Chapter 5

Empirical Learning Results in *POLLYANNA*

5.1. Introduction

The *value* of empirical learning is demonstrated by results of testing the theory space search (*TSS*) component of *POLLYANNA*. Empirical data shows approximations generated from generic simplifying assumptions to have widely varying levels of accuracy and efficiency. The candidate theory space includes some theories with Pareto optimal combinations of accuracy and efficiency, as well as others that are non-optimal. Empirical learning is thus needed to separate the optimal theories from the non-optimal ones. It works as a filter on the process of generating approximations from generic simplifying assumptions. Empirical tests serve an additional purpose as well. Theory space search collects data that precisely characterizes the tradeoff between accuracy and efficiency among the candidate approximate theories. The tradeoff data can be used to select a theory that best balances the competing objectives of accuracy and efficiency in a manner appropriate to the intended performance context.

The *feasibility* of empirical learning is also addressed by results of testing the theory space search component of *POLLYANNA*. In order for empirical testing to be feasible, candidate approximate theories must be operationally usable. Candidate hearts theories generated by *POLLYANNA* are shown to be operationally usable by experimental results from the theory space search *(TSS)* phase of learning. They run on a real machine producing results that can be compared with training examples. Feasibility also depends on the information and computation costs of empirical testing. Information costs result from the need to supply the system with training examples. Computation costs result from the need to execute candidate theories. Both types of costs grow with the numbers of candidate theories to be tested. Experimental results show that empirical testing in *POLLYANNA* is limited more by the computation costs of executing candidate theories than by the information costs of obtaining many training examples. *POLLYANNA* contrasts in this respect with traditional inductive learning systems. The feasibility of empirical learning depends also on the intended performance context, and on the resources available in the context of learning. Measurements from the theory space search phase indicate that *TSS* algorithms performing exhaustive search would not be feasible for the hearts domain, although they may be feasible for other applications. *TSS* algorithms that avoid exhaustive search hold considerably more promise.

5.2. Overview

Empirical results from exhaustively testing the large hearts theory space are presented in Section 5.3. Scatterplots of accuracy versus efficiency show the appearance of Pareto optimal and non-optimal candidates. Techniques for measuring the tradeoff between accuracy and efficiency are presented. The empirical data is interpreted in terms of the semantics of candidate hearts theories. The data indicates considerable redundancy and decomposability in the large hearts theory space. Results of empirically testing theory space monotonicity are presented as well. This section serves to support the claim that empirical learning is useful in *POLLYANNA* (Subsidiary Claims #2 and #3 in Section 1.6.2.2).

Progressive learning behavior in *POLLYANNA* is analyzed in section 5.4. The appearance of knowledge level learning in *POLLYANNA* is discussed in terms of the system's state of knowledge at different points in time. Knowledge level learning is further demonstrated by generating learning curves that measure improvements in accuracy with increasing numbers of training examples. (See Claim #7 in Section 1.6.7). The contribution of analytic learning is analyzed by comparing empirical results from the large hearts theory space to results from a random collection of candidate theories. The contribution of empirical learning is analyzed by showing how empirical results vary with changes in the game playing strategy manifested in training examples.

The feasibility of empirical testing is discussed in Section 5.5. The problem is analyzed partly in terms of the information and computation resources required for exhaustive testing. These are determined largely by the number of candidate theories to be tested and by the goals of learning. The problem is also discussed in terms of the resources available for learning. These are determined largely by the context in which learning occurs. The feasibility of empirical learning in the hearts domain is discussed as well. Feasibility hinges in part on the control strategy used to generate candidate theories. The costs of exhaustive testing are estimated for one particular automatic control strategy. The estimates show exhaustive testing to be expensive but feasible. This section serves to support the claim that empirical learning is feasible in *POLLYANNA* (Subsidiary Claim #2 in Section 1.6.2.2).

5.3. Analysis of TSS Experiments

5.3.1. Types of TSS Output

POLLYANNA's theory space search (*TSS*) module generates two different types of output. Its primary output is a theory or set of theories meeting specified accuracy and efficiency goals. (See Figure 4-1.) *TSS* also generates measurements of the average accuracy and efficiency of each candidate theory that was tested empirically during the search process. Although these measurements are not formal goals of learning, they are nevertheless quite useful. Empirical measurements can be used to quantify the tradeoff between accuracy and efficiency. These measurements can be combined with contextual knowledge to select a theory that best balances the competing objectives of accuracy and efficiency.

5.3.2. A Sample TSS Run

A test run of *POLLYANNA*'s empirical *TSS* module is described in this section. Two main inputs were used in the test: One input was the large hearts theory space described in Section 4.4.12. A second input was the training example set *A* described below. The search algorithm *TSS-G* was selected for use in this test run. (See Figure 4-19.) This algorithm finds a minimal cost theory meeting a specified error rate bound as described in Figure 4-19. The error rate bound was set to zero so that the entire large hearts theory space would be searched.[48]The system required approximately *10* CPU days on a Symbolics *3600* to process the entire space of *72* candidate theories against *208* training examples.

The training example set *A* is described in Figure 5-1. This training set was created by the example generator described in Figure 4-23. The examples were represented in *Form#1*, i.e. they indicate all the choices the teacher considered to be optimal. The examples manifest the general card playing strategy outlined in Figure 4-23. The general strategy was instantiated for this training set using the optimization measures *Point-Value*, *Relative-Depth* and *Shown-Out* in the priority order shown in Figure 5-1. The strategy first optimizes the current trick score and then optimizes for future tricks as a secondary criterion. It minimizes *Point-Value(choice)* in order to optimize for the current trick. It maximizes *Point-Value(c)* to optimize for future tricks. The measures *Relative-Depth* and *Shown-Out* are used successively to break ties in the measure(s) of higher priority.

Results from this test run are presented graphically in Figure 5-2. The graph plots one point for each of the *72* candidate theories in the large hearts theory space. The horizontal axis measures the efficiency candidate theories. Efficiency was measured in terms of the average cost per training example of evaluating the theory. Costs were

---

[48]The algorithm *TSS-G* was used here as a tool for collecting experimental data. The entire large hearts space was searched in support of research objectives. Exhaustive search would might be avoided in a real application context by supplying *TSS-G* or some other algorithm with a goal that is easier to achieve.

```
Example Format:            Form#1.

Number of Examples:        (4 Games) * (52 Cards) = 208

Example Strategy:

   Current Trick Priority: 1. Minimize Point-Value
                           2. Maximize Relative-Depth
                           3. Minimize Shown-Out

   Future Trick Priority:  1. Maximize Point-Value
                           2. Minimize Relative-Depth
                           3. Maximize Shown-Out
```

Figure 5-1: Training Set *A* Specifications

computed using the abstract clock defined in Figure 4-25 with uniform weights. The costs are therefore equal to the average number of primitive operations per training example. The vertical axis indicates accuracy of each candidate theory. Accuracy was measured in terms of average error rates. Error rates were computed using the definition of E_1 in Figure 4-24, for examples represented in *Form#1*. The error rates are therefore equal to the probability that the candidate theory will select a card that is outside of the teacher's optimal card set. The scatterplot shows considerable variation in levels of accuracy and efficiency of candidate theories.

All of the Pareto optimal theories are marked with circles in Figure 5-2. In the absence of contextual knowledge, it is not possible to determine which of the Pareto optimal theories is best. A tradeoff results from any attempt to compare two Pareto optimal theories t_1 and t_2. If theory t_1 is more accurate, then theory t_2 is more efficient. Likewise, if theory t_1 is more efficient, then theory t_2 is more accurate. Each Pareto optimal theory therefore cannot be improved in efficiency except by sacrificing accuracy. Likewise each cannot be improved in accuracy except by sacrificing efficiency. Depending on the accuracy and/or efficiency bounds relevant to the application context, any one of the Pareto optimal theories may be considered "best". None of the non-Pareto-optimal theories will ever be considered "best", so long as accuracy and efficiency are the only relevant concerns. A non-Pareto-optimal theory can always be improved in either accuracy or efficiency or both by selecting a Pareto optimal one.

The value of empirical learning is clearly illustrated by the data in Figure 5-2. Empirical learning is useful because of the generality of generic simplifying assumptions. Measurements of accuracy and efficiency show that *GSAs* generate both good approximate theories, i.e., the Pareto optimal ones, as well as bad approximate theories, i.e., the non-Pareto-optimal ones. The large hearts theory space contains *7* Pareto optimal theories and *65* non-Pareto-optimal ones. Empirical learning therefore becomes useful. Empirical test can separate the Pareto optimal theories from the non Pareto optimal ones.

Theory Space = LARGE   Example Set = A

Figure 5-2: Accuracy v. Efficiency Scatterplot

5.3.3. Measuring the Accuracy / Efficiency Tradeoff

The Pareto optimal theories define a set of options available in any given performance context. These options enable accuracy to be traded in return for efficiency, and vice versa. The tradeoff can be precisely quantified using the accuracy and efficiency measurements shown in Figure 5-2. A quantitative measure of the tradeoff is shown in Figure 5-3. This graph is identical to the scatterplot of Figure 5-2 except that a solid line has been added. The solid line represents the effective tradeoff between accuracy and efficiency that results from the set of Pareto optimal theories. Among the Pareto optimal theories, error rates vary from a low of approximately *0.15* to a high of approximately *0.43*, i.e., a factor of about *2.9*. Computation costs vary from a low of approximately *134* operations per example to a high of approximately *110900* operations per example, i.e., a factor of about *830*.

`Theory Space = LARGE    Example Set = A`

Figure 5-3: The Accuracy / Efficiency Tradeoff

The accuracy/efficiency tradeoff curve can be defined conceptually in the following way: Consider each Pareto optimal theory, *t* with cost *C* and error rate *E*. Draw a vertical ray from upward from *(C,E)* to infinity and a horizontal ray rightward from *(C,E)* to infinity. These rays define a quadrant containing pairs of efficiency and accuracy levels that are dominated by the pair *(C,E)*. Taking the union of all such quadrants, for each Pareto optimal theory, one obtains the region of all sub-optimal *(C,E)* points. The tradeoff line of Figure 5-3 defines the border of this semi-infinite region of sub-optimal points.

The tradeoff curve in Figure 5-3 can be interpreted by considering the following situation: Suppose the performance context requires an approximate theory meeting a cost bound equal to *C*. The appropriate theory is determined by first finding the value *C* along the horizontal, efficiency axis. A vertical line is then drawn to intersect the tradeoff curve. The intersection occurs at the best accuracy level that can be achieved within the efficiency threshold *C*. The corresponding Pareto optimal theory is found by following the tradeoff curve leftward from the point of intersection to the first circled theory. Suppose instead that the context requires an approximate theory meeting an error rate bound of *E*. The appropriate theory is now determined by first finding the value *E* on the vertical, accuracy axis. A line drawn rightward intersects the tradeoff curve at the best efficiency level that can be achieved at accuracy level *E*. The corresponding Pareto optimal theory is found by following the tradeoff curve downward to the first circled theory. Notice in each case that the resulting Pareto optimal theory will likely achieve a level of efficiency or accuracy that is better than the specified bound.[49]

POLLYANNA's *TSS* algorithms return different results depending on the contextual knowledge provided to the system in the form of learning goals. (See Figure 4-1.) This fact is illustrated by the accuracy / efficiency tradeoff curve in Figure 5-3. For example, suppose algorithm *TSS-E*, (Figure 4-18), were given a cost bound of *C*. Depending on the value of *C*, any one of the *7* Pareto optimal theories could be returned by *TSS-E* as the most accurate theory satisfying this cost bound. The theory returned by *TSS-E* would change as the value of *C* passes through efficiency levels corresponding to corners of the tradeoff curve. Suppose instead that algorithm *TSS-G*, (Figure 4-19), were given an error rate bound *E*. Depending on the value of *E*, any one of the *7* Pareto optimal theories could be returned as the most efficient theory satisfying this accuracy bound. The theory returned by *TSS-G* would change as the value of *E* passes through accuracy levels corresponding to corners of the tradeoff curve.

The accuracy/efficiency tradeoff curve in Figure 5-3 is rather jagged. The jaggedness indicates a rather inflexible tradeoff between accuracy and efficiency. Inflexibility results from the fact that only a finite set of discrete accuracy and efficiency levels is available in the set of Pareto optimal theories. A more flexible tradeoff can be obtained by selecting combinations of Pareto optimal theories. Suppose that theories t_1 and t_2 are both adopted for use in practice. Suppose further that a weighted coin with probability p is used to decide whether to invoke t_1 or t_2 on any given problem instance, i.e., either t_1 is invoked or t_2 is invoked, but never both. The overall expected efficiency will be: $<Efficiency>=p*e_1+(1-p)*e_2$, assuming the cost of a coin toss is small enough to ignore. The expected accuracy will be: $<Accuracy>=p*a_1+(1-p)*a_2$. Average case accuracy and efficiency levels at any point along the line connecting the points (e_1,a_1) and (e_2,a_2) can be obtained with a suitable choice for p, the bias of the

[49]Algorithms *TSS-E* and *TSS-G* implicitly carry out the graphical procedures described above. In a real application context, these algorithms would be used instead of the graphical methods in order to avoid exhaustive search.

weighted coin. Combinations of accuracy and efficiency lying at any point on the convex hull of the scatterplot of Figure 5-3 can be reached by choosing an appropriate pair of theories. All points lying inside the convex hull can be reached by building composite theories from two or more individual theories; however, the points on the interior are sub-optimal. The accuracy/efficiency tradeoff curve for composite theories is described by the solid line in Figure 5-4.

Theory Space = LARGE   Example Set = A

Figure 5-4: The Accuracy / Efficiency Tradeoff: Composite Theories

An interesting comparison can be drawn between the two tradeoff curves defined in Figures 5-3 and 5-4. Notice that the composite theories can attain some combinations of accuracy and efficiency that cannot be reached using any single theory. Some of the Pareto optimal theories in Figure 5-3 are actually dominated by composite theories lying on the tradeoff line of Figure 5-4. This occurs because the dominated Pareto optimal theories do not lie on the convex hull of the scatterplot.

Composite theories can present a problem in contexts that require considering the variance of accuracy or efficiency. A composite theory can have greater variance in accuracy and/or efficiency than the individual theories from which it was constructed. For example, consider a limiting case in which each individual theory has zero variance in efficiency. In this case, the composite theory will have greater variance than each individual theory. Such a composite theory would be no help in the context of a real time system that requires an efficiency bound to be met on each individual example. When the context requires considering worst case values of accuracy and/or efficiency, composite theories provide no advantage whatsoever.

5.3.4. Domain Level Interpretation of Results

Empirical results from the test run can be interpreted in terms of the semantics of candidate hearts theories. Consider the hearts theory space shown in Figure 4-15. This small space is a *16* member subset of the large hearts theory space of *72* theories. The small space contains one theory for each combination of versions of the four key hearts functions: (1) *Exp-Win (EWN)*, the probability of winning the current trick; (2) *Exp-Trick-Value (ETV)*, the expected value of the current trick; (3) *Exp-Next-Win (ENW)*, the probability of winning the next trick; (4) *Exp-Next-Trick-Value (ENT)*, the expected value of the next trick. Each quantity can be computed using either a naive or sophisticated version, as described in Figure 3-4. The naive versions compute constant values for the corresponding quantities, while the sophisticated versions compute values that depend on the card choice under consideration. Recall that the choice evaluation function *Exp-Game-Score* is a monotonically increasing function of each of the four quantities, as described in Figure 3-2.

5.3.4.1. Empirical Results for Selected Theories

Empirical results for theories in the small hearts space are shown in Figure 5-5. This scatterplot is the same as that presented in Figure 5-2, except that points for the sixteen selected theories are labeled, and the other points are deleted.[50] Notice that all of the Pareto optimal theories are included in the small hearts space of Figure 4-15. Were this small space used in the *TSS* phase instead of the large theory space, the resulting set of Pareto optimal theories would be the same. Although this fact is clear after observing the results of empirical learning, it was not known beforehand. Notice also that the abstract space contains some theories that are not Pareto optimal. Thus empirical learning would still be useful even if it were possible to limit consideration only to those candidate theories appearing in the abstract space.

[50]The reader may wish to compare theories appearing in this empirical data to the semantic analysis of theories presented in Chapter 3. The following pairs correspond: *(T0,A), (T1,C), (T2,B), (T3,I), (T4,E), (T5,M), (T6,F), (T7,G) and (T8,P).*

Theory Space = SMALL   Example Set = A

Figure 5-5: Accuracy and Efficiency of Selected Theories

The most efficient candidate in the large hearts theory space is theory *A*. This Pareto optimal theory uses naive versions to compute all four quantities in the hearts evaluation function. Theory *A* actually returns a constant as the value of *Exp-Game-Score* for all card choices in all situations. When this theory is used, all choices are equally good. The choice set *t-goods* includes all the legal choices. The player simply makes random selections from his legal choices. The error rate E_1 for theory *A* is equal to *0.43*. This is just the percentage of choices the teacher considered to be sub-optimal, averaged over all examples.

Theories *E, C, I* and *B* use sophisticated versions of *Exp-Win*, *Exp-Trick-Value*, *Exp-Next-Win*, and

Exp-Next-Trick-Value respectively. For all other quantities, each uses the naive version. Observe the locations of these points on the scatterplot of Figure 5-5. In terms of accuracy, these four theories fall into the order: *E, C, B, I*, with *E* being the most accurate and *I* being the least accurate. These results indicate the following order of importance when each quantity is used by itself in the evaluation function: *Exp-Win*, *Exp-Trick-Value*, *Exp-Next-Trick-Value*, *Exp-Next-Win*. When starting at theory *A*, a refinement of *Exp-Win* has the greatest positive impact on accuracy. A refinement of *Exp-Next-Win* has the greatest negative impact in accuracy. Refinements of *Exp-Trick-Value* and *Exp-Next-Trick-Value* have intermediate levels of impact on overall accuracy.

Empirical results for theories *E,C,B* and *I* can be interpreted in terms of the card playing strategies that result from using the four theories. First consider theories *E* and *I*. These theories minimize the odds of winning the *current* trick, *Exp-Win*, and the odds of winning the *next* trick, *Exp-Next-Win*, respectively. Theory *E* generates evaluations that are *non-decreasing* functions of card rank. It minimizes *Exp-Win* by usually selecting cards of *minimal* rank. Although this theory sometimes considers cards of different rank to be equally good, it never prefers a card of greater rank to a card of lesser rank. This represents a strategy of playing low to avoid winning the current trick. Theory *I* is nearly opposite to theory *E*. It generates evaluations that are strictly *decreasing* functions of rank. Theory *I* thus minimizes *Exp-Next-Win* by selecting cards of maximal rank. This represents a strategy of playing high to get rid of cards that might win the next trick. Theories *E* and *I* have rather different accuracy levels. Theory *E* is somewhat better than random guessing, while theory *I* is worse than random guessing. These results suggest an interaction between the quantities *Exp-Win* and *Exp-Next-Win*. Theory *I* indirectly maximizes quantity *Exp-Win*, while attempting to minimize *Exp-Next-Win*. Theory *I* avoids winning the next trick by getting rid of high cards in the current trick. This strategy indirectly tends to increase the odds of winning the current trick.

Now consider theories *C* and *B*. These theories minimize the point value of the *current* trick, *Exp-Trick-Value*, and the point value of the *next* trick, *Exp-Next-Trick-Value*, respectively. Theory *C* minimizes current trick point value by playing cards of *minimal* point value in the current trick. Theory *B* is opposite to theory *C*. It minimizes next trick point value by playing cards of *maximal* point value in the current trick, in order to get rid of them. In contrast to theories *E* and *I*, the two theories *C* and *B* have nearly the same accuracy and efficiency levels. Each is slightly less efficient and slightly more accurate than the random guessing of theory *A*. It is perhaps surprising that these two opposite strategies should both be more accurate than random guessing; however, the improvement is small and may not be statistically significant.

Theory *P* is the most accurate theory in the large hearts theory space. It is also the least efficient theory in the small space of Figure 4-15. This theory uses sophisticated versions of all four of the quantities in the evaluation function. Theories *L, N, H* and *O* each use a naive version of quantities *Exp-Win, Exp-Trick-Value, Exp-Next-Win* and *Exp-Next-Trick-Value* respectively. For all other quantities, each uses the sophisticated version. Observe the locations of these theories in Figure 5-5. In terms of accuracy, these four theories fall into the order: *L*, *N*, *O* and *H*, with *L* being the least accurate and *H* being the most accurate. This suggests the following rank ordering of the importance of the four quantities, assuming a naive version of only one quantity will be used: *Exp-Win, Exp-Trick-Value*, *Exp-Next-Trick-Value*, *Exp-Next-Win*. Starting at theory *P*, simplification of *Exp-Win* has the greatest negative impact on accuracy. Simplification of *Exp-Next-Win* has the least negative impact on accuracy. Simplification of *Exp-Trick-Value* and *Exp-Next-Trick-Value* have intermediate levels of impact on accuracy.

