistics* (PWS-Kent, Boston, MA, 1990).

[11] T.G. Dietterich, H. Hild and G. Bakiri, A comparative study of ID3 and backpropagation for English text-to-speech mapping, in: *Proceedings Seventh International Workshop on Machine Learning*, Austin, TX (1990).

[12] S. Dutta and P. Bonissone, Integrating case based and rule based reasoning: the possibilistic connection, in: *Proceedings Sixth International Conference on Uncertainty in AI*, Cambridge, MA (1990).

[13] J.L. Fleiss, *Statistical Methods for Rates and Proportions* (Wiley, New York, 1981).

[14] A.R. Golding, Pronouncing names by a combination of rule-based and case-based reasoning, PhD thesis, Stanford University, Stanford, CA (1991).

[15] A.R. Golding and P.S. Rosenbloom, Improving rule-based systems through case-based reasoning, in: *Proceedings AAAI-91*, Anaheim, CA (1991).

[16] A.R. Golding and P.S. Rosenbloom, A comparison of Anapron with seven other name-pronunciation systems, *J. Am. Voice Input/Output Soc.* **14** (1993).

[17] F. Hayes-Roth, D. Waterman and D. Lenat, eds., *Building Expert Systems* (Addison-Wesley, Reading, MA, 1983).

[18] H.A. Kautz and J.F. Allen, Generalized plan recognition, in: *Proceedings AAAI-86*, Philadelphia, PA (1986).

[19] D.H. Klatt, Review of text-to-speech conversion for English, *J. Acoust. Soc. Am.* **82** (3) (1987).

[20] J. Kolodner, *Case-Based Reasoning* (Morgan Kaufmann, San Mateo, CA, 1993).

[21] J.L. Kolodner, Retrieving events from a case memory: a parallel implementation, in: *Proceedings CBR Workshop*, Clearwater Beach, FL (1988).

[22] P. Koton, Reasoning about evidence in causal explanations, in: *Proceedings AAAI-88*, St. Paul, MN (1988).

[23] J.E. Laird, A. Newell and P.S. Rosenbloom, Soar: an architecture for general intelligence, *Artif. Intell.* **33** (1987) 1-64.

[24] M. Liberman and A. Prince, On stress and linguistic rhythm, *Linguist. Inquiry* **8** (2) (1977).

[25] S. Morris and P. O'Rorke, An approach to theory revision using abduction, in: *Proceedings AAAI Spring Symposium on Automated Abduction*, Stanford, CA (1990).

[26] H.T. Ng and R.J. Mooney, A first-order horn-clause abductive system and its use in plan recognition and diagnosis, submitted.

[27] J.R. Quinlan, Combining instance-based and model-based learning, in: *Proceedings Tenth International Workshop on Machine Learning*, Amherst, MA (1993).

[28] J.R. Quinlan and R.L. Rivest, Inferring decision trees using the minimum description length principle, *Inform. Comput.* **80** (1989).

[29] M.A. Redmond, Learning by observing and understanding expert problem solving, PhD thesis, Tech. Rept. GIT-CC-92/43, Georgia Institute of Technology, Atlanta, GA (1992).

[30] C.K. Riesbeck and R.C. Schank, *Inside Case-Based Reasoning* (Lawrence Erlbaum, Hillsdale, NJ, 1989).

[31] E.L. Rissland, J.J. Daniels, Z.B. Rubinstein and D.B. Skalak, Case-based diagnostic analysis in a blackboard architecture, in: *Proceedings AAAI-93*, Washington, DC (1993).

[32] E.L. Rissland and D.B. Skalak, CABARET: rule interpretation in a hybrid architecture, *Int. J. Man-Mach. Stud.* **34** (6) (1991).

[33] T.J. Sejnowski and C.R. Rosenberg, Parallel networks that learn to pronounce English text, *Complex Syst.* **1** (1987).

[34] M.F. Spiegel and M.J. Macchi, Synthesis of names by a demisyllable-based speech synthesizer (Orator), *J. Am. Voice Input/Output Soc.* **7** (1990).

[35] K. Sullivan and R. Damper, A psychologically-governed approach to novel-word pronunciation within a text-to-speech system, in: *Proceedings ICASSP*, Albuquerque, NM (1990).

[36] M.M. Veloso, Learning by analogical reasoning in general problem solving, PhD thesis, Tech. Rept. CMU-CS-92-174, Carnegie Mellon University, Pittsburgh, PA (1992).

[37] A.J. Vitale, An algorithm for high accuracy name pronunciation by parametric speech synthesizer, *J. Comput. Linguist.* **17** (3) (1991).

[38] T. Vitale, The automation of name and address output as a utility for the telecommunications industry, in: *Proceedings National Communications Forum* (1989).

[39] G. Vossos, J. Zeleznikow, T. Dillon and V. Vossos, An example of integrating legal case based reasoning with object-oriented rule-based systems: IKBALS II, in: *Proceedings 3rd International Conference on AI and Law*, Oxford (1991).

[40] R. Wilensky, *Planning and Understanding* (Addison-Wesley, Reading, MA, 1983).

[41] D.C. Wilkins, Apprenticeship learning techniques for knowledge based systems, PhD thesis, Tech. Rept. STAN-CS-88-1242 or KSL-88-14, Stanford University, Stanford, CA (1988).