Additional insight into these empirical results can be obtained by considering the strategy used in generating the training example set *A*. (See Figures 5-1 and 4-23.) The strategy always prefers to play a card that is guaranteed to avoid taking points in the current trick, if such a card is available. This is consistent with the empirical result showing that the current trick quantities, *Exp-Win* and *Exp-Trick-Value*, are more important than the future trick quantities, *Exp-Next-Win* and *Exp-Next-Trick-Value*. As described in Section 4.5.2.2, a card is guaranteed to take no current trick points if it is certain to lose the trick or if the trick is certain to have no points. The first possibility apparently occurs more often. Thus the empirical results show *Exp-Win* to be more important than *Exp-Trick-Value*. In cases where more than one card is certain to lose the current trick, the example strategy prefers to dump cards of maximal point value. Thus it is not surprising that *Exp-Next-Trick-Value* is more important than *Exp-Next-Win*.

5.3.4.2. Interactions Among Sub-Expressions

Interaction among the four basic quantities *Exp-Win*, *Exp-Trick-Value*, *Exp-Next-Win* and *Exp-Next-Trick-Value* can be analyzed with the help of Figures 5-6, 5-7, 5-8 and 5-9. These four figures correspond to the four quantities *Exp-Win, Exp-Trick-Value, Exp-Next-Win* and *Exp-Next-Trick-Value*. Each figure shows the effect of refining the naive version of the quantity into the sophisticated version. Each naive or sophisticated version can be used in a total of eight different theories. Thus each figure has eight arrows indicating the refinement impact in eight different contexts.

Several different questions can be asked about the interactions of the various different quantities. To begin with, for a given function *f*, one may ask whether a refinement of function *f* will improve accuracy in all contexts. Figure 5-6 indicates that the arrows describing refinements of *Exp-Win* all point downward, i.e., refinements of this quantity always improve accuracy. As for the quantities *Exp-Trick-Value*, Figure 5-7, and *Exp-Next-Win*, Figure 5-8, some arrows point upward and some point downward, i.e., refinements of these quantities can improve or degrade accuracy, depending on the context. In the case of quantity *Exp-Next-Trick-Value*, Figure 5-9, all the arrows point downward or are level, i.e., refinements of this quantity either improve accuracy or have no effect on accuracy.

Theory Space = SMALL   Example Set = A

Figure 5-6: Impact of Refining *Exp-Win*

A second question concerns the interaction between pairs of quantities (*Exp-Win*,*Exp-Trick-Value*) and (*Exp-Next-Win*,*Exp-Next-Trick-Value*) that are multiplied together to get the player's score on the current and next trick respectively. Figure 5-6 indicates that refinements of *Exp-Win* have greater positive impact when the sophisticated version of *Exp-Trick-Value* is used. Likewise Figure 5-7 indicates that refinements of *Exp-Trick-Value* have greater positive impact when the sophisticated version of *Exp-Win* is used. Thus refinements of *Exp-Win* and *Exp-Trick-Value* seem to enhance or amplify each other. Figures 5-8 and 5-9 show the opposite interaction between quantities *Exp-Next-Win* and *Exp-Next-Trick-Value*. Refinements of *Exp-Next-Win* have greater positive impact when the naive version of *Exp-Next-Trick-Value* is used. Likewise, refinements of *Exp-Next-Trick-Value* have

Theory Space = SMALL   Example Set = A

Figure 5-7: Impact of Refining *Exp-Trick-Value*

greater positive impact when the naive version of *Exp-Next-Win* is used. Thus refinements of *Exp-Next-Win* and *Exp-Next-Trick-Value* seem to diminish or substitute for each other.

5.3.5. Redundancy in the Theory Space

Theory space redundancy is illustrated by the data in Figure 5-2. The data suggests that many theories can be grouped into sets for which all members appear to have the same accuracy level. The underlying numeric accuracy values demonstrate that many theories lie at *exactly* equal accuracy levels. These observations are summarized by the histogram in Figure 5-10. This histogram displays the number of candidate theories that manifest each accuracy

Theory Space = SMALL   Example Set = A

Figure 5-8: Impact of Refining *Exp-Next-Win*

level appearing in the scatterplot. Although the theory space includes *72* candidate theories, only *22* distinct accuracy levels appear in the data. In some cases, as many as six theories have exactly the same accuracy level.

Efficiency levels of some theories are very nearly equal to each other, although efficiency is never replicated exactly in this data. Computational costs are only slightly increased by many refinements that replace one function version with a more sophisticated version. Consider that only *48* distinct points are observed in the scatterplot of 5-2, even though a total of *72* points are plotted. The remaining *24* theories do not show up because their efficiency and accuracy levels are both nearly or exactly equal to corresponding values of other theories. The data points are too close to be distinguishable on this graph.

Theory Space = SMALL   Example Set = A

Figure 5-9: Impact of Refining *Exp-Next-Trick-Value*

The exact replication of accuracy levels can be explained in several ways. One possibility is pure coincidence. Two theories t_1 and t_2 may attain identical accuracy levels by chance, even though they make different predictions on the training examples. The specific errors could be different even though overall accuracy is the same. Another explanation is more likely since the accuracy level replication is exact. The two theories t_1 and t_2 could have made identical predictions on all the observed training examples. Identical predictions could occur for two possible reasons. The examples on which t_1 and t_2 differ may occur only rarely. None may have appeared in the training set. It might also turn out that t_1 and t_2 are semantically equivalent.

Theory Space = LARGE   Example Set = A

Figure 5-10: Accuracy Level Histogram

Most of the accuracy replication in Figure 5-2 appears to result from semantic equivalence of theories. This observation results from examining the definitions of some candidate theories. Consider the case of theories *TH867* and *TH881* which both have an error rate E_1 equal to *0.42*. These two theories differ only in the definitions of the generic function *Exp-Point-Value(c,p,...,p')*. This function computes the expected point value of the card played by player *p*' in the current trick, assuming the current player *p* plays card *c* now. Under theory *TH867*, the function *Exp-Point-Value* is computed by an equiprobable average over the range of the random variable *Point-Value*. Under theory *TH881*, the function *Exp-Point-Value* is computed by assuming each player is equally likely to play any card in the deck. Although different numeric values are returned by these two versions of *Exp-Point-Value* the difference does not turn out to be significant. Each version ignore the actual point value of the card *c* played by the current player. Each uses *Exp-Point-Value* in an evaluation function with the form *Eval(c)=K*Exp-Point-Value(c)+F(c)*. The rank order of card evaluations is therefore always the same under the two theories *TH867* and *TH881*, even though the actual numeric evaluations are different. These two theories therefore lead to the came card playing behavior in all situations. Upon examining the definitions of other sets of theories, it becomes clear that this phenomenon occurs quite often in the large hearts theory space. Many refinements merely change the scale of card evaluations while leaving the rank ordering unchanged.

5.3.6. Decomposability of the Theory Space

Interesting relationships between entire groups of points can be seen in Figure 5-2. Several groups of points appear to correspond systematically to each other, i.e., for groups g_1 and g_2, each point in g_1 differs from a corresponding point of g_2 by a fixed displacement in both accuracy and efficiency. For example, several triplets of points are found in Figure 5-2. All three theories in each triplet have the same relative efficiency levels and exactly the same accuracy levels. Upon examining the underlying theory definitions, it becomes clear that the pairs of corresponding theories all differ from each other in the same way. The same refinement $f_i \rightarrow f_j$ moves from an arbitrary member of one group to the corresponding member of the other group.

The hearts theory space may be partly decomposable if these systematic relationships are not accidental. Decomposability would be quite useful in facilitating the search process. Suppose *TSS* makes a refinement $t_a \rightarrow t_b$ corresponding to the substitution of versions $f_i \rightarrow f_j$. Suppose further that *TSS* records the incremental effect of this refinement on the levels of accuracy and efficiency. When exploring another part of theory space, *TSS* may make another refinement $t_c \rightarrow t_d$ corresponding to the same version substitution $f_i \rightarrow f_j$ once again. If the refinement is guaranteed to have the same incremental impact as it did before, then empirical evaluation of the resulting theory could be avoided. Exploitation of this decomposability property would be facilitated by developing analytic techniques to predict whether a refinement will have the same impact in new parts of the theory space. In the absence of such analytic techniques, a search control mechanism might assume decomposability to estimate the impact of refinements prior to actual empirical testing. A similar control technique was used in Keller's *MetaLEX* system [Keller 87].

5.3.7. Univariate Analysis of Accuracy and Efficiency

5.3.7.1. Comparison of Accuracy Measures

Empirical data was collected to test the relationship between the two types of training examples described in Figure 4-21. As defined in Figure 4-24, the accuracy measure E_1 is used when the examples specify all optimal choices, and the measure E_2 is used when the examples specify only one optimal choice. The correlation between measures E_1 and E_2 is shown in Figure 5-11. This diagram plots E_1 on the horizontal axis and E_2 on the vertical axis for each of the *72* theories in the large hearts theory space. The error rates were collected in the course of processing the training set *D*, a set of *52* examples that manifest the same strategy as used in the set *A*, described in Figure 5-1. (The example sets *A* and *D* were generated using different initial card deals.) The scatterplot indicates a strong positive correlation between the two accuracy measures. Nevertheless the relationship is not monotonic, i.e., the rank order of theory accuracy levels will change when one measure is substituted for the other. The Pareto optimal set may change as well. The average values of these two measures are proportional given a sufficient number of examples. (See Section 4.5.2.4). The present results show the proportionality to occur only for training sets larger than the ones used in testing *POLLYANNA*.

5.3.7.2. Comparison of Efficiency Measures

Empirical data was collected to test the importance of the weights used to assign costs to each type of primitive operation recorded in the abstract clock of Figure 4-25. It would be desirable to show that the results are entirely independent of the clock weights. This could be demonstrated by generating scatterplots that show the correlation between the different components of the abstract clock. Suppose the clock components are monotonically increasing functions of each other. This would imply that the rank order of theory efficiency levels would be invariant with respect to changes in the clock weights. Thus the Pareto optimal set would be invariant as well.

Sensitivity to timing weights is illustrated by the data in Figures 5-12 and 5-13. The horizontal axis of each graph measures the clock component *CALLS*, i.e., the total number of function calls. The vertical axis in Figure 5-12 measures the clock component *ARITHMETICS*, i.e., the number of arithmetic operations. The vertical axis in Figure 5-13 measures the clock component *KNOWNS*, i.e., the number of calls to the primitive function that tests whether the value of a random variable is known. These efficiency measurements were collected in the course of processing the training set *D* described above. Both plots show positive correlation between the different components of the clock; however, the correlation between *CALLS* and *KNOWNS* is a lot weaker than the correlation between *CALLS* and *ARITHMETICS*. Notice that *neither* plot shows a strictly monotonic relationship.

Theory Space = LARGE   Training Set = D

Figure 5-11: Correlation of Accuracy Measures

Therefore the Pareto optimal set will *not* be invariant with respect to all changes in clock weights. If the candidate theories are moved from one computing environment to another, the relative costs of each operation may change. The set of Pareto optimal candidates may change as well.

`Theory Space = LARGE    Training Set = D`

Figure 5-12: Correlation of Efficiency Measures

5.3.7.3. Empirical Tests of Theory Space Monotonicity

Empirical data was collected to test whether the large hearts theory space is monotonic in the efficiency of candidate theories. The test program simply checks each refinement link *Refinement*(t_1,t_2) that connects two theories t_1 and t_2. It determines whether the average cost of t_1 is no greater than the average cost of t_2, taken over a given training set. Monotonicity tests were performed on a variety of training sets. The results were the same in all cases: Proper monotonicity was observed on nearly all refinement links in the large hearts theory space, i.e., *162* out of *186* links. A small number of links violated the monotonicity constraint, i.e. *24* out of *186* links.

Theory Space = LARGE   Training Set = D

Figure 5-13: Correlation of Efficiency Measures

All the violations involved a single *Primitive-Refinement* pair, i.e., the one replacing version *PLC-0* with version *PLC-1* of the function *Prob-Lead-Card(c,p,tr,private,public,c')*. (See Appendix J.) This function computes the odds that the lead card is *c'* in the state that results from player *p* playing card *c* in the current state. The naive version *PLC-0* assumes that all cards are equally likely to be the lead card. The sophisticated version *PLC-1* simply looks up the leader's card in the problem state description. The function *Prob-Lead-Card* is called indirectly within an expression of the form: ∑ *(s in Suits) p(s)*q(s)*, where *p(s)* computes the odds that *s* is the lead suit. *POLLYANNA* uses a multiplication function that avoids computing the second argument whenever the first argument is zero. When a theory uses the naive version *PLC-0*, all probabilities *p(s)* are a non-zero constant.

Whenever a theory uses the sophisticated version *PLC-1*, the probability *p(s)* is zero for three out of four suits. The sophisticated version therefore leads to fewer evaluations of *q(x)* and a more efficient theory than the naive version.

The observed non-monotonicity can blamed on specific violations of the auxiliary assumptions used in proving the assumption impact theorems for *Function Invariance (FI)* and *Argument Abstraction (AA)*. (See Figures 4-10 and 4-11.) One violation involves the assumption that the evaluation trees of theories t_1 and t_2 are isomorphic, whenever t_1 and t_2 differ by an application of *FI* or *AA*. The clever multiplication function violates this constraint by conditionally evaluating its second argument. A second violation involves the assumption that all functions have constant costs, regardless of their arguments. This violation is really a consequence of the first violation. Notice that the large hearts theory space might well be entirely monotonic if a normal version of multiplication had were used.

Theory spaces generated by *POLLYANNA* are not intended to be monotonic in accuracy. Empirical tests show that no such monotonicity occurs in the hearts theory space. As efficiency falls along paths in the theory space, accuracy may rise or fall or remain constant. Non-monotonic changes in accuracy are illustrated graphically by Figures 5-6, 5-7, 5-8 and 5-9. These graphs show refinement links in the small hearts theory space. Most of the arrows point downward and to the right. These indicate cases in which sacrifices of efficiency did lead to improvements in accuracy. A few arrows point upward and to the right. The indicate cases in which sacrifices of efficiency actually made the accuracy worse. (Notice that no arrows point to the left. The small hearts space is entirely monotonic in efficiency.)

5.4. Measures of Learning in *POLLYANNA*

POLLYANNA can be viewed as a learning system in several different senses. Two different approaches will be used to characterize the learning that occurs in this system. A qualitative approach involves examining the system's knowledge base at several distinct points of the learning process. A more quantitative measure of learning results from examining the incremental effect of each additional training example processed by the system. Empirical evidence will be presented to support this quantitative measure of learning.

5.4.1. Knowledge Level Learning in *POLLYANNA*

A qualitative view of learning is based on considering *POLLYANNA*'s state of knowledge at three points in time: (S_0) Prior to operation: *POLLYANNA*'s knowledge consists of the initial intractable theory. (S_1) After analytic methods have generated a candidate theory space: *POLLYANNA*'s knowledge may be said to include only the areas of agreement among the candidate theories. (S_2) After empirical methods have tested the candidate theories: *POLLYANNA*'s knowledge consists of a single goal theory. (See Figure 5-14.) In moving from state S_0 to state S_1, the knowledge base becomes less correct, since a correct theory is replaced with approximate ones. The knowledge base in state S_1 is also be considered less complete with respect to areas of disagreement among candidate theories. In moving from state S_1 to state S_2, the knowledge base becomes more correct as the error prone theories fail empirical testing. The knowledge base also becomes more complete in state S_2, since a large set of candidate theories is replaced with a single goal theory. Comparing the initial state S_0 and the final state S_2, the knowledge becomes less correct and remains equally complete. The state transitions are characterized in Figure 5-14, using the dimensions of correctness (accuracy), completeness (scope/precision), and tractability (efficiency).

STATE:	KNOWLEDGE BASE:
S_0	Initial Theory
S_1	Theory Space
S_2	Final Theory

Transition:	Accuracy:	Scope/Precision:	Efficiency:	KLL:	TKLL:
$S_0 \rightarrow S_1$	Down	Down	Up	Yes	-
$S_1 \rightarrow S_2$	Up	Up	Up	Yes	+
$S_0 \rightarrow S_2$	Down	Same	Up	Yes	-

Figure 5-14: Comparing States of Knowledge in *POLLYANNA*

An analysis of knowledge-level learning in *POLLYANNA* is also shown in Figure 5-14. Two definitions of knowledge level learning are considered. The standard one *(KLL)* says simply that knowledge level learning occurs whenever there is a change in the deductive closure of a system’s knowledge base [Dietterich 86]. Standard knowledge level learning occurs across all three state transitions. An alternative definition *(TKLL)* accounts for the difference between changes that improve accuracy and those that degrade accuracy [Subramanian and Smith 88]. Truth-theoretic knowledge level learning occurs only if the deductive closure changes, and the knowledge base moves closer to a true model of the world.[51] The knowledge base moves away from a true model across transitions $S_0 \rightarrow S_1$ and $S_0 \rightarrow S_2$. It presumably moves toward a true model across transition $S_1 \rightarrow S_2$, assuming that empirical testing really does lead to greater accuracy.

POLLYANNA acquires meta-level knowledge during the empirical phase of learning. Both *Cost(t)* and *Error-Rate(t)* may be considered properties of candidate theories. Prior to empirical testing, the costs and error rates of candidate theories are unknown. After empirical testing, the system has obtained an estimate of average cost and

[51]The definition of *TKLL* in [Subramanian and Smith 88] requires that (a) no true facts be lost and (b) no false facts be added to the knowledge base. In the present analysis this definition is relaxed to use an average measure of accuracy.

error rates. The system has also determined a set of Pareto optimal theories, and has generated a curve indicating the quantitative tradeoff between accuracy and efficiency. These may all be considered types of meta-knowledge that are acquired during the empirical phase of learning.

5.4.2. Learning Curves for *POLLYANNA*

A more quantitative measure of learning results from an incremental view of the empirical phase. If *POLLYANNA* really learns something by processing training examples, the results ought to improve with the size of the training set. Such improvement can be demonstrated with *learning curves* [Langley 88]. Given some arbitrary measure of performance, a learning curve shows how the measured performance depends on the number of examples. In most empirical learning systems, performance is measured in terms of accuracy. In most analytic learning systems, performance is measured in terms of efficiency. The situation is more complicated in the context of *POLLYANNA*, since performance in measured in terms of both accuracy and efficiency.

Learning curves appropriate to *POLLYANNA* can be defined in the following way: Suppose the context requires a theory meeting some efficiency bound *C*. Suppose further that algorithm *TSS-E* is used on a training set *S-Train* to find a most accurate theory meeting the efficiency bound *C*. (See Figure 4-18.) The resulting theory *t* could be evaluated by testing its accuracy against a test set *S-Test*. A learning curve shows how the accuracy of theory *t* on the test set *S-Test* depends on the size of the training set *S-Train*. The curve will normally depend on the cost bound *C*. An entire family of curves will result from using various different cost bounds. A procedure for generating such learning curves is shown in Figure 5-15. The procedure takes an cost bound *C*, a training set *S-Train* and a test set *S-Test* as input. It generates a function called *Error-Rate(N;C)*. This function represents the accuracy on *S-Test* of the theory *t* that would result from processing the first *N* examples *S-Train* in the algorithm *TSS-E* with an efficiency bound of *C*.[52]

A selection of learning curves for *POLLYANNA* is found in Figures 5-16, 5-17, 5-18, 5-19 and 5-20. The curves were generated using the example set *A*, described in Figure 5-1, as the testing set. The training set *E* consisted of *52* examples generated using the same strategy used to generate the testing set *A*. (The example sets *A* and *E* were generated using different initial card deals.) The learning curves are ordered according the cost parameter *C*, with Figure 5-16 representing the lowest cost bound and Figure 5-20 representing the highest. These specific efficiency values were selected in a two step process. A larger set of curves was generated at efficiency

---

[52]A natural dual set of curves would result from: (a) Choosing an error rate bound *E*, (b) Using *TSS-G* to find the most efficient theory *Theory(N;E)* with error rate *E* or better, as measured against the first *N* examples in *S-Train*, (See Figure 4-19.) (c) Measuring the cost of *Theory(N;E)* against the test set *S-Test* and (d) Observing how *Cost(N;E)* depends on the number of examples *N*.

```
Given: A Cost Bound C.
       A Set of Training Examples S-Train
       A Set of Test Examples S-Test

Find:  A learning curve showing how accuracy depends on the
       number of training examples.

For N = 1 to |S-Train| do

   a. Let Theory(N;C) be the most accurate theory with cost C or less,
      measured against the first N examples in S-Train.

   b. Let Error-Rate(N;C) be the error rate E1 of Theory(N;C),
      measured against S-Test.
```

Figure 5-15: Procedure for Generating Learning Curves

levels scattered across the total range of values. These five curves were then selected by the author as examples of qualitatively different curves.

The learning curves show that performance generally improves with increases in the size of the training set. Most of the curves show a downward trend, i.e., the error rates tend to fall as the number of examples goes up. Accuracy is thus a generally increasing function of the training set size. Notice that the curves do not show monotonic improvement. Chance events in the training data can cause a theory of lower accuracy to be adopted in favor of a theory of higher accuracy. This occurs because the learning system measures accuracy against the training set, while the learning curves measure accuracy against the test set. The curves can rise or fall before settling down.

The learning curves also indicate an interesting relationship between the rate of learning and the cost bound parameter. Learning appears to occur most quickly for high cost bounds and less quickly for lower bounds. Thus the learning curve in Figure 5-20 settles down quickly. The curves in Figures 5-19, 5-18 and 5-17 take longer to settle down. At the extreme low end of the cost scale, the curve in Figure 5-16 shows no improvement at all. Any improvement would come only after a larger number of examples. These observations suggest the following explanation: At low levels of cost, the candidate theories are more accurate than random guessing only for rarely appearing examples. Candidates tend to be dominated by a randomly guessing theory until the training set becomes quite large. At higher levels of cost, candidate theories are more accurate than random guessing on most examples. Candidates distinguish themselves from random guessing after a small number of examples. This causes learning to be faster.

Theory Space = LARGE   Training Set = E   Test Set = A
Cost Bound = 51850 Operations/Example

Figure 5-16: Learning Curve for a Fixed Cost Bound

5.4.3. The Contribution of Analytic Learning

The preceding analyses have demonstrated learning behavior in *POLLYANNA*. A question naturally arises regarding which components of the system deserve credit for the improvement demonstrated by the learning curves. Successful learning in *POLLYANNA* might be attributed to either the analytic component, the empirical component, or both. The issue can be addressed by undertaking a sensitivity analysis. The analytic component serves to generate the space of candidate theories. The analytic contribution can therefore be analyzed by determining how *POLLYANNA*'s results depend on the candidate theory space. The empirical component serves to test candidates

Theory Space = LARGE   Training Set = E   Test Set = A
Cost Bound = 67899 Operations/Example

Figure 5-17: Learning Curve for a Fixed Cost Bound

against training examples. The empirical contribution can therefore be analyzed by determining how *POLLYANNA*'s results depend on the training examples. The analytic contribution will be analyzed in this section. The empirical contribution will be analyzed in the next section.

Imagine placing a band of monkeys in front of a row of structure editors. Each editor is designed so that all keystroke sequences lead to well formed hearts evaluation functions. After the monkeys are turned loose, they would pound away and eventually output a collection of candidate hearts theories. The candidate theories would then be processed against training examples in the empirical component of *POLLYANNA*. A scatterplot would

Theory Space = LARGE   Training Set = E   Test Set = A
Cost Bound = 87480 Operations/Example

Figure 5-18: Learning Curve for a Fixed Cost Bound

likely show that measurements of accuracy and efficiency vary over the candidates. A number ($N \geq 1$) of Pareto optimal theories would be found. Provided that more than one Pareto theory is found, the data would exhibit a tradeoff between accuracy and efficiency. The tradeoff curve would have a downward trend, which is simply a consequence of manner in which such curves are constructed. The monkeys would therefore likely generate data that is qualitatively similar to the scatterplots and tradeoff curves in Figures 5-3 and 5-4.

The contribution of analytic learning is tested by an experiment comparing *POLLYANNA*'s results to those that would follow from using a *random theory space*. A "random theory space" is defined in relation to an "unbiased

Theory Space = LARGE   Training Set = E   Test Set = A
Cost Bound = 105720 Operations/Example

Figure 5-19: Learning Curve for a Fixed Cost Bound

theory space". (See Section 1.3.2.2). An unbiased hearts theory space is defined in the following way: First notice that each candidate theory defines a function *Choice:[Situations→Actions]*, where *Situations* is the set of all possible game states and *Actions* is the set of all pairs: *(Goods,Bads)*. The sets *goods* and *bads* meet two constraints: (1) The two sets *goods* and *bads* are a partition of the card choices that are legal in *situation*; (2) *goods* contains at least one member. An "unbiased" theory space would include all valid functions of this form, i.e., all members of the set *[Situations→Actions]* meeting the specified restrictions. A random theory space is then defined to be randomly selected subset of the unbiased space.

Theory Space = LARGE   Training Set = E   Test Set = A
Cost Bound = 120000 Operations/Example

Figure 5-20: Learning Curve for a Fixed Cost Bound

A fair comparison to the large hearts theory space requires a random theory space of the same size. Since the hearts space contains *72* candidate theories, the random space should also contain *72* theories. Rather than generating the random theories in advance, they can be generated on the fly. Instead of using a random process to construct a whole function mapping all *Situations* to *Actions*, it suffices to randomly generate the sets *Goods* and *Bads* for only the situations actually encountered in the examples. Strictly speaking it would be necessary to record each pair *(Situation,Action)* in case the same situation is encountered again; however, such a record is not necessary since situations are never duplicated in practice. For this reason, a random space of size *72* is effectively equivalent to a space containing *72* copies of a single random selection theory. When faced with a given game *Situation*, the random selection theory generates a random partition of the legal choices into *Goods* and *Bads*, subject to the constraint that *Goods* not be empty.

Results of the comparison are shown in Figure 5-21. This graph shows the same accuracy and efficiency data for the large hearts space that was presented in above in Figure 5-2. It also shows results from processing *72* random theories against the same set of training examples. In the absence of any reasonable method of measuring the efficiency of the random theories, only accuracy values of the random theories were measured. The random theory results are represented by the two horizontal lines. These indicate the highest and lowest error rates obtained among the *72* random theories. The error rates for all *72* random theories lie at or between these levels. Notice that the random theories lie in an interval centered about the value *0.42*, nearly the same error rate as the simplest theory in the hearts space, i.e., a theory that considers all choices to be optimal.

The data in Figure 5-21 exhibits several interesting features. Notice that the hearts theories show much more variation in accuracy than the random theories. The random theories lie at error rates E_1 ranging from a high of *0.46* to a low *0.39*, an interval of length *0.06*. In contrast to this, the hearts theories lie at error rates E_1 ranging from a high of *0.45* to a low of *0.14*. They span an interval an interval of length *0.31*, about five times larger than the accuracy range of random theories. The extra variability of hearts theories lies mostly in the direction of greater accuracy. All the hearts theories are more accurate than the worst random theory and many hearts theories are more accurate than the best random theory.

Learning curves might also be generated for the random theory space. The curves could then be compared to those that result from using the large hearts theory space. Such a test is not actually necessary. The results can be predicted in advance. Unbiased learning algorithms can do no better than learning by rote [Mitchell 80]. When faced with a test example, an unbiased learner can answer correctly if the same example appears in the test set. Otherwise it must make a random guess on the test example. Significant learning occurs only when the number of training examples is so large that test examples are likely to have appeared in the training set. For training sets of the sizes used in testing *POLLYANNA*, such duplication is extremely improbable. Were learning curves to be generated for the random theory space, the random theories could do no better than guess blindly on the test set. The resulting accuracy would therefore be independent of the number of training examples. The learning curves would be flat, except for random fluctuations.

The preceding results can be interpreted in terms of the quality of *POLLYANNA*'s inductive bias. The candidate theory space was generated by applying analytic methods to the initial intractable hearts theory. The space

Theory Space = LARGE   Example Set = A

Figure 5-21: Comparing the Hearts Theory Space to Random Theories

is considered "biased" since it contains only a minute fraction of the unbiased theory space defined above. The hearts theory space achieved superior results when compared to a random theory space with an equally strong bias, i.e., with the same number of candidates. The results thus show *POLLYANNA*'s analytic methods to generate a theory space that is more effectively biased than a random theory space.

5.4.4. The Contribution of Empirical Learning

The importance of training examples has already been demonstrated in part by the scatterplot in Figure 5-2. This graph indicates the presence of both Pareto optimal and non-optimal theories in the candidate theory space. As observed above, the training examples are required to separate the optimal theories from the non-optimal ones. A simple argument suggests that the Pareto optimal theories must inevitably depend on the training examples, to some extent. Suppose two theories t_1 and t_2 have identical efficiency but have different accuracy, as measured against some training set. Suppose also that t_1 is Pareto optimal, whereas t_2 is not. Thus t_1 is more accurate that t_2. Consider what happens if the training examples are inverted so that all previously correct predictions become errors and previous errors become correct predictions. Theory t_2 will then have greater accuracy than theory t_1. This means that t_1 is not contained in the Pareto optimal set resulting from the modified training examples.

The contribution of empirical learning is further tested by an experiment determining how *POLLYANNA*'s empirical results are sensitive to the training examples. Two training sets were prepared to test the issue of *POLLYANNA*'s sensitivity to examples. The training set *B* (Figure 5-22) was generated using a different initial deal and a slightly different game playing strategy from the one used for training set *A* (Figure 5-1). The new set *B* considers the measure *Relative-Depth* (Figure 4-23) to be the best predictor of the expected points to be taken in the current or future tricks. In contrast to this, the set *A* considers the measure *Point-Value* to be a better predictor of expected points to be taken. Thus training set *B* pays more attention to the rank of cards, whereas the set *A* pays more attention to point value. The training set *C* (Figure 5-23) was generated using yet another initial deal, and a strategy aimed at losing the game of hearts. This strategy is opposite to the one used for set *B*. When training set *B* would minimize rank or point value, the losing set *C* maximizes these measures, and vice versa.

```
Example Format:            Form #1.

Number of Examples:        (4 Games) * (52 Cards) = 208

Example Strategy:

   Current Trick Priority: 1. Maximize Relative-Depth
                           2. Minimize Point-Value
                           3. Minimize Shown-Out

   Future Trick Priority:  1. Minimize Relative-Depth
                           2. Maximize Point-Value
                           3. Maximize Shown-Out
```

Figure 5-22: Training Set *B* Specifications

A comparison of the empirical results obtained from training sets *A*, *B*, and *C* is found in Figure 5-24. The leftmost column of the table describes various different efficiency ranges. The three remaining columns list the

```
Example Format:           Form #1.

Number of Examples:       (4 Games) * (52 Cards) = 208

Example Strategy:

   Current Trick Priority: 1. Minimize Relative-Depth
                           2. Maximize Point-Value
                           3. Maximize Shown-Out

   Future Trick Priority:  1. Maximize Relative-Depth
                           2. Minimize Point-Value
                           3. Minimize Shown-Out
```

Figure 5-23: Training Set *C* Specifications

Pareto optimal theories obtained in processing each training set. In order to interpret this table, imagine that the context requires a theory satisfying a cost bound *C*. Next find the efficiency interval containing the bound *C*. Moving across the corresponding row, the table indicates the most accurate theory satisfying the cost bound, as measured against each example set, i.e., the theory returned by algorithm *TSS-E*. (See Figure 4-18.) The efficiency intervals for which *B* and *A* differ are marked with a single or double * symbol. The intervals for which *B* and *C* differ are marked with a single or double # symbol.

Results from different training sets can disagree in two main ways. One type of difference involves variation in accuracy measurements. Sometimes two training sets lead to different average efficiency values for a given theory, without differing about the Pareto optimality or non-optimality of the theory. For a given Pareto optimal theory, the results disagree only about the specific cost bound at which the theory can be adopted. Differences of this sort are marked with a single * or # symbol. A more substantial type of difference involves the Pareto optimality of theories. For some efficiency intervals, one training set leads to a theory that was not considered Pareto optimal using the other training set. Differences of this sort are marked with a double * or # symbol.

Results shown in Figure 5-24 for the training set *A* list two Pareto optimal theories, *E* and *B*, (Figure 4-15), which were not considered Pareto optimal under training set *B*. These two theories focus on the single quantities *Exp-Trick-Value* (the expected current trick value) and *Exp-Next-Trick-Value* (the expected next trick value) respectively. Considering that training set *A* focuses more on point value than *B*, this result is not surprising. Results for the training set *B* show three Pareto optimal theories *M*, *I*, and *K*, which were not considered Pareto optimal under training set *A*. Theories *M* and *I* ignore the single quantities *Exp-Trick-Value* and *Exp-Next-Trick-Value* respectively. This result is also not surprising considering the lesser emphasis that *B* places on point value. The Pareto optimality of theory *K* under training set *B* cannot be so explained. It may be only a statistical anomaly.

The data in Figure 5-24 demonstrates the sensitivity of *POLLYANNA*'s results to the input training examples.

Efficiency Interval:		Training Sets: A	B	C	
0 - 128		None	None	None	
128 - 131	*	None	A	None	#
131 - 134	*	None	A	A	
134 - 7410		A	A	A	
7410 - 7940		A	A	B	##
7940 - 8970	**	B	A	B	##
8970 - 5810	**	C	A	B	##
5810 - 6010	*	E	A	B	##
6010 - 6230		E	E	B	##
6230 - 6540	*	G	E	B	##
6540 - 6700		G	G	B	##
6700 - 7000	**	G	F	B	##
7000 - 7230	**	H	F	B	##
7230 - 8570		H	H	B	##
8570 - 9290		H	H	TH858	##
9290 - 1030		H	H	TH866	##
1030 - 1080	**	H	M	TH866	##
1080 - 1110	**	H	O	TH866	##
1110 - 1120	**	P	O	TH866	##
1120 - Infinity		P	P	TH866	##

* An interval where B differs from A.
** An interval where B uses a theory not in the Pareto set of A, or vice versa.

\# An interval where B differs from C.
\## An interval where B uses a theory not in the Pareto set of C, or vice versa.

Figure 5-24: Variation of Results with Training Sets

For small changes in the examples, *B* and *A*, the results are not greatly different. There is considerable overlap in the two sets of Pareto optimal theories. The few differences that do appear can be understood in terms of the strategies used to generate the examples. For large changes in the examples, *B* and *C*, the results differ greatly. There is very little overlap in the Pareto optimal sets.

5.5. The Feasibility of Empirical Learning

The feasibility of empirical learning is determined by a variety of factors. Most of the factors are related to resource constraints. In different situations, varying amounts of information and computation resources will be required to carry out *POLLYANNA*'s theory space search (*TSS*) algorithms. *Resource requirements* depend largely on the number of candidate theories to be tested, which in turn depends on the control strategy used in the analytic generation phase of learning. These requirements are also influenced by the learning goals given to the *TSS* module. The *Resources available* for learning will depend on factors related to the context in which learning occurs. All of these factors will be examined in the following sections. The feasibility of exhaustive empirical testing in hearts will be examined as well.

5.5.1. Information and Computation Costs

Empirical testing requires both information and computation resources. Information in the form of training examples is required in order to distinguish between good (Pareto optimal) and bad candidate theories. The required number of examples will normally grow with the number of different candidate theories. For example, a trivial lower bound of *O(log(N))* indicates the number of examples required to distinguish accuracy levels of *N different* candidate theories.[53] Computation time and space are required to execute a candidate theory in order to compare predictions against training examples. If each theory must be executed separately against each training example, the total computation time is proportional to the product of the number of theories and the number of training examples. Information and computation requirements thus both grow with the number of different theories in the candidate theory space.

5.5.2. The Impact of Redundancy

Theory space redundancy appears to have a significant impact on the types of costs incurred during the empirical learning phase of *POLLYANNA*. Consider what happens when a series of redundant theories ($t_1,t_2,...,t_n,...$) is added to an initial theory space. The redundant theories add to the computation costs of empirical testing, since each new theory must be tested against the same training examples as those in the original space. Redundant theories apparently do not add to the information costs of empirical testing.[54] A great deal of redundancy occurs in theory spaces generated by *POLLYANNA*. The information expense of training examples is therefore likely to be less limiting than the computation time required to execute candidate theories. This statement appears consistent with the hearts domain experimental results. A small number of examples (*52*) and a long time (*2.5* CPU days) were required to process the training sets that led to convergence in the learning curves in Figures 5-16, 5-17, 5-18, 5-19 and 5-20.

---

[53]The lower bound follows from representing training data as positive or negative examples in the form *Optimal(choice,situation)* or *Sub-Optimal(choice,situation)*. On any set of $M \leq log(N)$ examples, some pair of theories must make identical predictions.

[54]The statement depends on the following conjecture. Consider a set *S* containing *N* classes of semantically equivalent theories. The number of examples required to partition *S* into *N* equivalence classes, with confidence level *C*, should depend only on the number *N* of classes, not on the total number of theories in those classes. (A proof of this conjecture awaits a more rigorous statistical analysis.)

5.5.3. Control Strategies and the Strength of Inductive Bias

Control strategies are needed in order to automatically generate candidate theory spaces. Weak control strategies lead to large and weakly biased theory spaces. Highly selective control leads to small and strongly biased spaces. Three control strategies were previously analyzed to determine how the number of candidates grows with depth in the random variable tree, or with the number of state variable references. (See Section 4.4.13.) These results demonstrated that spaces of enormously different sizes could result from various control strategies. *POLLYANNA* actually generates a whole family of candidate theory spaces. For any given cost bound *K*, the system generates the space of theories *t* such that $Cost(t) \leq K$. These are just the candidates that would be examined by the best efficiency first algorithm *TSS-E*, when given the cost bound *K* as a parameter. (See Figure 4-18.) Each cost bound thus defines a different theory space. As the cost bound rises, the size of the space rises as well.

5.5.4. The Performance Context

The feasibility of empirical learning is influenced by factors pertaining to context. The performance context impacts feasibility through the accuracy and efficiency goals. (See Figure 4-1.) Consider goal type *E*, which asks for a minimal error rate theory meeting a cost bound *K*. As the value of *K* goes up, *TSS-E* must test more and more theories and the computation time will rise as well. A weak efficiency bound leads to a more difficult optimization problem. Consider goal type *G*, which asks for a minimal cost theory meeting a given error rate *K*. As the value of *K* goes up, algorithm *TSS-K* is likely to find a goal theory sooner. A weak accuracy bound leads to an easier optimization problem. Similar results apply to algorithm *TSS-C* for goal type *C*, which specifies error and/or cost thresholds, but not optimization criteria. Weak efficiency bounds lead to expensive search. Weak accuracy bounds facilitate search.

5.5.5. The Learning Context

Resources available for learning depend on the context in which learning occurs. Interesting cases can be distinguished by considering the ratio of resources available for learning to the resources available in the performance context. Consider the case of domain theories that are only *relatively* intractable, so that the initial intractable theory can be executed during the learning phase. (E.g., the air traffic control example presented in Section 1.5.1.3). The computational expense of *TSS* will usually not be a problem in such a context. The system is likely to test only those candidates meeting a cost bound that is small compared to the resources available for learning. The practical information cost of learning is also low in this case, since the initial theory itself can be used to generate examples. A different situation obtains when a domain theory is *absolutely* intractable, so the initial theory cannot be executed in the learning context. Information costs may be significant since training examples must be supplied by humans. The computation cost of testing each theory may be large compared to the computational resources available for learning.

5.5.6. The Feasibility of Empirical Testing in Hearts

The feasibility of empirical testing in hearts can be estimated under several different scenarios. Several different search algorithms might be used: *TSS-C, TSS-E, TSS-G* or *TSS-K*. (See Section 4.5.1.) These algorithms use different search control methods and therefore examine different sets of theories appearing in the candidate theory space. Theory spaces might be generated using any of the approximation generator control strategies: *GENERATE, GENERATE-A* and *GENERATE-B,* proposed in Section 2.7. The sizes of theory spaces resulting these strategies were analyzed in Section 4.4.13. The numbers of candidate theories depend on the depth d to which the random variable tree is expanded. In order to generate all the same approximations as resulted from human control, these procedures must expand the hearts tree to a depth of at least *7* levels. Under these conditions, *GENERATE* leads to a space of size $S \approx 10^{81}$, *GENERATE-A* leads to a space of size $S \approx 10^{22}$. *GENERATE-B* would result in a space of size $S = 52{,}521{,}875$, about one million times larger than the hearts theory space generated with human control. Procedure *GENERATE-B* would produce candidate theories that are similar, but not identical to the ones that resulted from human control. (See Section 3.4.5.)

Empirical testing would be extremely difficult, if not impossible using any of the algorithms *TSS-E, TSS-G* and *TSS-K*. These algorithms all carry out forms of exhaustive search. Each examines all candidate theories with costs less than some specified bound. Consider what is required in order that these algorithms locate the most sophisticated theory in the human generated space. (Theory *P* in Figure 5-5.) This candidate has cost nearly equal to or greater than any other theory in the spaces that would be produced by the automatic control strategies *GENERATE GENERATE-A*, and *GENERATE-B*. Algorithms *TSS-E, TSS-G* and *TSS-K* would therefore probably examine nearly all candidate theories before locating the most sophisticated hearts theory. Such an exhaustive search is out of the question for spaces resulting from *GENERATE* and *GENERATE-A*. It would also not be feasible using the current implementation for the state abstraction space resulting from *GENERATE-B*. The running time would increase from about *2.5* CPU days to at least *2.5* million CPU days.

Exhaustive testing of the hearts state abstraction space might become feasible if *POLLYANNA* were enhanced in certain ways. Three envisioned enhancements would have a significant impact on this issue. The candidates hearts theories could be reformulated into semantically equivalent forms that are drastically more efficient. (See Section 3.6.) The cost of evaluating individual theories would then be much lower. Methods of detecting redundant evaluation functions could probably cut down on the size of the hearts state abstraction space. (See Section 4.4.9.)

The number of theories to be tested would then be much lower. Truth-maintenance methods could be used to avoid repeated evaluation of expressions that are shared by many candidates in the theory space. (See Section 5.6.6 below.)

Theory space search algorithm *TSS-C* has somewhat more potential than the other algorithms. This algorithm is designed to find candidate theories meeting both cost and error rate thresholds. It does not attempt to optimize either efficiency or accuracy. (See Section 4.3.) Unlike the other *TSS* algorithms, algorithm *TSS-C* does not examine all theories appearing within a specified cost bound. It uses a *best accuracy first* control strategy and terminates search upon finding any candidate meeting the cost and error rate thresholds. When used with the automatically generated hearts theory spaces, this algorithm would dive deeply into the theory space. It would not waste time examining vast numbers of inaccurate candidate theories. Empirical testing might well be feasible using algorithm *TSS-C* in combination with the hearts state abstraction space produced by *GENERATE-B*. These observations suggest that the *satisficing* goal type *C* is considerably easier to achieve than the *optimizing* goal types *E*, *G*, and *K*. Notice that algorithm *TSS-C* could be used to perform limited efficiency or accuracy optimization by repeatedly invoking the algorithm with successively lower thresholds.

5.6. Enhancements to Theory Space Search

5.6.1. Reducing the Costs of Empirical Learning

Empirical learning in *POLLYANNA* is limited by the computation and information costs of testing candidate theories. Various extensions to the *TSS* module can be envisioned to address these factors and reduce the computational costs of empirical learning. Some extensions would attempt to lower the number of candidate theories that must be considered. For example, intelligent search control strategies might achieve such economies. Other extensions would attempt to lower the number of training examples that must be considered. Were it possible to develop methods that get more mileage out of a given set of examples, the required number of examples might be smaller. Finally some extensions would avoid testing all pairs of theories and examples. Examples of such extensions will be discussed below.

5.6.2. Intelligent Search Control

All of *POLLYANNA*'s theory space search algorithms use simplistic search control strategies. Search might become more efficient if more sophisticated control strategies were used instead. One approach would use *sensitivity analysis* to control search. Sensitivity analysis would attempt to estimate the magnitude and/or direction of the change in the top level evaluation function that would result from each potential refinement of a theory t_1 into t_2. The analysis could begin by noticing which sub-expression is modified by the refinement. It would then use analytic methods like partial differentiation to estimate the sensitivity of the top level evaluation to changes in each sub-expression. Refinements could then be rank ordered according to their likely impact on the top level result.

5.6.3. Extended Analysis of Training Examples

All the *TSS* algorithms test theories against training examples in a very simple way. The tests involve only the top level results of candidate theories. Given an example *situation*, they compare the sets *t-goods* and *t-bads* generated by a candidate theory against the teacher's choices *goods* and *bads*. (See Section 4.5.2.3). This simple comparison ignores potentially useful information that could be extracted from the execution traces of candidate theories. For example, the specific numerical evaluations produced by candidate theories for each card choice are not considered. By examining these evaluations, the system might determine how much the values must change in order to correct errors in the sets *t-goods* and *t-bads*. Given two candidate theories t_1 and t_2 with identical error rates, the numerical evaluations might indicate that t_1 can be corrected by a smaller change than is needed to correct t_2. Such information could be taken as grounds for examining refinements of t_1 before those of t_2.

5.6.4. Use of Enriched Training Examples

A sufficiently cooperative teacher might be able to supply training examples that are more informative than those defined in Figure 4-21. In order to illustrate the value of such enriched examples, it helps to make a distinction between "global" and "local" training information [Keller 87]. Global training information includes only the final answer of a computation. The information is considered "global" it gives only the final answer, without providing any information about the rationale behind the final answer. The training examples described in Figure 4-21 contain purely global information, i.e., they include the sets *goods* and *bads* without any indication of how these sets were determined. In contrast to this, local training information indicates the rationale behind the final answer. It includes data about the series of inferences or computations that lead up to the final answer.

Many types of local information could be usefully included in the training examples processed by *POLLYANNA*. Rather than merely distinguishing between good and bad choices, the teacher might give a partial or total rank ordering of all the choices that are legal in a given situation. He might also indicate whether the good choices are much better or only a little better than the bad ones. Information about the factors that figure into the teacher's choices might also be useful. In the context of a system to learn evaluation functions, the teacher might convey the important factors by indicating which sub-expressions of the evaluation function were considered decisive. Additional local information might include values or rank orderings of sub-expressions that are involved in the computation.

Human domain experts are usually able to supply only qualitative training information. When examples are generated by humans, local training information will likely be limited to rank orderings of choices or rank orderings on the importance of factors. In contrast to this, quantitative local training information may be quite easy to obtain when training examples are generated by machine. Suppose the initial domain theory suffers only from relative intractability so that the exact theory can be used to generate examples (See Section 1.5.1.3). Intermediate results such as the values of sub-expressions could be easily extracted from the execution trace.

Local training information can help *POLLYANNA* with the problem of determining which simplifying assumptions should be blamed for errors observed during empirical testing. Consider how blame is assigned when only global training information is available. When a theory t is observed to make errors, *POLLYANNA* assigns blame by testing each refinement of theory t. The system determines the top level accuracy that results from replacing various function definitions with their primitive refinements. If global accuracy improves after a refinement replacing function version f_1 with f_2, the system implicitly assigns blame to version f_1. The blame assignment problem becomes much simpler if the system has access to local information in the training examples. Suppose the examples provide either rank orderings or numerical values of sub-expressions appearing in the evaluation function. Such information could be compared to the values returned by various function calls during evaluation of a candidate theory. The comparisons could be used to determine which functions are most in error in need of improvement.

5.6.5. Early Testing of Search Goals

Knowledge of the search goal can sometimes be used to avoid exhaustive testing of candidate theories. Suppose the call $Evaluate(theory,examples,K_1,K_2)$ is used to indicate that the system is searching for a theory such that $Cost(t) \leq K_1$ and $Error\text{-}Rate(t) \leq K_2$. The *Evaluate* procedure could maintain a record of the average accuracy after each example is processed. Suppose the average error rate after M examples is E_m. Suppose further that a total of N examples will be processed. The final accuracy E_f must lie in the range: $E_m*(M/N) \leq E_f \leq [M*E_m+N-M]/N$. If the goal accuracy is above this range, then the candidate theory must fail to meet the accuracy bound. Likewise if the goal accuracy is below this range, the theory must successfully meet the accuracy bound. In either case, the remaining examples need not be processed. A similar procedure could be used to predict failure to meet the efficiency bound before all examples are tested. Predicting successful satisfaction of the efficiency bound appears not to be possible, unless one can establish an *a priori* upper bound on the cost of applying a theory to a single training example.

5.6.6. Truth Maintenance Techniques

A truth-maintenance mechanism could be used to reduce the computational expense of empirical testing. Truth maintenance systems are generally useful for determining which inferences remain valid when one assumption is replaced by another [Doyle 79]. In the context of *POLLYANNA*, truth-maintenance methods would determine whether the results of some computations are unchanged when one candidate theory is refined into another. For example, suppose the system has already tested theory t_1 against a set E of training examples. Suppose further that theory t_2 differs from t_1 only by a replacement of version f_1 with version f_2. Let g be a function such that: (a) version g_i is used in both t_1 and t_2 and (b) version g_i does not directly or indirectly call function f. In this case, any function call $g(x)$ will return the same value in theory t_1 as it does in t_2, so long as the argument x is not changed. A truth maintenance system would be designed to detect this situation and avoid recomputing $g(x)$ in theory t_2.

The preceding observation suggests that memoization could be used to implement a truth maintenance system. When theory t_1 is tested against example set E, the results of all function evaluations could be stored in a table. For each function call $g(x)$ returning a value v, the table would store the value v under the key $g(x)$. When theory t_1 is refined into t_2 by replacing f_1 with f_2, all of the evaluations depending on f would be deleted from the table. Entries indexed under $f(x)$ would be removed. In addition, the system would remove all entries indexed under $h(x)$, where h directly or indirectly calls function f. The resulting table could be used to avoid some function evaluations when theory t_2 is tested against the same example set E. This mechanism would treat references between function definitions in the same manner as traditional truth-maintenance systems treat inferential dependencies among logical formulae. A similar mechanism could be used to maintain efficiency measurement records as one theory is refined into another.

5.6.7. Incremental Learning Methods

Incremental processing of training examples might also help to avoid some empirical testing of theories. In contrast to batch mode processing of examples, incremental methods process examples one at a time. These methods take a stream of training examples, $e_1,...,e_i,...,e_n$, as input. As each example is processed, the system maintains and updates a candidate theory, or set of candidate theories, that best fit all the examples seen so far. In the course of processing a new example e_i, the system might replace one candidate set C_i with another candidate set C_j. Incremental methods are especially useful when they can update the current candidates with only limited processing of theories and examples. In particular, one hopes to avoid both: (a) Testing previously rejected candidates against the new example e_i and (b) Testing the new candidate theories in C_j against all the previously processed examples.

Effective incremental methods depend on the organization of the space of candidate hypotheses. For example, the candidate elimination algorithms described in [Mitchell 82] search through a *version space* of concept descriptions organized according to the *generalization-of* and *specialization-of* relations. When an algorithm replaces one candidate concept c_1 with a more specialized concept c_2, it need not examine all previously considered examples. Only the correctly classified positive examples and the incorrectly classified negative examples need be re-examined when moving from c_1 to c_2. The results of reprocessing all other examples can be predicted from the fact that c_2 is a specialization of c_1. The incremental methods described in [Mitchell 82] thus depend on organizing the hypothesis space to reflect semantic relationships among theories.

Theory spaces monotonic in efficiency are particularly ill-suited to exploiting incremental processing of examples. Knowing that theory t_1 can be refined into t_2, it is not generally possible to say anything about the relative accuracy of these two theories. This fact holds true of both the average accuracy of t_1 and t_2 as well as the accuracy on an individual example. If theory t_1 commits an error on example e, theory t_2 may or may not process example e correctly. The *Refinement* relation says only that theory t_2 has equal or lower efficiency than t_1. This extra cost may or may not lead to a more accurate result.

The theory spaces used in *POLLYANNA* might be organized differently in order to exploit incremental processing of examples. It would be desirable to organize a candidate theory space to be monotonically increasing in accuracy with respect to each *individual* training example. Inferences of the following sort would then be possible: If theory t_1 correctly handles example e, and theory t_2 is a refinement of t_1, then theory t_2 also correctly classifies example e. When moving from theory t_1 to theory t_2, it would only be necessary to reconsider the examples on which t_1 committed errors.

A weaker condition could be used to organize a space of theories in which each theory represents an approximate concept description. An approximate concept description space could be partially ordered according to relations of concept inclusion. Initial efforts in this direction are presented in [Keller 83], which described several operations for transforming one concept description into another. The operations are classified according to the type of error they introduce. Operations can be *concept generalizing*, *concept specializing* or *concept preserving*. If an approximate theory space is generated by these operations, incremental methods described in [Mitchell 82] could be used to avoid reprocessing examples when moving from one theory to another.

5.6.8. Starting at the Initial Intractable Theory

The search algorithms described above all commence search at the most efficient theory, i.e., the theory space root. This starting point makes complete sense, for most of the goals in Figure 4-1. It appears to be necessary for goal types *E*, *G* and *K*. A theory *t* cannot be known to satisfy one of these goals, without testing all the more efficient theories lying along the path from *t* to the theory space root. If the simpler theories must be tested, *TSS* might as well start at the simplest.

An alternative strategy is possible for goal type *C*, which asks for a theory meeting specified accuracy and efficiency thresholds. This goal can be established by testing a single theory. Once the system finds a single theory meeting the cost and error rate bounds, the goal is established and the system need not examine any other theory. A search aimed at achieving goal *C* need not start at the theory space root. Algorithm *TSS-C* could be replaced with an alternate algorithm. The new algorithm would start with the initial, intractable theory. This initial theory would gradually be replaced by more efficient ones, moving backwards along the $Refinement(t_i,t_j)$ relation. The best accuracy first control strategy of *TSS-C* could be replaced by a strategy that uses some combined measure of efficiency and accuracy to order the search. Practical difficulties might be encountered in attempting to begin the search with the initial intractable theory. The method would only work if the initial domain theory is only relatively intractable, but not absolutely intractable. (See Section 1.5.1.3).

5.6.9. Alternative Theory Space Organization

An interesting symmetry is exhibited by the list of goal types in Figure 4-1. The set of goal types is invariant when one interchanges the roles of *Cost(t)* and *Error-Rate(t)*. Notice that the symmetry does not carry over to the search algorithms used to process these goals. The search algorithms treat efficiency goals quite differently from accuracy goals. This difference results from the manner in which the theory space is organized. The space is monotonic in efficiency but not in accuracy.

A new family of algorithms would become feasible if it were possible to generate a theory space monotonic in accuracy, rather than efficiency. An approach to generating accuracy monotonic concept description spaces was mentioned above. (See Section 5.6.7.) If such a space can be constructed, each of the algorithms *TSS-C*, *TSS-E, TSS-G and TSS-K* could be replaced by a new algorithm. Each new algorithm would be the same as the old one, with the roles of *Error-Rate(t)* and *Cost(t)* interchanged. For example, *best efficiency first search* would become *best accuracy first search* and vice versa. *Efficiency bounded depth first search* would become *accuracy bounded depth first search*. Pruning conditions previously involving efficiency bounds would be replaced with new ones involving accuracy thresholds. Rather than starting at the theory space root, which has optimal efficiency, each new algorithm would commence searching from the theory with optimal accuracy, i.e., the initial intractable theory.

5.7. Summary of Empirical Learning Results

- **Empirical Testing as a Filter on Analytic Generation:** Empirical learning is useful in *POLLYANNA* for distinguishing between good and bad approximations. Empirical measurements show that analytically generated theories have varying levels of accuracy and efficiency. Some candidates are superior to others on both measures of performance. Empirical tests are therefore needed to distinguish the candidates with Pareto optimal combinations of accuracy and efficiency from the candidates with non-optimal combinations. Empirical learning thus serves as a filter on the imperfect analytic generation process.
- **The Accuracy v. Efficiency Tradeoff Curve:** Empirical learning generates data that precisely quantifies the tradeoff between accuracy and efficiency. A tradeoff curve can be plotted to indicate combinations of accuracy and efficiency that can be attained by selecting from a given set of approximate theories. Empirical measurements show the Pareto optimal theories to provide a moderately flexible tradeoff between accuracy and efficiency. Stochastic combinations of Pareto optimal theories lead to a smoother tradeoff. Tradeoff curves can be used to select an approximate theory that best balances the competing objectives of accuracy and efficiency.
- **Measures of Learning:** Several different experiments demonstrate learning behavior in *POLLYANNA*. One experiment evaluates the importance of training examples. It shows how the results of learning depend on the training data. Another experiment evaluates the importance of the initial intractable theory. It compares learning results in *POLLYANNA* to the results of processing a random space of candidate theories. A third experiment generates learning curves to show how accuracy improves with the number of training examples. A family of curves is used to measure improvements in accuracy that occur at various levels of efficiency. The curves show learning to occur more quickly at high efficiency levels. They also demonstrate knowledge level learning behavior in *POLLYANNA*.
- **The Feasibility of Empirical Learning:** Empirical learning is limited by the information and computation costs of testing candidate theories against training examples. Computation costs appear to be a more significant limiting factor in *POLLYANNA*. Redundant, equivalent candidates drive up computation costs without raising information costs. A variety of techniques could be used to reduce the computation costs of empirical testing. These include elimination of redundant candidates, intelligent search control, intensive analysis of training examples, truth-maintenance techniques, incremental learning and alternative organizations of the candidate theory space, among others.

Chapter 6

Conclusion

6.1. Introduction

A powerful method of learning approximations to intractable theories has been demonstrated in the *POLLYANNA* system. The concept of *generic simplifying assumptions* (*GSAs*) proved crucial in developing a mechanism to transform initially intractable theories into approximate theories of greater efficiency. Integration of analytic and empirical learning methods was essential for distinguishing between good and bad approximations, as well as for measuring the terms of the tradeoff between accuracy and efficiency. The fundamental learning strategy of trading accuracy for efficiency was demonstrated in a variety of experiments.

Experiments with the *approximation generation* module demonstrated the power of generic simplifying assumptions. A number of approximate theories were generated through the interaction of three types of knowledge: intractable theories, generic simplifying assumptions and truth-preserving reformulations. The generality of this framework was demonstrated by experiments using the same set of operations to form approximate theories in the domains of hearts and scheduling. Semantic analysis showed the resulting approximate theories to be equivalent to a variety of hearts and scheduling heuristics.

Methods of analytic reasoning about approximate theories were demonstrated in the *theory space search* module. Theorems describing the efficiency impact of generic simplifying assumptions were formulated and proved. Assumption impact theorems were used to compare relative efficiency of approximate theories. The system makes such comparisons in an effort to build candidate theory spaces that are monotonic in the efficiency of computational theories. Methods of analytically detecting equivalence of candidate theories were demonstrated as well.

The importance of integrated analytic and empirical learning was demonstrated by experiments with the *theory space search* module. Candidate approximate theories were shown to have varying levels of accuracy and efficiency. Empirical learning methods were used to separate the desirable candidates with Pareto optimal combinations from the undesirable ones with suboptimal combinations. The Pareto optimal theories were shown to support a flexible tradeoff between accuracy and efficiency. Empirical tests also generated data to precisely quantify the terms of the tradeoff. This data was shown useful for selecting a candidate theory that strikes an appropriate balance between accuracy and efficiency.

6.2. Evaluation Overview

Experimental and analytic support for the thesis of the dissertation will be evaluated in this chapter. The principal claim asserts that *POLLYANNA* is a viable approach to the intractable theory problem: *The intractable theory problem can be solved by analytically generating approximations from generic simplifying assumptions and empirically testing approximations against training examples.* (See Claim #1 in Section 1.6.1.) The viability claim is supported indirectly through subsidiary claims about features of the *POLLYANNA* architecture. One subsidiary claim argues that the analytic generator is capable of deriving the right sorts of candidate theories. (See Subsidiary Claim #1 in Section 1.6.1.) A second subsidiary claim asserts properties of the system as a whole, including both the analytic and empirical components. Empirical testing is claimed to be a feasible means of separating the desired goal theories from the undesirable candidates. (See Subsidiary Claim #2 in Section 1.6.1.) The subsidiary claims taken together support the primary claim of the thesis.

The viability claim is evaluated in Section 6.3. Criteria used to evaluate the viability of *POLLYANNA* are outlined in Figure 6-1. The first criterion involves properties of the best individual approximate theories generated by the analytic component of *POLLYANNA*. Experiments involving human control show the generator can derive approximate theories with important accuracy, efficiency and semantic properties. These results demonstrate the existence of useful approximate theories lying somewhere in the space of all candidates that can be generated by the system. (See Sections 6.3.2 through 6.3.5.) The second criterion involves properties of the entire system. The system uses empirical tests to distinguish useful candidate theories from all the other candidates generated. Empirical testing will be feasible depending on the information and computation costs of testing candidate theories against training examples. These costs of depend on the numbers of candidates to be tested. Intelligent control strategies serve to limit generation of candidates. Viability of the entire system can thus be evaluated by examining the costs of empirical testing under various search control strategies. (See Section 6.3.6.)

- Accuracy, efficiency, and semantic properties of the best approximate theories generated.
- Information and computation costs of testing candidate theories.
- Context dependent criteria:
  - Accuracy and efficiency goals.
  - Information and computation resources available for learning.
- Generality within and between domains.

Figure 6-1: Criteria for Evaluating the Viability of *POLLYANNA*

The viability of *POLLYANNA* depends ultimately on matters of context. (See Figure 6-1.) The system will succeed or fail depending on the accuracy and efficiency goals derived from a specific performance context, and the resources available for learning in a specific learning context. (See Sections 5.5.4) and 5.5.5.) Some performance contexts will dictate tough accuracy and efficiency goals that cannot be met by theories generated in the analytic component of *POLLYANNA*. Other performance contexts will determine weaker goals that can easily be satisfied. The analytic component will therefore succeed or fail in generating suitable goal theories depending on the intended performance context. The learning context determines the information and computation resources available for learning. The required empirical testing may or may not be feasible within the resource constraints determined by the learning context. Empirical learning should be comparatively easy when the initial theory is intractable relative only to the performance context, and is computationally tractable relative to resources available in the context of learning. The success of *POLLYANNA* will therefore depend on both the performance context and the context of learning.

Generic simplifying assumptions were claimed to provide a domain independent mechanism for deriving heuristics from intractable theories. (See Claim #2 in Section 1.6.3.) This claim will be evaluated by considering generality properties of the *PT-GSA* family of generic simplifying assumptions. (See Section 6.4.) The generic nature of *PT-GSAs* will be examined by considering results of experiments in the hearts and job scheduling domains. Generality properties of reformulation operators will also be discussed. Additional domains for which the *PT-GSAs* appear suited will be examined. A set of domain selection guidelines will be presented as well.

A paradigm of future research on the intractable theory problem is presented in Section 6.8. The paradigm defines several types of research tasks that are likely to be encountered in the course of extending *POLLYANNA* to handle new types of intractable theories. The concept of generic simplifying assumptions (*GSAs*) lies at the heart of the research paradigm. Guidelines for developing new *GSAs* are presented. The process of reconstructing heuristics is examined. Potential instances of the paradigm are discussed as well. Examples of potential *GSAs* appropriate to various representation formalisms are examined. The range of the *POLLYANNA* paradigm is characterized in terms of a distinction between local and global approximation techniques.

6.3. Viability of the *POLLYANNA* Methodology

6.3.1. Non-Deterministic Capabilities of Analytic Generation

The viability of *POLLYANNA* depends in part on properties of the analytic generation component. (See Subsidiary Claim #1 in Section 1.6.1.) Learning can succeed only if the generator can derive at least one approximate theory meeting the specified accuracy and efficiency goals. Capabilities of the generator are supported by experiments with the mechanical, interactive implementation of *POLLYANNA*. The implemented system is capable of mechanically applying *GSA* and reformulation operators to intractable theories; however, it lacks an implemented control structure that would choose appropriate operator sequences. Only the non-deterministic capabilities are tested by the mechanical implementation, since all search control decisions are made by a human. Experiments involving human control nevertheless demonstrate that approximate theories with important properties can be derived through the interaction of intractable theories, *GSAs* and reformulations. Accuracy, efficiency and semantic properties are important in order to establish the overall quality of generated theories. Operationality properties are important in order that generated theories be usable in empirical testing and in the intended performance context. Context sensitivity properties are important in order for sets of candidates to provide a flexible tradeoff between accuracy and efficiency. Experimental evidence supporting the appearance of these properties will be examined in Sections 6.3.2 through 6.3.5.

6.3.2. Semantic Properties of Approximate Theories

Semantic analysis demonstrates that many approximate theories generated by *POLLYANNA* are equivalent to heuristics stated in verbal terms. (See Sections 3.3.1.2 and 3.3.2.2.) Scheduling heuristics equivalent to generated theories include *Soonest Scheduling Deadline First* and *Most Critical Job First*. (The two heuristics *Enable Many Jobs* and *Enable a Well Prepared Job* are special cases of *Most Critical Job First*.) Hearts heuristics equivalent to generated theories include *Lose the Current Trick*, *Lose and Dump High Ranks* and *Lose and Dump High Point Values*. (The *Dump High Rank Rule* and *Dump High Point Value Rule* are special cases of these.) The system also generated approximate theories equivalent to almost all of the hearts heuristics in a goal set formulated prior to implementation, i.e., eight out of nine rules in the heuristic rule set. (See Section 3.4.1.) The mechanically generated hearts heuristics are actually more general than the goal set, and would apply to more complex versions of the hearts game.

The claims of semantic equivalence to hearts heuristics are more strongly supported than the corresponding claims in the scheduling domain. Semantic analysis of hearts theories was supported by empirical tests. Measured accuracy levels were seen to be generally consistent with the game playing strategies manifested in training examples. (See Section 5.3.4.) The semantic analysis of scheduling theories was not supported by empirical testing. The semantic claims for scheduling therefore rest entirely on manual algebraic analysis of the approximation generator output. (See Appendix M.) Empirical tests of approximate scheduling theories would provide additional support for the semantic claims.

Semantic analysis is useful for evaluating approximate theories. Once an approximate theory has been shown equivalent to a verbal heuristic, it can be evaluated in terms of quality and originality. (See Sections 3.4.3 and 3.4.4.) In terms of originality, *POLLYANNA* has *reconstructed* heuristics specifically anticipated by the designer. The system has *rediscovered* heuristics not anticipated by the designer, but which may have been previously known to other humans. The system has also *discovered* some heuristics that probably have not ever been previously formulated by anyone. In terms of quality, *POLLYANNA* has generated heuristics that are *not plausible*, *plausible* and *novice*. (See Sections 3.4.3 and 3.4.4 for the specific heuristics in each category.)

Efficient heuristics performing on the level of a human *expert* cannot currently be generated by *POLLYANNA*. Approximate theories semantically equivalent or superior to expert heuristics do appear in the underlying approximate theory space; however, they are highly inefficient. This difficulty is largely a result of efficiency cliffs. (See Section 3.6.) Efficiency cliffs so drastically increase the execution costs of theories that they become effectively useless. A variety of methods were proposed to attack such difficulties and ultimately generate expert level heuristics. (See Section 3.6.) Some proposals involved developing new *GSA* or reformulation operators. Others involved careful strategies for representation of domain theories. Still others would require applying the *POLLYANNA* methodology to an entirely new and richer representation formalism. Additional research is needed to determine whether these proposals will succeed.

POLLYANNA is nevertheless quite successful in generating heuristics that are plausible or that perform on the level of a human novice. These results are especially significant considering the nature of the derivation process. Inputs to the derivations include only the initial intractable theory, generic simplifying assumptions and truth-preserving reformulations. The heuristics were not built into the theories. The possibility of such heuristics is not immediately obvious upon examining the initial domain theories. The heuristics fall out naturally from a process of instantiating generic simplifying assumptions at various points in the domain theories. A small and well defined set of operations proved sufficient for generating a variety of heuristics in both the hearts and scheduling domains.

6.3.3. Accuracy, Efficiency and Optimality of Approximate Theories

Empirical tests show that *POLLYANNA* generates approximate theories across a wide range of accuracy and efficiency levels. All the generated hearts theories were nevertheless efficient enough to be tested within reasonable computational resources. Each is far more efficient than the initial domain theory. (See Section 5.3.2.) Accuracy levels of theories were measured in comparison to random guessing. The worst were roughly equal in accuracy to results of random guessing. The best were shown to be considerably better than random guessing. (See Section 5.4.3.) Experimental evidence also demonstrates that empirical testing of theories is useful. Measurements of accuracy and efficiency demonstrated that both Pareto optimal and non-optimal theories are generated by *POLLYANNA*. (See Section 5.3.2.) Empirical testing is useful for separating the optimal candidates from the non-optimal ones. (See Subsidiary Claim #2 in Section 1.6.2.2.)

6.3.4. Operational Usability of Approximate Theories

Operationality properties of generated theories were demonstrated by experiments with the empirical component of *POLLYANNA*. All approximate theories generated by *POLLYANNA* are claimed to be operationally usable, i.e., they can be executed on a real machine. The generated hearts theories were all tested for usability in the *theory space search* module of *POLLYANNA*. Each was shown to be capable of making card choice decisions when given training examples describing game situations as input. (See Section 5.3.2.) Although there is no reason to expect a different outcome in the scheduling domain, none of the scheduling theories were actually tested for usability. The hearts results nevertheless provide considerable support for the claim of usability.

Operational usability is especially significant considering the types of simplifying assumptions used to generate heuristics in *POLLYANNA*. The generic simplifying assumption *Equiprobable Random Variables (EP)* is the workhorse of the approximation generator. Assumptions of this type correspond to Bernoulli’s Principle of Insufficient Reason. (See Section 2.5.2.) Bernoulli’s principle has received attention in a variety of contexts, most notably in Bayesian decision theory [Jaynes 83a]. The informal principle is easy to state. Implementation of the principle is unfortunately not so straightforward. The entire *PT-FORMALISM* was developed mainly for the purpose of implementing Bernoulli’s principle. (See Section 2.4.1.) The value of *EP* assumptions was recognized early in the course of attempting to reconstruct hearts heuristics. As the *PT-FORMALISM* was developed and derivations were implemented, additional assumptions and reformulations were needed, including assumptions of *Probabilistic Independence (IN)* and the reformulations *Probability/Expectation of Composite Functions (PC/EC)* (See the discussion of interactions between approximations and reformulations in Section 2.6.5.) As a result of this development process, *POLLYANNA* is able to convert the informal Principle of Insufficient Reason into operationally usable heuristics.

6.3.5. Context Sensitivity of Approximate Theories

Context sensitivity is important in order that *POLLYANNA* learn approximate theories suited to varying performance contexts. The context sensitivity property is demonstrated by the results of empirical testing. (See Section 5.3.3.) *POLLYANNA* was shown to generate approximate theories across a spectrum of accuracy and efficiency levels. Theories generated at one end of the spectrum are highly efficient, but rather prone to errors. Theories at the other end of the spectrum are highly accurate, but more expensive to execute. Theories at intermediate points were generated as well. Accuracy levels across the spectrum were shown to vary by a factor of about *2.9*. Efficiency levels across the spectrum were shown to vary by a factor of about *830*.

POLLYANNA's empirical learning modules were shown to produce data that precisely quantifies the tradeoff between accuracy and efficiency. (See Subsidiary Claim #3 in Section 1.6.2.2.) The tradeoff can be measured by analyzing the accuracy and efficiency] levels of Pareto optimal theories. A tradeoff of limited flexibility was shown to result from selecting individual Pareto optimal theories. A smoother tradeoff was shown to result from building composite theories out of pairs of Pareto optimal theories. A smooth tradeoff is important for context sensitive learning. Each point along the tradeoff curve strikes a different balance between the competing goals of accuracy and efficiency. The appropriate balance will depend on learning goals derived from performance context. *POLLYANNA* thus provides a flexible means of tailoring an domain theory to suit the intended performance environment.

6.3.6. The Feasibility of Complete Automation

The ultimate viability of *POLLYANNA* depends on developing a completely automatic control structure. The implemented system is non-deterministic. Operators were defined and implemented, but no automatic control structure is used to restrict their application. Successful control strategies must meet two conditions: (1) *Weakness*: Control restrictions must be weak enough so that suitable goal theories are not omitted from the candidate theory space. (2) *Strength*: Control restrictions must be strong enough so that a manageable number of candidate theories are generated. Control is a difficult problem because these two conditions are somewhat in conflict with each other. Evidence of the feasibility of automatic control was presented in the preceding chapters and will be reviewed below. (See Subsidiary Claim #2 in Section 1.6.2.2.)

The approximation generator control strategy *GENERATE* was proposed in Section 2.7. This procedure was formulated as a generalization of regularities that appeared in human control traces. A method of using means/ends analysis was proposed to extend this procedure to handle a broader range of constructs. Two restrictive strategies were proposed as well. *GENERATE-A* was proposed to limit redundancy by exploiting information about the representation of problem states. *GENERATE-B* was proposed to generate only distinct problem state abstractions, by limiting some of the ambiguity inherent in Bernoulli's Principle of Insufficient Reason.

The weakness condition was addressed by examining coverage properties of the proposed automatic control strategies. (See Section 3.4.5.) Fully implemented versions of *GENERATE*, *GENERATE-A* and *GENERATE-B* would likely produce approximations that are identical or similar to the ones resulting from human control, along with additional approximations as well. Evidence for this fact comes from analyzing traces of human control. The human generated results appear to correspond closely to approximations that would be result from the automatic control procedures. Many of the human generated approximations can be identified with specific approximate function versions that would be produced by the automatic procedures.

The strength condition was addressed by deriving analytic bounds on the numbers of candidate theories produced by each of the proposed control strategies. (See Section 4.4.13.) Analytic bounds were used to estimate the numbers of additional theories that would result if the automatic control procedures generated all the approximate theories generated under human control. These analytic results were then used to estimate the difficulty of empirical testing in the theory space search module. (See Section 5.5.) Empirical testing would not be feasible using theory space search algorithms *TSS-E*, *TSS-G* and *TSS-K* which perform exhaustive search for optimizing goals. (See Section 4.5.1.) The candidate theory spaces resulting from the proposed automatic control would be too large for exhaustive search. Empirical testing might be feasible using algorithm *TSS-C*. This algorithm solves satisficing goals, rather than optimizing goals. It uses a control strategy that avoids exhaustive search.

Empirical learning is limited by the information and computation costs of testing candidate theories against training examples. Computation costs appear to be a more significant limitation in *POLLYANNA*. The information costs of obtaining training examples appear to be far less limiting. This is a consequence of the presence of redundant, equivalent theories in the candidate theory spaces produced by the system. (See Section 5.5.) The existence of such redundancy was indicated by empirical tests as well as by analysis of invariance properties of choice evaluation functions. (See Sections 4.4.9 and 5.3.5.) Redundant theories raise the computational costs of empirical testing if each candidate theory must be tested separately. They appear not to raise the information costs of distinguishing the accuracy levels of semantically distinct candidates.

A variety of approaches were proposed for limiting the computational costs of empirical testing. Efficiency monotonic theory space organization enables pruning operations that avoid empirical testing of some candidates. (This proposal was actually implemented in *POLLYANNA*.) Methods of detecting redundant evaluation functions would lower the numbers of candidate theories generated. (See Section 4.4.9.) The cost of candidate theory execution could be lowered by applying reformulations that write approximate theories in simpler equivalent forms. (See Section 3.6.) Truth-maintenance and incremental learning techniques would each help avoid fully executing each theory against each example. (See Sections 5.6.6 and 5.6.7.) Decomposability properties of the theory space might also be exploited for this purpose. (See Section 5.3.6.) Intelligent theory space search could be guided by sub-expressions sensitivity analysis. (See Section 5.6.2.) It could also be guided by local training information that helps assign blame for errors committed by approximate theories. (See Section 5.6.4.)

The task of complete automation has been considerably clarified by this research. Several automatic control strategies have been proposed and analyzed. The strategies appear to be quite promising; however, they require further development. The key issues that affect automation have also been identified. Automation is impeded mainly by a surfeit of redundant theories that raise the computational costs of empirical testing. Methods of avoiding redundancy and lowering the computational costs have been proposed. Although the issue remains open in the absence of a working demonstration, this study has made considerable progress toward demonstrating the feasibility of complete automation.

6.4. Generality and Domain Independence

POLLYANNA has been shown to use a domain independent mechanism for generating heuristics. Starting with an initial domain theory, the system generates heuristics by applying sequences of operators implementing generic simplifying assumptions (*GSAs*) and truth-preserving reformulations. Experimental results have demonstrated that these operator are truly *generic*. Several different types and degrees of generality have been demonstrated. *GSAs* and truth-preserving reformulations both exhibit formal, prima facie generality, as well as useful generality. They manifest generality within single domains, as well as generality across domains. The evidence supporting these claims of generality will be reviewed below. (See Claim #2 Section 1.6.3.) The following sections concern primarily the generality of the *GSAs* and reformulations implemented in this research. Generality of the underlying *POLLYANNA* methodology will be discussed in Section 6.8.

6.4.1. Evidence of Domain Independence

Prima facie generality is evident upon examining the definitions of *GSA* and reformulation operators used by *POLLYANNA*. (See Sections 2.5 and 2.6 and Appendices F and G.) The operator definitions involve only domain independent concepts like functions and random variables. These concepts can be instantiated to the specific functions and variables that are relevant to many domains of interest. They are formally applicable to any domain theory that can represented in the *PT-FORMALISM*. (See Section 2.4.1.)

Generic simplifying assumptions were shown to exhibit useful generality across domains. Useful generality was demonstrated by deriving heuristics for both hearts and scheduling. Plausible heuristics for each domain resulted from instantiating the same *GSA* operators in two different contexts. This result is especially significant considering how the *PT-GSA* family was originally developed. The *PT-GSAs* and *PT-FORMALISM* were developed by attempting to reconstruct known heuristics for the hearts domain. Both were formulated and completely implemented before job scheduling was even chosen as a second domain of investigation. These two domains would not normally be considered particularly similar to each other. Despite the apparent differences between these domains, the *PT-GSAs* developed in studying hearts were found to derive plausible heuristics for scheduling as well.

Generality across domains was examined in detail by comparing the rates at which individual *GSA* and reformulation operators were used in each domain. (See Section 3.5.2.) Every *GSA* operator was used at least once in each domain. Many were used several times in each domain. Generality across domains was thus established at the level of individual *GSA* operators. Similar results established generality across domains of individual reformulation operators. Most reformulation operators were used in each of the two domains. The analysis revealed somewhat less generality across domains for the reformulation operators than for the *GSA* operators. This result suggests that *POLLYANNA* does not yet contain a sufficiently complete set of reformulation operators. New reformulations will likely be necessary as new domain theories are investigated, before a complete set is ultimately attained.

Generic simplifying assumptions were also shown to exhibit useful generality within single domains. This fact was demonstrated by examining the individual operator applications used to generate heuristics. (See Section 3.5.1.) Derivations of several mutually inconsistent hearts heuristics were seen to involve common *GSA* operators. In some cases different heuristics were derived using different instantiations of the same operator. In other cases they shared the same instantiations. Generality within a domain thus occurs at the level of generic and specific simplifying assumptions. The same phenomena were observed in the scheduling domain. These analyses indicate the variety of heuristics that result from systematic instantiation and combination of generic simplifying assumptions within a single domain.

The originality properties of *POLLYANNA*'s heuristics can be attributed to the generality of generic simplifying assumptions. (See Section 2.2.4.1.) Originality of *POLLYANNA*'s hearts heuristics resulted from generality *within* a single domain. The *PT-GSAs* were developed by reconstructing known hearts heuristics such as the *Dump High Rank Rule* and the *Dump High Point Value Rule*. (See Section 2.2.1.) Additional hearts heuristics were discovered or rediscovered by systematically applying these *GSA* operators throughout the hearts domain. Originality of *POLLYANNA*'s scheduling heuristics can be attributed to generality *across* domains. Scheduling heuristics were rediscovered by transferring *GSAs* developed for hearts and instantiating them in the context of the scheduling domain.

6.4.2. Guidelines for Selection of Application Domains

The vantage of hindsight permits identifying features that are common to the hearts and scheduling domains. It appears possible to isolate a common structure that may partly account for *POLLYANNA*'s success in generating plausible heuristics for these two domains. The shared features do not limit the *formal* generality of the generic simplifying assumptions used in *POLLYANNA*. The similarities may nevertheless be relevant to determining when the *PT-GSAs* will be *useful* for generating plausible heuristics. The shared features could then be used as guidelines for selecting appropriate application domains.

Hearts and scheduling are both structured in terms of a set of decisions *D*. Each decision selects an item from a stockpile *S*, e.g., cards or jobs. The selected item is assigned to a particular decision point, e.g., tricks or time slots. The decisions collectively define a tuple $(i_1,...,i_n)$ that associates one stockpile item with each decision point. Decisions are evaluated in terms of an optimization criterion *F* expressible as a function of the complete tuple $F(i_1,...,i_n)$. Heuristics are developed by splitting the function *F* into sub-expressions representing factors that figure in the optimization criterion, e.g., winning a trick, the value of a trick, meeting job deadlines, satisfying job preconditions. Each heuristic focuses attention on some factors and ignores other factors.

Consider floor planning as a potential application domain. This domain appears to correspond well to the structure shared by hearts and scheduling. A floor plan is an assignment of rooms to locations on a two dimensional surface. Assignments are required to meet adjacency conditions among pairs of rooms. Floor plans are evaluated according to the total amount of area consumed. Rooms correspond to cards in hearts and jobs in scheduling. Locations correspond to tricks in hearts and time slots in scheduling. The analogy to scheduling is especially compelling. Adjacency constraints correspond to job preconditions. Area usage constraints correspond to job completion deadlines. These observations strongly suggest that *POLLYANNA*'s methods could be used to learn floor planning heuristics. One floor planning heuristic might guide decisions by focusing attention on area minimization. Another heuristic might focus on meeting adjacency constraints.

Intelligent automated design appears to be a potentially fruitful area of application for the *PT-GSA* family of generic simplifying assumptions. The *PT-GSAs* might well be useful for developing evaluation functions that guide search through a space of possible designs. Approximate evaluation functions would estimate the quality of partially specified designs. Estimates would result from using *Equiprobable Random Variable (EP)* assumptions to reason about properties of the unspecified portions of the proposed partial design. This approach would likely be most effective in design domains for which intractability results less from the need to search a large design space, than from the cost of evaluating individual proposed designs. *POLLYANNA* appears to be especially good at splitting costly evaluation functions into distinct decision factors, and subsequently learning to distinguish the critical factors from the non-critical ones.

Domains involving hierarchical reasoning appear especially well suited to the approximation techniques developed in this research. *POLLYANNA* generates heuristic evaluation functions by dividing computations into subexpressions corresponding to separate factors involved in decisions. The system separately approximates each subexpression in order to learn which factors are critical, and which are non-critical. The same approximation strategy might well apply to other types of hierarchical problem solving. Top down planning or design problems seem likely to fit into this framework. Top level problems would be decomposed into subproblems which are then approximated separately. The system would learn to distinguish critical subproblems from non-critical subproblems. Computational resources could then be allocated away from non-critical subproblems amenable to quick and dirty methods, and toward critical subproblems requiring more intensive problem solving efforts.

6.5. Principled Derivations of Heuristics

POLLYANNA has been shown to use a principled method of deriving heuristics. The method is considered "principled" because it identifies both the facts and the assumptions on which derived heuristics depend. (See Claim #3 in Section 1.6.4.) The derivation structure records the facts from the initial theory *(IT)* and the instantiated generic simplifying assumptions *(GSAs)* used in the derivation. If the initial theory is correct and the instantiated *GSAs* are assumed to be true, the heuristic follows as a logically sound conclusion. This soundness property depends on the truth-preserving status of reformulations used in derivations. The reformulations used in *POLLYANNA* are all theorems of probability theory, logic, set theory or algebra. In most cases, this fact was obvious from the definitions of reformulation operators. Informal justification was provided in the few remaining cases. (See Appendix G.) The heuristic derivation methods used in *POLLYANNA* may be considered "principled" for an additional reason as well. Assumptions of *Equiprobable Random Variables (EP)* and *Probabilistic Independence (IN)* both describe maximum entropy probability distributions. (See Section 2.5.5.) The entropy concept provides a limited *a priori* justification for these particular generic simplifying assumptions.

Insight into the underlying nature of heuristics often follows as a fringe benefit from the principled derivations used in *POLLYANNA*. A derivation structured as *H=IT+GSA+R* decomposes a heuristic into several components. Some parts correspond to hard facts (*IT* and *R*). Other parts are assumptions (*GSAs*). For example, the *Dump High*

Rank Rule derivation depends partly on a *fact* from the domain theory describing how high rank cards defeat low rank cards. It also depends partly on an *assumption* that opponents are equally likely to play any card in the deck. Principled derivations thus reveal the rationale underlying plausible heuristics. They also help determine limits to the range of application of heuristics. Heuristics represented in this manner will presumably be more likely to fail in contexts where the assumptions are known to be violated. They will also be expected to require modification when the supporting facts of the initial theory are changed. This observation suggest using derivational analogy techniques to update heuristics to reflect changes in the initial domain theory. (See Section 2.2.5.)

6.6. Comparisons to Existing Machine Learning Techniques

6.6.1. Explanation-Based Generalization (*EBG*)

POLLYANNA is capable of learning heuristics for intractable theories outside the scope of explanation-based generalization. (See Claim #6 in Section 1.6.7.) *EBG* applies only to theories that are relatively (but not absolutely) intractable or that are deterministically (but not non-deterministically) intractable. (See Section 1.5.1.) The hearts domain theory suffers from absolute, non-deterministic intractability. (See Section 2.4.2.) *EBG* therefore fails to handle the hearts theory because it cannot build complete explanations of training examples. *POLLYANNA* does learn heuristics for hearts and therefore applies to a domain theory outside the scope of *EBG*. *POLLYANNA* succeeds in hearts by sacrificing accuracy in order to gain efficiency in return. *EBG* fails in using truth-preserving methods to improve efficiency. The failure of *EBG* concerns only the hearts domain *theory*. *EBG* might succeed with some other computational theory of the hearts *domain*, provided that theory is either relatively (but not absolutely) intractable or deterministically (but not non-deterministically) intractable. No such computational theory is currently known to exist for hearts.

6.6.2. Traditional Inductive Learning

Traditional inductive learning programs can also be compared to *POLLYANNA*. (See Conjecture #1 in Section 1.6.7.) Consider the case of a program that learns by selecting a member of a fixed space of candidate hypotheses [Mitchell 82]. Such systems learn effectively to the extent that the hypothesis space is biased to suit the domain under study [Mitchell 80]. Such programs are unfortunately rather rigid. They may work well in one domain, only to fall apart upon encountering a domain for which the hypothesis space is not well suited.

POLLYANNA potentially avoids rigidity by generating a candidate hypothesis space that depends on the domain under study. Candidate theories are generated by applying operators to an initial domain theory. When the domain changes, the initial theory changes and the space of candidates changes as well. Analytic learning results from the hearts and scheduling domains have shown *POLLYANNA* to generate a space of candidate theories that depends on the initial domain theory. In exchange for such advantages, users of *POLLYANNA* must pay the price of building a new domain theory, or modifying an existing one, for each new domain of application.

POLLYANNA is not the only system capable of generating a hypothesis space from an initial domain theory. Some alternative approaches to the same task are described in [Russell and Grosof 87; Flann and Dietterich 86; Ellman 89a; Danyluk 89]. For example, methods using a declarative knowledge base to generate candidate hypotheses are discussed in [Russell and Grosof 87]. Methods of using tractable fragments of an intractable theory to enrich the representation of instances prior to an empirical learning phase are discussed in [Flann and Dietterich 86]. Such systems might be usefully evaluated in terms of cost effectiveness, i.e., the rate and quality of learning versus the difficulty of engineering an appropriate initial domain theory. Additional research is needed to determine the cost effectiveness of each approach.

6.7. Knowledge Level Learning in *POLLYANNA*

Empirical tests demonstrated knowledge level learning behavior in *POLLYANNA*. (See Claim #7 in Section 1.6.7.) Learning curves were plotted to measure improvements in accuracy as a function of the number of training examples processed in the theory space search module. (See Section 5.4.) An entire family of curves was needed to measure learning progress. Each curve measured improvements in the best accuracy that can be obtained within a some fixed computational cost bound. Most learning curves exhibited gradual but non-monotonic improvements in accuracy. Learning occurred rather slowly when candidates are constrained by tight efficiency bounds. Learning occurs more quickly when efficiency constraints are lenient. A possible explanation was suggested to account for these observations. If efficiency bounds are tight, all available candidate theories are highly simplistic. They fail to distinguish themselves from a randomly guessing theory until the training set becomes quite large. If efficiency bounds are lenient, sophisticated theories are then available. They distinguish themselves from a randomly guessing theory after a small number of examples.

6.8. *POLLYANNA* as a Paradigm

The research underlying *POLLYANNA* can be viewed as one step along the path toward a solution to the intractable theory problem. Progress toward this goal is visible at two levels in this research. One level concerns a particular set of generic simplifying assumptions, the *PT-GSA* family, and a particular formalism for representing intractable theories, the *PT-FORMALISM*. Experimental results have demonstrated that a variety of plausible heuristics can be learned by applying *PT-GSAs* to theories in the *PT-FORMALISM*. Considerable evidence has been assembled to demonstrate the viability of these specific techniques within the *POLLYANNA* architecture.

At a higher level of abstraction, this research has demonstrated a methodology for attacking the intractable theory problem. *POLLYANNA* may be seen as an instance of a paradigm. (See Conjecture #3 in Section 1.6.8.) The paradigm describes how to investigate new types of intractable theories by developing new families of generic simplifying assumptions. Several distinct components of the paradigm will be discussed below. They describe the types of research tasks likely to be encountered in the course of extending this research. Continued research within the paradigm holds considerable promise of leading to a more complete solution to the intractable theory problem.

6.8.1. Components of the Paradigm

The *POLLYANNA* paradigm rests on a fundamental conjecture about the nature of heuristics: *All problem-solving heuristics can be derived by applying generic simplifying assumptions and truth-preserving reformulations to an initially intractable theory: H=IT+GSA+R.* (See Conjecture #2 in Section 1.6.8.) Components of the paradigm flowing from this conjecture are listed in Figure 6-2. Each component describes a distinct kind of knowledge that must be developed in order to handle a new type of intractable theory. The first component is a knowledge representation formalism, i.e., a language for representing intractable theories. Each formalism will be associated with a family of *GSAs* and *Rs* that are designed to interface with theories represented in that formalism. Another component consists of analytic techniques for reasoning about properties of approximate theories. Each such technique will likely be specialized to a particular formalism and the approximate theories that result from a particular family of *GSAs*. The final component comprises control strategies that guide application of *GSAs* and *Rs* to limit the potentially explosive number of generated approximate theories.

- A knowledge representation formalism.
- A family of generic simplifying assumptions.
- A family of truth-preserving reformulations.
- Analytic techniques for reasoning about the accuracy, efficiency, equivalence, consistency or completeness of approximate theories.
- Control strategies that guide search through the space of approximate theories by limiting the application of generic simplifying assumption and reformulation operators.

Figure 6-2: Components of the Paradigm

6.8.2. A Methodology for Reconstruction of Heuristics

New generic simplifying assumptions can be developed through a process of reconstruction of known heuristics. The *PT-GSA* family used in *POLLYANNA* was developed by reconstructing hearts heuristics. (See Section 2.2.6.) Insights into the reconstruction process have been extracted from this experience. The overall process is outlined in Figure 6-3. The process begins with a *plausible* heuristic *H*, e.g., the *Dump High Rank Rule*. An iterative process proceeds to extract all the facts *F* from the domain theory on which the heuristic depends, e.g., the rule that high cards defeat low cards. On each iteration, *H* is decomposed into facts *F*, a less plausible heuristic *H*' and reformulation knowledge *R*. The process terminates when the heuristic is no longer plausible. Absence of plausibility suggests that no more domain knowledge is compiled into the heuristic. The final version of the heuristic is then considered a specific simplifying assumption *SA*, e.g. the assumption that opponents are equally likely to play any card. A generic simplifying assumption (*GSA*) results from generalizing the specific simplifying assumption into a domain independent statement, e.g., *Equiprobable Random Variables (EP)*.

```
Reconstruct(H)

     1. While (H is plausible) do
        a. Decompose H into F + H' + R.
        b. H ← H'.
     2. SA ← H.
     3. GSA ← Generalize(SA).
```

Figure 6-3: Reconstruction of Heuristics

A meta-heuristic can be now formulated to guide the process of developing new generic simplifying assumptions. *GSAs* should result from generalizing specific assumptions with a property called "*neutral plausibility*". They should be neither especially plausible nor especially implausible. Implausible assumptions are undesirable for the obvious reason that they tend to make errors. Plausible assumptions are also undesirable as candidates to become generic simplifying assumptions. Plausibility is likely to result from domain knowledge implicitly contained in the assumption. Plausible assumptions are therefore not likely to be domain independent and truly generic.

As an illustration of the meta-heuristic, consider the derivation of the *Dump High Rank Rule*. (See Section 2.2.1). Reconstruction of this rule resulted in a derivation based on a specific assumption that opponents are equally likely to play any card. The specific assumption was generalized into *Equiprobable Random Variables (EP)*. A skeptic might ask why the original heuristic could not itself be generalized into a generic simplifying assumption, e.g., *Pick the Highest Numbered Choice*. Generalization of the *Dump High Rank Rule* is dangerous precisely because of its plausibility. The dumping heuristic is plausible in part because of the fact that high cards defeat low cards in hearts. The heuristic might become less plausible if the rules of hearts were changed. In contrast to this, the specific *EP* assumption is not so plausible and is not vulnerable to changes in the domain theory. Neutral plausibility makes it a good candidate for generalization.

6.8.3. Potential Instances of the Paradigm

The experience of *POLLYANNA* suggests that approximation and reformulation interact with each other in significant ways. (See Section 2.6.5.) Some reformulations serve mainly to enable subsequent *GSA* applications. Some *GSA* applications serve mainly to enable subsequent efficiency improving reformulations. Interactions of these sorts appear to be quite widespread. Examples such interactions will be presented below. They provide evidence supporting the generality of the heuristic generation framework *H=IT+GSA+R*, and the associated paradigm for attacking intractable theory problems.

Problem spaces defined by *STRIPS* operators fit well into the paradigm [Fikes et al. 72]. A *STRIPS* operator is defined by a list boolean expressions defining preconditions of the operator, as well as a list of boolean expressions defining the effects of the operator. Generic simplifying assumptions could be defined to simplify *STRIPS* problem spaces by ignoring selected preconditions of operators. For example, operations similar to *Truify* and *Falsify* could assert that certain preconditions will always or never be met. Such operations would be quite similar to previously studied methods for generating abstraction spaces [Sacerdoti 74; Pearl 84]. Efficiency improving reformulations can be defined for *STRIPS* problem spaces as well. One such reformulation operates by *factorization of subproblems* that can be shown not to interact with each other [Mostow and Prieditis 89]. Interactions between approximations and reformulations can be observed in the context of *STRIPS* problem spaces. Subproblem factorization is sometimes possible only after making approximations. Some useful approximations are possible only after applying reformulations that enrich the representation of problem states. Interactions of both types are present in the work of [Mostow and Prieditis 89]. *Goal regression* is also a standard type of reformulation often applied to *STRIPS* operator sequences [Nilsson 80]. Possible interactions between goal regression and approximation would be interesting to study.

Algebraic and differential equations are also examples of representation formalisms that might fit into the paradigm. *Taylor series expansions* are a type of reformulation that often applies usefully to algebraic and differential equations. A Taylor series expansion is often used to set up subsequent approximations that drop high order terms. The most common approximation assumes *linearity* by ignoring all but zeroth and first order terms. Assumptions of linearity often enable subsequent reformulations that produce closed form solutions. Solutions to differential equations are often facilitated by assumptions of *symmetry properties*, especially spatial symmetries such as translational, spherical or cylindrical symmetry. Reformulations useful for solving differential equations include *coordinate transformations* and *separation of variables*.

Techniques for approximating combinatorial optimization problems might also fit into the paradigm. One approximation technique involves solving a *relaxed version* of the original optimization problem [Papadimitriou and Steiglitz 82; Pearl 84]. Consider the problem of finding the minimal value M_s of function $F(x_1,...,x_n)$ over the set $S \subset I^n$. A relaxation technique begins by approximating set S with some superset R. The superset R is chosen to be amenable to a closed form solution, i.e., a type of efficiency improving reformulation. The minimal value M_r of $F(x_1,...,x_n)$ on set R is a lower bound on M_s. The opposite technique of solving an *over constrained version* of the original problem leads to an upper bound. Values in between the upper and lower bounds represent approximate solutions. These techniques could be mapped into the *POLLYANNA* paradigm by defining generic simplifying assumptions to add or delete constraints on the set over which the function is being optimized. Reformulations would be defined to implement closed form solutions that apply to well behaved sets.

Additional techniques for approximate combinatorial optimization operate by simplifying the function $F(x_1,...,x_n)$ to be optimized. One technique proceeds by finding a linear approximation to $F(x_1,...,x_n)$ making the problem amenable to a closed form solution, e.g., using a *Taylor expansion* and assumptions of *linearity*. Yet another approach uses *decomposition* by finding an approximate version of F with the form $F(x_1,...,x_n) \approx M(G(x_1,...,x_k),H(x_{k+1},...,x_n))$ where $M(x,y)$ is a monotonic function. The search space is then amenable to *factorization* since the global minimum is composed of two smaller parts that are respectively minimal on functions G and H. Reformulations involving *parameter transformations* are often crucial to this process, since decomposition often depends on having an appropriate parameterization of the choice set.

6.8.4. Limits of the Paradigm

The *POLLYANNA* paradigm will naturally fit some types of intractable theory problems better than others. Some components of the paradigm may not be present in any given application. Consider the issue of operational equivalence among theories. Equivalence was a problem in the *POLLYANNA* implementation due largely to the invariance properties of evaluation functions. Tools for detecting equivalence were therefore needed. Equivalence may not be such a problem outside the context of evaluation functions. Consistency and completeness are another example. These were trivial matters given the function oriented representation used in *POLLYANNA*; however they may be more significant for representations based on predicate calculus. (See Section 4.4.10.)

Search control may be relatively unimportant in some applications. Consider the problem of finding a polynomial approximation to a function $F(x)$. A Taylor expansion might be used if the interval of relevance is known and is small. The space of candidate theories would be parameterized by the number of terms included in the expansion. Control would not be a problem since the candidates could be totally ordered according to efficiency. If the series converges uniformly, all the candidates are Pareto optimal. Notice that empirical learning may still be useful. Empirical tests could generate data measuring accuracy and efficiency as a function of the number of terms included in the expansion. A tradeoff curve would help decide how many terms to use. This example illustrates a special type of empirical learning problem. Empirical tests are used only to measure the tradeoff, and not to distinguish between optimal and non-optimal candidates.

The heuristic generation framework, *H=IT+GSA+R*, seems best suited to approximation techniques involving incremental, local operations, rather than global operations applied in a single step. (See Section 2.3.3.) Local operations apply to syntactically distinct portions of theories. For example, *Function Invariance (FI)* is applied to one expression at a time. Most of the heuristics generated in *POLLYANNA* resulted from an incremental process of applying many generic simplifying assumptions. Global operations apply to a theory as whole. Some approximation techniques seem to apply global operations in a single step. Examples of such unitary global simplification methods include the following: (1) *Scaling*: Replace a large problem with a smaller problem homomorphic to the large one; (2) *Perturbation*: Find a well behaved problem that is close to the original one, and solve the original one by modifying the solution to the well behaved one. (3) *Qualitization*: Change quantitative variables into qualitative ones. Global approximation techniques may well fall outside of the *POLLYANNA* paradigm.

6.8.5. Extensions of the Paradigm

6.8.5.1. Abstraction: Trading Scope or Precision for Efficiency

An alternative approach to the intractable theory problem uses *abstraction* rather than approximation as a fundamental simplification technique. Abstraction may be defined as an operation that improves efficiency by limiting the *scope* or *precision* of a computational theory. Scope is limited by constraining the set of problems solved. Precision limited by lowering the amount of information provided about solutions. Abstraction is a truth-preserving process. The abstract theory says less, but speaks just as correctly as the original theory. Abstraction sacrifices scope or precision to gain efficiency, just as approximation sacrifices accuracy to gain efficiency in return.

Abstraction techniques are widely used in qualitative reasoning about physical systems [Bobrow 85]. For example, one qualitative reasoning technique involves replacing quantitative differential equations with qualitative ones [de Kleer and Brown 85]. The behavior of a system is then simulated using the qualitative model. Qualitative models may be more efficient than quantitative ones; however, they pay a price in the precision of their predictions. Simulations of qualitative models often fail to predict precise outcomes. In some cases a range of outcomes will be predicted. Existing qualitative reasoning systems are usually constructed by hand. Techniques similar to the ones used in *POLLYANNA* might be developed to automate the process of transforming precise intractable quantitative models into efficient models that provide only qualitative descriptions of physical systems.

6.8.5.2. An Expanded View of the Intractable Theory Problem

An expanded view of the intractable theory problem is provided by the table in Figure 6-4. The table lists three types of theories, each with a specific type of defect. Intractable theories have problems with efficiency. Approximation cures intractability transforming the initial theory into an incorrect theory, which suffers from problems of accuracy. Abstraction cures intractability by transforming the initial theory into an incomplete theory, which suffers from problems of scope or precision. This analysis suggests a revision to the original intractable theory problem definition. (See Figure 1-3 in Section 1.2.5.) An alternate formulation would add scope and precision constraints to the list of explicit learning goals. A learning system could then use abstractions to trade scope or precision for efficiency, just as *POLLYANNA* uses approximations to trade accuracy for efficiency. Interesting interactions between approximation and abstraction might well be observed in such a learning system.

Figure 6-4: Solutions to the Intractable Theory Problem

6.9. Summary of Contributions

6.9.1. An Approach to the Intractable Theory Problem

A powerful new method of attacking the intractable theory problem has been demonstrated by this research. The technique is based on a strategy of sacrificing accuracy in order to gain efficiency in return. *Generic simplifying assumptions (GSAs)* are used to transform initially intractable theories into approximate theories of greater efficiency. The *POLLYANNA* system implements this strategy using a generate and test architecture. During the analytic generation phase, a space of candidate approximate theories results from systematic application of *GSAs* to the initially intractable theory. During the empirical testing phase, candidate approximate theories are evaluated against teacher-provided training examples. Measurements of accuracy and efficiency are used to guide a search through the space of approximate theories. The search is directed at finding theories meeting specified accuracy and efficiency goals.

6.9.2. A Family of Generic Simplifying Assumptions

The power of a particular family of generic simplifying assumptions was demonstrated by experiments with the *POLLYANNA* implementation. *The Probability Theory Generic Simplifying Assumptions (PT-GSAs)* were defined to include assumptions of *Function Invariance Equiprobable Random Variables* and *Probabilistic Independence*. The *PT-GSAs* were seen to provide an operational implementation of the Principle of Insufficient Reason. A variety of plausible heuristics were generated by applying *PT-GSAs* to computationally intractable theories. Generated heuristics were shown to be operationally usable by executing them on a real machine. The generic nature of *PT-GSAs* was demonstrated as well. *PT-GSAs* were used to generate heuristics for the card game hearts and for a job scheduling domain. The *PT-GSAs* were thus shown to possess generality across application domains.

6.9.3. A New Type of Learning Problem

A qualitatively new type of learning problem has been revealed by the research embodied in *POLLYANNA*. Most empirical learning research has focused on improving the *accuracy* of computational theories. More recent analytic learning research has focused on improving the problem solving *efficiency* of theories. Both types of research focus on a single dimension along which a computational theory can evaluated. In contrast to previous studies, this research has investigated a *tradeoff* between the two dimensions of accuracy and efficiency. Learning is not viewed as a process of absolute improvement in the quality of a computational theory. Learning in *POLLYANNA* involves a sacrifice of one dimension in favor of another dimension. The learning process results in a whole collection of theories that Pareto optimal along the dimensions of accuracy and efficiency. The collection provides a range of options permitting a flexible tradeoff between these two dimensions. *POLLYANNA* also acquires data that quantifies the terms under which accuracy can be traded for efficiency, and visa versa. This data is itself a significant product of learning. It may be viewed as a type of meta-level knowledge that described properties of approximate theories rather than properties of the domain. The quantitative tradeoff data can be used to select an approximate theory that balances the competing goals of accuracy and efficiency in a manner best suited to the intended performance context.

6.9.4. A Style of Learning

A novel style of machine learning is demonstrated by *POLLYANNA*. Approximate theory formation is treated as a problem of allocating limited computational resources. Analytic methods are used to hierarchically decompose computations into subproblems. Empirical methods are used to learn the difference between critical and non-critical subproblems. Relatively accurate but computationally expensive methods are used to solve critical subproblems. Relatively approximate but computationally efficient methods are used to solve non-critical subproblems. Computational resources are therefore allocated away from non-critical subproblems and toward more critical ones. *POLLYANNA* learns intelligent ways of allocating limited computational resources.

References

[Andrews 83] Andrews, J., D. *Win at Hearts.* Dover Publications, New York, New York, 1983.

[Angluin and Smith 83] Angluin, D. and Smith, C. H. ‘‘Inductive Inference: Theory and Methods.’’ *Computing Surveys 15*, 3, 1983, pp. 237 - 269.

[Balzer et al. 76] Balzer, R. N., Goldman, and Wile, D. On the Transformational Implementation Approach to Programming. Proceedings of the 2nd International Conference on Software Engineering, IEEE, 1976, pp. 337-343".

[Barstow 85] Barstow, D., R. ‘‘Domain Specific Automatic Programming.’’ *IEEE Transactions on Software Engineering 11*, 11, 1985, pp. 1321 - 1336.

[Bennett 87] Bennett, S. W. Approximation in Mathematical Domains. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1987.

[Bennett 89] Bennett, S. W. Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains. Technical Report UILU-ENG-89-2204, University of Illinois, Beckman Institute for Advanced Science and Technology, Urbana-Champaign, IL, 1989. Masters Thesis.

[Berliner and Ebeling 89] Berliner, H. and Ebeling, C. ‘‘Pattern Knowledge and Search: The SUPREM Architecture.’’ *Artificial Intelligence 38*, 2, 1989, pp. 161 - 198.

[Bird 80] Bird, R. ‘‘Tabulation Techniques for Recursive Programs.’’ *Computing Surveys 12*, 4, 1980, pp. 403 - 413.

[Bobrow 85] Bobrow, D. G., Ed. *Qualitative Reasoning about Physical Systems.* MIT Press, Cambridge, MA, 1985. Also found in Artificial Intelligence, Volume 24.

[Burstall and Darlington 77] R. M. Burstall and J. Darlington. ‘‘A transformational system for developing recursive programs.’’ *Journal of the ACM 24*, 1, January 1977, pp. 44-67.

[Carbonell 86] Carbonell, J. G. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell, Ed., *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, Los Altos, CA, 1986, pp. 371 - 392.

[Cardelli 85] Luca Cardelli and Peter Wegner. ‘‘On Understanding Types, Data Abstraction, and Polymorphism.’’ *ACM Computing Surveys 17*, 4, December 1985, pp. 471-522.

[Chase et al. 89] Chase, M., Zweben, M., Piazza, R., Burger, J., Maglio, P., Hirsh, H. Approximating Learned Search Control Knowledge. Proceedings of the Sixth International Conference on Machine Learning, Morgan Kaufmann, 1989.

[Chien 87] Chien, S. A. Extending Explanation-Based Learning: Failure-Driven Schema Refinement. Technical Report UILU-ENG-87-2203, University of Illinois, Coordinated Science Laboratory, Urbana-Champaign, IL, 1987.

[Chien 89] Chien, S. A. Using and Refining Simplifications: Explanation-Based Learning of Plans in Intractable Domains. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989.

[Cohen and Feigenbaum 82] Cohen, P. R. and Feigenbaum, E. A. (Eds.). *The Handbook of Artificial Intelligence, Volume 3.* William Kaufmann, Inc., Los Altos, CA, 1982.

[Danyluk 89] Danyluk, A. P. A Survey of Machine Learning Systems Integrating Explanation-Based and Similarity-Based Methods. Technical Report CUCS-467-89, Columbia University Department of Computer Science, 1989.

[Davis and Patterson 75] Davis, E. and Patterson, J. ‘‘A Comparison of Heuristic and Optimal Solutions in Resource Constrained Project Scheduling.’’ *AI Magazine 21*, 8, 1975, pp. 944 - 955.

[de Kleer and Brown 85] de Kleer, J. and Brown J. A Qualitative Physics Based on Confluences. In Bobrow, D. G., Ed., *Qualitative Reasoning about Physical Systems*, MIT Press, Cambridge, MA, 1985, pp. 7 - 83. Also found in Artificial Intelligence, Volume 24.

[DeJong and Mooney 86] DeJong, G. and Mooney, R. ‘‘Explanation-Based Learning: An Alternative View.’’ *Machine Learning 1*, 2, 1986, pp. 145 - 176.

[Dietterich 86] Dietterich, T. G. ‘‘Learning at the Knowledge Level.’’ *Machine Learning 1*, 3, 1986, pp. 287 - 315.

[Dietterich and Bennett 88] Dietterich, T. G. and Bennett, J. S. Varieties of Operationality. Technical Report 88-30-6, Department of Computer Science, Oregon State University, Corvallis, Oregon, 1988.

[Doyle 79] Doyle, J. ‘‘A Truth Maintenance System.’’ *Artificial Intelligence 12*, 1979, pp. 231 - 272.

[Doyle 88] Doyle, R. Hypothesizing Device Mechanisms: Opening Up the Black Box. Technical Report 1047, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1988. PhD Thesis.

[Ellman 87] Ellman, T. P. Explanation-Based Methods for Simplifying Intractable Theories: A Thesis Proposal. Technical Report CUCS-265-87, Columbia University, New York, NY, 1987.

[Ellman 88] Ellman, T. Approximate Theory Formation: An Explanation-Based Approach. Proceedings of the Seventh National Conference on Artificial Intelligence, Morgan Kaufmann, 1988.

[Ellman 89a] Ellman, T. ‘‘Explanation-Based Learning: A Survey of Programs and Perspectives.’’ *Computing Surveys 21*, 2, June 1989, pp. 163 - 221.

[Ellman 89b] Ellman, T. P. Mechanical Generation of Heuristics for Intractable Theories. Technical Report CUCS-400-88, Columbia University, New York, NY, 1989.

[Ellman 89c] Ellman, T. Mechanical Generation of Heuristic Theories: Trading Accuracy for Efficiency. Proceedings of the AAAI Spring Symposium on AI and Limited Rationality, 1989.

[Fikes et al. 72] Fikes, R. E., Hart, P. E., and Nilsson, N. J. ‘‘Learning and Executing Generalized Robot Plans.’’ *Artificial Intelligence 3*, 1972, pp. 251 - 288.

[Flann and Dietterich 86] Flann, N. S. and Dietterich, T. G. Selecting Appropriate Representations for Learning from Examples. Proceedings of the Fifth National Conference on Artificial Intelligence, Morgan Kaufmann, 1986.

[Fox et al. 89] Fox, M., Sadeh, N. and Baykan, C. Constrained Heuristic Search. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989.

[Garey and Johnson 79] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, New York, 1979.

[Gaschnig 79] Gaschnig, J. A Problem Similarity Approach to Devising Heuristics: First Results. Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 1979.

[Gibson 74] Gibson, W. B. *Hoyle’s Modern Encyclopedia of Card Games.* Doubleday and Company., Garden City, NY, 1974.

[Guida and Somalvico 79] Guida, G. and Somalvico, M. ‘‘A Method for Computing Heuristics in Problem Solving.’’ *Information Sciences 19*, 1, 1979, pp. 251 - 259.

[Gupta 87] Gupta, A. Explanation-Based Failure Recovery. Proceedings of the Sixth National Conference on Artificial Intelligence, Morgan Kaufmann, 1987.

[Hammond et al. 88] Hammond, K., Converse, T. and Marks, M. Learning from Opportunities: Storing and Re-using Execution-Time Optimizations. Proceedings of the Seventh National Conference on Artificial Intelligence, Morgan Kaufmann, 1988.

[Jaynes 83a] Jaynes, E. T. Where do We Stand on Maximum Entropy? In R. D. Rosenkrantz, Ed., *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, D. Reidel Publishing Company, Boston, Massachusetts, 1983, pp. 210 - 314.

[Jaynes 83b] Jaynes, E. T. Information Theory and Statistical Mechanics. In R. D. Rosenkrantz, Ed., *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, D. Reidel Publishing Company, Boston, Massachusetts, 1983, pp. 8 - 18.

[Jaynes 83c] Jaynes, E. T. Prior Probabilities. In R. D. Rosenkrantz, Ed., *E. T. Jaynes: Papers on Probability, Statistics and Statistical Physics*, D. Reidel Publishing Company, Boston, Massachusetts, 1983, pp. 114 - 130.

[Keller 83] Keller, R. M. Learning by Re-expressing Concepts for Efficient Recognition. Proceedings of the National Conference on Artificial Intelligence, Morgan Kaufmann, 1983, pp. 182-186.

[Keller 87] Keller, R. M. The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance. Technical Report ML-TR-7, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1987. PhD Thesis.

[Keller 88] Keller, R. M. ‘‘Defining Operationality for Explanation-Based Learning.’’ *Artificial Intelligence 35*, 1988, pp. 227 - 241.

[Kibler 85] Kibler, D. Natural Generation of Heuristics by Transforming the the Problem Representation. Technical Report TR-85-20, University of California, Irvine, CA, 1985.

[Knoblock 89] Knoblock, C. Learning Hierarchies of Abstraction Spaces. Proceedings of the Sixth International Conference on Machine Learning, Morgan Kaufmann, 1989.

[Korf 87] Korf, R. E. ‘‘Planning as Search: A Quantitative Approach.’’ *Artificial Intelligence 33*, 1, 1987, pp. 65 - 88.

[Kronsjo 79] Kronsjo, Lydia *Algorithms: Their Complexity and Efficiency.* John, Wiley and Sons, New York, New York, 1979.

[Kuhn 70] Kuhn, T. *The Structure of Scientific Revolutions.* The University of Chicago Press, Chicago, IL, 1970.

[Laird et al. 86] Laird, J. E., Rosenbloom, P. S. and Newell, A. ‘‘Chunking in SOAR: The Anatomy of a General Learning Mechanism.’’ *Machine Learning 1*, 1, 1986, pp. 11 - 46.

[Laird, et al. 87] Laird, J. E., Newell, A. and Rosenbloom, P. S. ‘‘SOAR: Architecture for General Intelligence.’’ *Artificial Intelligence 33*, 1987, pp. 1 - 64.

[Langley 88] Langley, P. ‘‘Machine Learning as an Experimental Science.’’ *Machine Learning 3*, 1, 1988, pp. 5-8.

[Lebowitz 83] Lebowitz, M. ‘‘Generalization from Natural Language Text.’’ *Cognitive Science 7*, 1, 1983, pp. 1 - 40.

[Lewis and Papadimitriou 81] Lewis, H. R. and Papadimitriou, C. H. *Elements of the Theory of Computation.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.

[Luce and Raiffa 57] Luce, R. D. and Raiffa, H. *Games and Decisions.* John Wiley and Sons, New York, NY, 1957.

[Michalski 80] Michalski, R. S. ‘‘Pattern Recognition as Rule-Guided Inductive Inference.’’ *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 4, 1980, pp. 349 - 361.

[Michalski 83] Michalski, R. S. ‘‘A Theory and Methodology of Inductive Learning.’’ *Artificial Intelligence 20*, 1983, pp. 111 - 161.

[Michalski et al. 83] Michalski, R. S., Carbonell, J. G. and Mitchell, T. M., eds. *Machine Learning, An Artificial Intelligence Approach.* Morgan Kaufmann, Los Altos, CA, 1983.

[Minton 88] Minton, S. Learning Effective Search Control Knowledge: An Explanation-Based Approach. Technical Report CMU-CS-88-133, Department of Computer Science, Carnegie-Mellon University , Pittsburgh, PA, 1988. PhD Thesis.

[Mitchell 80] Mitchell, T. M. The Need for Biases in Learning Generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1980.

[Mitchell 82] Mitchell, T. M. ''Generalization as Search.'' *Artificial Intelligence 18*, 1982, pp. 203 - 226.

[Mitchell 83] Mitchell, T. M. Learning and Problem Solving. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1983.

[Mitchell et al. 86] Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T. ''Explanation-Based Learning: A Unifying View.'' *Machine Learning 1*, 1, 1986, pp. 47 - 80.

[Mohan and Tong 89] Mohan, S. and Tong. C. Automatic Construction of a Hierarchical Generate and Test Algorithm. Proceedings of the Sixth International Conference on Machine Learning, Morgan Kaufmann, 1989.

[Mostow 81] Mostow, D. J. Mechanical Transformation of Task Heuristics into Operational Procedures. Technical Report CMU-CS-81-113, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, 1981. PhD Thesis.

[Mostow 83] Mostow, J. A Problem Solver for Making Advice Operational. Proceedings of the National Conference on Artificial Intelligence, Morgan Kaufmann, 1983, pp. 279-283.

[Mostow 89] Mostow, J. ''Design by Derivational Analogy: Issues in the Automated Replay of Design Plans.'' *Artificial Intelligence 40*, 1989, pp. 119 - 184.

[Mostow and Cohen 85] Mostow, J. and Cohen, D. Automating Program Speedup by Deciding What to Cache. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1985.

[Mostow and Fawcett 87] Mostow, J. and Fawcett, T. Approximating Intractable Theories: A Problem Space Model. Technical Report ML-TR-16, Rutgers University, Department of Computer Science, New Brunswick, NJ, 1987.

[Mostow and Prieditis 89] Mostow, J. and Prieditis, A. E. Discovering Admissible Heuristics by Abstracting and Optimizing. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989, pp. 701 - 707.

[Natarajan and Tadepalli 88] Natarajan, B. K. and Tadepalli, P. Two New Frameworks for Learning. Proceedings of the Fifth International Conference on Machine Learning, Morgan Kaufmann, 1988.

[Newell 82] Newell, A. ''The Knowledge Level.'' *Artificial Intelligence 18*, 1982, pp. 87 - 127.

[Newell and Simon 72] Newell, A. and Simon, H. *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Nilsson 80] Nilsson, N. J. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA, 1980.

[Papadimitriou and Steiglitz 82] Papadimitriou, C. H. and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[Pearl 84] Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[Polya 57] Polya, G. *How to Solve it.* Doubleday Anchor Books, New York, NY, 1957.

[Rajamoney and DeJong 87] Rajamoney, S. and DeJong, G. The Classification, Detection and Handling of Imperfect Theory Problems. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1987.

[Ralston and Rabinowitz 78] Ralston, A. and Rabinowitz, P. *A First Course in Numerical Analysis.* McGraw-Hill Book Company, New York, New York, 1978.

[Russell and Grosof 87] Russell, S. J. and Grosof, B. N. A Declarative Approach to Bias in Concept Learning. Proceedings of the Sixth National Conference on Artificial Intelligence, Morgan Kaufmann, 1987.

[Sacerdoti 74] Sacerdoti, E. D. ‘‘Planning in a Hierarchy of Abstraction Spaces.’’ *Artificial Intelligence 5*, 1974, pp. 115 - 135.

[Savitch, et al. 87] Savitch, W. J., et al, (Eds.) *The Formal Complexity of Natural Language.* Kluwer Academic Publishers, Norwell, MA, 1987.

[Segre 88] Segre, A. M. Operationality and Real World Plans. Proceedings of the AAAI Spring Symposium on Explanation-Based Learning, AAAI, 1988.

[Shannon 48] Shannon, C. E. ‘‘A Mathematical Theory of Communication.’’ *The Bell System Technical Journal 27*, 3, 1948, pp. 379 - 423.

[Smith 88] D. R. Smith. KIDS: A Knowledge-Based Software Development System. Proceedings of the AAAI88 Workshop on Automated Software Design, American Association for Artificial Intelligence, August, 1988.

[Smith et al. 86] Smith, S., Fox, M. and Ow, P. ‘‘Construction and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.’’ *AI Magazine 7*, 4, 1986, pp. 45 - 61.

[Stefik and Bobrow 86] Stefik, M. and Bobrow, D. ‘‘Object-Oriented Programming: Themes and Variations.’’ *AI Magazine 6*, 4, 1986, pp. 40 - 62.

[Stokey 89] Stokey, R. ‘‘AI Factory Scheduling: Multiple Problem Formulations.’’ *SIGART Newsletter 1*, 110, October 1989, pp. 27 - 30.

[Subramanian and Smith 88] Subramanian, D. and Smith, D. Knowledge Level Learning: An Alternative View. Proceedings of the AAAI Spring Symposium on Explanation-Based Learning, AAAI, 1988.

[Swartout 83] Swartout, W. ‘‘XPLAIN: A System for Creating and Explaining Expert Consulting Systems.’’ *Artificial Intelligence 21*, 1983, pp. 285 - 325.

[Tadepalli 89] Tadepalli, P. Lazy Explanation-Based Learning: A Solution to the Intractable Theory Problem. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989.

[Tsotsos 87] Tsotsos, J. K. A ’Complexity Level’ Analysis of Vision. Proceedings of the First International Conference on Computer Vision, IEEE Computer Society, 1987.

[Ullman 82] Ullman, J. D. *Principles of Database Systems.* Computer Science Press, Rockville, Maryland, 1982.

[Unruh and Rosenbloom 89] Unruh, A. and Rosenbloom, P. Abstraction in Problem Solving and Planning. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989.

[Utgoff 86] Utgoff, P. E. Shift of Bias for Inductive Concept Learning. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell, Ed., *Machine Learning: An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, Los Altos, CA, 1986, pp. 107-148.

[Valiant 84] Valiant, L. G. ‘‘A Theory of the Learnable.’’ *Communications of the ACM 27*, 1984, pp. 1134 - 1142.

[Winston 72] Winston, P. H. Learning Structural Descriptions From Examples. In P. H. Winston, Ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, NY, 1972.

Appendix A

Derivations of Two Hearts Heuristics

The following derivations illustrate how two hearts heuristics can be derived from an intractable hearts domain theory by using generic simplifying assumptions. All the assumptions in these derivations are members of the *PT-GSA* family of generic simplifying assumptions: *Functional Invariance (FI)*, *Equiprobable Random Variables (EP)* and *Probabilistic Independence (IN)*. These derivations thus illustrate the power of *PT-GSAs* for generating heuristics. Neither derivation has been mechanized. They were constructed by hand for illustration purposes only. An actual trace of an implemented derivation is shown in Appendix I.

Heuristic Number 1:

```
Dump High Rank Rule:

                   IF:   ALL LEGAL CARD CHOICES LOSE
                         THE CURRENT TRICK,

                   THEN: PLAY A LEGAL CARD OF HIGHEST RANK.
```

Derivation:

1. The player chooses a legal card that minimizes his expected score. He uses an evaluation function that returns his expected score at the end of the game, given that he plays card *c* in the current trick:

```
Choice = Minimize(Legal-Cards, Exp-Game-Score)

Exp-Game-Score(c) =

 = Current-Score
   + Exp-Current-Trick-Score(c)
   + ∑ (t in Tricks-Left) Exp-Future-Trick-Score(t,c)
```

2. The player's expected score on trick *t* can be written in terms of the probability of winning the trick and the expected value of the trick. This requires an assumption of probabilistic independence:

```
Exp-Future-Trick-Score(t,c) = Prob-Win(t,c) * Exp-Trick-Value(t,c)
```

IN Assumption: The probability of winning the trick is statistically independent of the trick value.

3. To compute the expected value of a trick, the player simply uses an average value, treating the function as a constant:

```
Exp-Trick-Value(t,c) = TOTAL-DECK-VALUE / NUMBER-OF-TRICKS
```

FI Assumption: The expected trick value is a constant, equal to the average value of a trick.

4. The probability of winning a trick can be expressed as a summation over all the cards that might be played and the range of possible lead suits. For each combination, the player must determine whether he wins the trick:

```
Prob-Win(t,c) = ∑ (c1,c2,c3,c4 in DECK, s in suits)
                   Prob-Trick(c1,c2,c3,c4,s,t,c)
                   * [If Win(c1,c2,c3,c4,s) then 1 else 0]
```

5. The player must now compute the odds that he will play c1, his opponents will play c2, c3 and c4 and that the lead suit will be s. Using an assumption of probabilistic independence, this may be written as a product of individual probabilities:

```
Prob-Trick(c1,c2,c3,c4,s,t,c) = Prob-Choice(me,c1,t,c)
                                * Prob-Choice(left(me),c2,t,c)
                                * Prob-Choice(right(me),c3,t,c)
                                * Prob-Choice(across(me),c4,t,c)
                                * Prob-Lead-Suit(s,t,c)
```

IN Assumption: The four card choices and the lead suit are statistically independent.

6. The player determines whether his card c1 wins the trick:

```
Win(c1,c2,c3,c4,s) = Defeats(c1,c2,s) ∧ Defeats(c1,c3,s)
                     ∧ Defeats(c1,c4,s)
```

7. Determining whether card c1 defeats card c2 when the lead suit is s:

```
Defeats(c1,c2,s) = If (Suit(c1) = s) ∧
                      ∧ [(Suit(c2) ≠ s) ∨ (Rank(c1) > Rank(c2))]
                      then True else False
```

8. In order for the player to compute the odds that he himself will play card k in trick t, he makes a simplifying assumption:

```
Prob-Choice(me,k,t,c)  = If Member(k,Hand(me) - {c})
                            then 1 / |Hand(me) - {c}|
                            else 0
```

EP Assumption: The player is equally likely to play any card remaining in his hand.

9. The probability that an opponent p will play card k in trick t:

```
Prob-Choice(p,k,t,c) = 1 / |DECK|  (If p is an opponent.)
```

EP Assumption: An opponent is equally likely to play any card in the deck.

10. The probability that s is the lead suit for trick t:

```
Prob-Lead-Suit(s,t,c) = 1 / |SUITS|
```

EP Assumption: The lead suit is equally likely to be any suit.

11. Combining the definitions of Prob-Trick (5), Prob-Choice (8,9) and Prob-Lead-Suit (10), the following expression results:

```
Prob-Trick(c1,c2,c3,c4,s,t,c) = Ka * If Member(c1,Hand(me) - {c})
                                        then 1 / |Hand(me) - {c}|
                                        else 0
```

Ka = 1 / [|SUITS| * $|DECK|^3$]

12. Substituting this definition of Prob-Trick (11) into the definition of Prob-Win (4), the result is:

```
Prob-Win(t,c) = Σ (c1 in DECK)
                  Σ (c2,c3,c4 in DECK, s in SUITS)
                     Ka * [If Win(c1,c2,c3,c4,s) then 1 else 0]
                     * [If Member(c1,Hand(me) - {c})
                           then 1 / |Hand(me) - {c}|
                           else 0]
```

13. The summation bounds in equation (12) can be changed so that c1 ranges over |Hand(me) - {c}| and s ranges over {Suit(c1)}, since all other terms are zero:

```
Prob-Win(t,c) = Σ (c1 in Hand(me) - {c})
                  Σ (c2,c3,c4 in DECK)
                     [If Win(c1,c2,c3,c4,Suit(c1) then 1 else 0]
                     * [Ka / |Hand(me) - {c}|]
```

14. The only non-zero summation terms in (13) are those for which c2, c3 and c4 are all either of lower rank than c1 or from a suit other than Suit(c1). The number of cards of lower rank is: Rank(c1) - Minimum-Rank. The number of cards in other suits is: |DECK|* [|SUITS| - 1] / |SUITS|. Equation (13) therefore reduces to:

```
Prob-Win(t,c) = [Ka / |Hand(me) - {c}|] *
               * Σ (k in Hand(me) - {c}) [Rank(k) + Kb]^3

Kb = {|DECK|* [|SUITS| - 1] / |SUITS|} - Minimum-Rank
```

15. Substituting (14) and (3) into (2):

```
Exp-Future-Trick-Score(t,c)

 = [TOTAL-DECK-VALUE / NUMBER-OF-TRICKS]
   * [Ka / |Hand(me) - {c}|]
   * Σ (k in Hand(me)-{c}) [Rank(k) + Kb]^3
```

16. Substitution (15) into (1):

```
Exp-Game-Score(c) =

 = Current-Score
   + Exp-Current-Trick-Score(c)
   + K1 * Σ (k in Hand(me) - {c}) [Rank(k) + K2]^3

K1 = |Tricks-Left| * [TOTAL-DECK-VALUE / NUMBER-OF-TRICKS]
     *  {1 / [|SUITS| * |DECK|^3] * |Hand(me) - {c}|]}

K2 = {|DECK| * [|SUITS| - 1] / |SUITS|} - Minimum-Rank
```

When all legal cards lose the current trick, Exp-Current-Trick-Score(c) is identically zero for all legal cards c. In such situations, Exp-Game-Score(c) reduces to a summation over the unplayed cards Hand(me) - {c}. Each term has the form $(\text{Rank}(k) + K2)^3$. The summation is minimized when a maximal rank card is removed from the hand.

Heuristic Number 2:

```
Dump High Point Value Rule:

                 IF:   ALL LEGAL CARD CHOICES LOSE
                       THE CURRENT TRICK,

                 THEN: PLAY A LEGAL CARD OF HIGHEST POINT VALUE.
```

Derivation:

1. The player chooses a legal card that minimizes his expected score. He uses an evaluation function that returns his expected score at the end of the game, given that he plays card *c* in the current trick:

```
Choice = Minimize(Legal-Cards, Exp-Game-Score)

Exp-Game-Score(c) =

 = Current-Score
   + Exp-Current-Trick-Score(c)
   + ∑ (t in Tricks-Left) Exp-Future-Trick-Score(t,c)
```

2. The player's expected score on trick *t* can be written in terms of the probability of winning the trick and the expected value of the trick. This requires an assumption of probabilistic independence:

```
Exp-Future-Trick-Score(t,c) = Prob-Win(t,c) * Exp-Trick-Value(t,c)
```

IN Assumption: The probability of winning the trick is statistically independent of the trick value.

3. To compute the probability of winning of a trick, the player simply uses an average value, treating the function as a constant:

```
Prob-Win(t,c) = 1 / NUMBER-OF-PLAYERS
```

FI Assumption: The odds of winning the trick are a constant equal to the average for all players.

4. The expected value of trick can be expressed as a summation over all combinations of cards that might be played:

```
Exp-Trick-Value(t,c) = ∑ (c1,c2,c3,c4 in |DECK|, s in |SUITS|)
                          Prob-Trick(c1,c2,c3,c4,s,t,c)
                          * {Point-Value(c1)+Point-Value(c2)
                             +Point-Value(c3)+Point-Value(c4)}
```

5. The player must now compute the odds that he will play c1, his opponents will play c2, c3 and c4 and that the lead suit will be s. Using an assumption of probabilistic independence, this may be written as a product of individual probabilities:

```
Prob-Trick(c1,c2,c3,c4,s,t,c) = Prob-Choice(me,c1,t,c)
                                * Prob-Choice(left(me),c2,t,c)
                                * Prob-Choice(right(me),c3,t,c)
                                * Prob-Choice(across(me),c4,t,c)
                                * Prob-Lead-Suit(s,t,c)
```

IN Assumption: The four card choices and the lead suit are statistically independent.

6. In order for the player to compute the odds that he himself will play card k in trick t, he makes a simplifying assumption:

```
Prob-Choice(me,k,t,c)  = If Member(k,Hand(me) - {c})
                            then 1 / |Hand(me) - {c}|
                            else 0
```

EP Assumption: The player is equally likely to play any card remaining in his hand.

7. The probability that an opponent p will play card k in trick t:

```
Prob-Choice(p,k,t,c) = 1 / |DECK|  (If p is an opponent.)
```

EP Assumption: An opponent is equally likely to play any card in the deck.

8. The probability that s is the lead suit for trick t:

```
Prob-Lead-Suit(s,t,c) = 1 / |SUITS|
```

EP Assumption: The lead suit is equally likely to be any suit.

9. Combining the definitions of Prob-Trick (5), Prob-Choice (6,7) and Prob-Lead-Suit (8), the following expression results:

```
Prob-Trick(c1,c2,c3,c4,s,t,c) = Ka * If Member(c1,Hand(me) - {c})
                                        then 1 / |Hand(me) - {c}|
                                        else 0

Ka = 1 / [|SUITS| * |DECK|^3]
```

10. Substituting this definition of Prob-Trick (9) into the definition of Exp-Trick-Value (4), the result is:

```
Exp-Trick-Value(t,c) = ∑ (c1 in DECK)
                         ∑ (c2,c3,c4 in DECK, s in SUITS)
                           Ka * [If Member(c1,Hand(me) - {c})
                                    then 1 / |Hand(me) - {c}|
                                    else 0]
                           * {Point-Value(c1)+Point-Value(c2)
                              +Point-Value(c3)+Point-Value(c4)}
```

11. The summation bounds in equation (9) can be changed so that c1 ranges over |Hand(me) - {c}|. The summation over suits can be replaces by a factor of |SUITS|:

```
Exp-Trick-Value(t,c) = ∑ (c1 in Hand(me) - {c})
                         ∑ (c2,c3,c4 in DECK)
                           [Ka / |Hand(me) - {c}|] * |SUITS|
                           * {Point-Value(c1)+Point-Value(c2)
                              +Point-Value(c3)+Point-Value(c4)}
```

12. Evaluating the summations over c2, c3 and c4 in (10):

```
Exp-Trick-Value(t,c) =

 = ∑ (k in Hand(me) - {c})
    [Ka / |Hand(me) - {c}|] * |SUITS|
    * [|DECK|^3 * Point-Value(k) + Kb]

Kb =  3 * |DECK^2| * ∑ (k in DECK) Point-Value(k)
```

13. Simplifying equation (11):

```
Exp-Trick-Value(t,c) =

 = Kc + Kd * ∑ (k in Hand(me) - {c}) Point-Value(k)

Kc = Ka * Kb * |SUITS|
```

```
Kd = Ka * |DECK|^3 * |SUITS| / |Hand(me) - {c}|
```

14. Substituting (12) and (3) into (2):

```
Exp-Future-Trick-Score(t,c) =

 = [1 / NUMBER-OF-PLAYERS]
   * [Kc + Kd * ∑ (k in Hand(me) - {c}) Point-Value(k)]
```

15. Substitution (13) into (1):

```
Exp-Game-Score(c) =
 = Current-Score
   + Exp-Current-Trick-Score(c)
   + K1 + K2 * ∑ (k in Hand(me) - {c}) Point-Value(k)

K1 = |Tricks-Left|
     * 3 * [1 / |DECK|] * ∑ (k in DECK) Point-Value(k)
     * [1 / NUMBER-OF-PLAYERS]

K2 = |Tricks-Left| * [1 / |Hand(me) - {c}|]
     * [1 / NUMBER-OF-PLAYERS]
```

When all legal cards lose the current trick, Exp-Current-Trick-Score(c) is identically zero for all legal cards c. In such situations, Exp-Game-Score(c) is proportional to the sum of the values of the cards in Hand(me) - {c}, i.e., the total value of the cards remaining in the player's hand. This quantity is minimal when the choice card c is a card of maximal point value.

Appendix B

Goal Heuristics for Hearts

B.1. A Heuristic Rule Set

The following heuristic rule set was used as a benchmark in the course of developing *POLLYANNA*. The benchmark served two distinct purposes. The heuristics were used partly to develop the *PT-GSA* family of generic simplifying assumptions. These *GSAs* were developed through a process of reconstructing heuristic rules in the benchmark rule set. The benchmark serves also to evaluate the performance of *POLLYANNA*.

These rules implement a complete strategy for playing hearts. Each card choice is made to satisfy the primary objective of minimizing the number of points taken during the current trick. In order to optimize the current trick, they first pare down the choices to those cards guaranteed to take zero points in the current trick. If no such card is available, they play cards of minimal point value. Among cards of equal point value, they choose cards of minimal rank. If at least some cards are guaranteed to take zero points in the current trick, the rules optimize for future tricks, by dumping dangerous cards. The first priority is to dump cards of high point value. The second priority is to dump cards of high rank. The rules are written assuming a version of the game in which the only point cards are hearts. All other cards have zero point value.

Conditions:

- L: You're the leader.
- VLS: You're void in the lead suit.
- HH: You have hearts in your hand.
- HOH: You have only hearts in your hand.
- LP: You're the last player for the current trick.
- HS: At least one heart has been played in the current trick.
- PU: You have at least one card in the lead suit which is less than some card in the lead suit which has already been played in the current trick.

Actions:

- PHH: Play your highest heart.
- PHC: Play your highest card of any suit.

- PLC: Play your lowest card of any suit.
- PHLS: Play your highest card in the lead suit.
- PHU: Play your highest card in the lead suit which is less than the highest card in the lead suit which was already played in this trick.
- PLLS: Play your lowest card in the lead suit.
- PLNH: Play your lowest card which is not a heart.

Rules:

- (L ∧ HOH) → PLC
- (L ∧ ¬HOH) → PLNH
- (¬L ∧ VLS ∧ HH) → PHH
- (¬L ∧ VLS ∧ ¬HH) → PHC
- (¬L ∧ ¬VLS ∧ LP ∧ ¬HS) → PHLS
- (¬L ∧ ¬VLS ∧ LP ∧ HS ∧ PU) → PHU
- (¬L ∧ ¬VLS ∧ LP ∧ HS ∧ ¬PU) → PHLS
- (¬L ∧ ¬VLS ∧ ¬LP ∧ PU) → PHU
- (¬L ∧ ¬VLS ∧ ¬LP ∧ ¬PU) → PLLS

B.2. *POLLYANNA*'s Coverage of the Heuristic Rule Set

The following table compares the card playing behavior of the heuristic rule set to *POLLYANNA*'s approximate hearts theory *T8*, shown in Figure 3-8. Since the rules apply to a version in which hearts are the only point cards, the approximate theory must be tailored to this version as well. The function *Point-Value* must obviously be changed. The only other modification involves the numerical constants appearing in Figure 3-5. The average point value of a card *B2* is equal to *1/4* for this version of the game. The following table shows that eight of the nine rules are consistent with approximate theory *T8*. The inconsistency occurs when a player is leading and he holds both hearts and non-hearts. The rules recommend the lowest non-heart. The theory may sometimes prefer a low heart to a high non-heart.

Consistent:	**Rule:**
Yes	**(L ∧ HOH) → PLC**
No	**(L ∧ ¬HOH) → PLNH**
Yes	**(¬L ∧ VLS ∧ HH) → PHH**
Yes	**(¬L ∧ VLS ∧ ¬HH) → PHC**
Yes	**(¬L ∧ ¬VLS ∧ LP ∧ ¬HS) → PHLS**
Yes	**(¬L ∧ ¬VLS ∧ LP ∧ HS ∧ PU) → PHU**
Yes	**(¬L ∧ ¬VLS ∧ LP ∧ HS ∧ ¬PU) → PHLS**
Yes	**(¬L ∧ ¬VLS ∧ ¬LP ∧ PU) → PHU**
Yes	**(¬L ∧ ¬VLS ∧ ¬LP ∧ ¬PU) → PLLS**

Appendix C

The Initial Intractable Hearts Theory

The hearts theory defines a function *Choice(p,tr,legals,private,public)* that determines which card player *p* should choose to play in trick *tr*. The choice function is given the list *legals* of allowable card choices, along with parameters *private* and *public,* which describe current game state. The variable *public* describes the game history that is visible to all players. The variable *private* contains a description of player *p*'s own hand. Both *private* and *public* are represented as *givens* that can be used to compute conditional expectation values or probabilities. (See Appendix E).

The evaluation function *Exp-Game-Score(c,p,tr,private,public)* computes the expectation value of λ*(d)Game-Score(p,t,d)* given the state that results when player *p* plays card *c* in the current game state. The *given* condition is computed by the function *Next-State(c,p,tr,private,public)*. The expected game score is implemented as a sum of four parts: *Exp-Past-Score*, *Exp-Current-Score*, *Exp-Current-Score*, and *Exp-Future-Score*. This extensional representation facilitates the process of separately approximating the portions of the hearts theory that analyze the past, current, next and future tricks. Each of the four functions *Exp-Past-Score*, *Exp-Current-Score*, *Exp-Current-Score*, and *Exp-Future-Score* directly computes a conditional expectation value of *Trick-Score* for the corresponding trick, or group of tricks.

Some hearts functions are left undefined in the following section, e.g., the functions *deals, private, past-tricks, future-tricks, prior-state-trick, prior-state-player and initial-leader.* These functions were never used in any derivations of approximate hearts theories. Nor are they referenced at run time by any derived approximate hearts theory. The presence or absence of these functions has no impact on any of the experimental results. For this reason, their definitions have been omitted.

The function definitions contain a number of constructs that were defined in LISP especially for *POLLYANNA*, e.g., *lambda*, +plus+, -minus-, *times*, /div/, andd, iff, orr, equall,* etc. These are all semantically equivalent to the corresponding LISP functions. (Note the convention of repeating the last letter of boolean functions.) The special versions were defined so that efficiency measurement demons could be associated with these low level operations.

```
(define choice
 (lambda* (p tr legals private public)
  (minimize (lambda* (c)
              (exp-game-score c p tr private public))
            legals)))

(define exp-game-score
  (lambda* (c p tr private public)
     (+plus+ (exp-past-score c p tr private public)
        (exp-current-score c p tr private public)
        (exp-next-score c p tr private public)
        (exp-future-score c p tr private public))))

(define exp-past-score
  (lambda* (c p tr private public)
    (exp (lambda* (d) (past-score p tr d))
                 (next-state c p tr private public))))

(define exp-current-score
  (lambda* (c p tr private public)
    (exp (lambda* (d) (current-score p tr d))
                 (next-state c p tr private public))))

(define exp-next-score
  (lambda* (c p tr private public)
    (exp (lambda* (d) (next-score p tr d))
                 (next-state c p tr private public))))

(define exp-future-score
  (lambda* (c p tr private public)
    (exp (lambda* (d) (future-score p tr d))
                 (next-state c p tr private public))))

(define next-state
  (lambda* (c p tr private public)
    (and-givens (update-state c p tr public)
                private)))

(define past-score
  (lambda* (p tr d)
    (sum (past-tricks tr)
         (lambda* (x) (trick-score p x d)))))

(define current-score
  (lambda* (p tr d) (trick-score p tr d)))

(define next-score
  (lambda* (p tr d) (trick-score p (following-trick tr) d)))

(define future-score
  (lambda* (p tr d)
    (sum (future-tricks tr)
         (lambda* (x) (trick-score p x d)))))

(define minimize
 (lambda* (fn set)
  (random-member
```

```
    (set-of set
            (lambda* (x)
                (equall (applyy fn (list x))
                       (minn (mapcar fn set))))))))

(define legal-choices
 (lambda* (p tr d)
   (set-of (deck)
           (lambda* (c)
              (andd (equall (hand c d) p)
                   (unplayed c p tr d)
                   (legal c p tr d))))))

(define legal
 (lambda* (c p tr d)
  (orr (equall p (leader tr d))
      (void p tr (lead-suit tr d) d)
      (equall (suit c) (lead-suit pt d)))))

(define void
 (lambda* (p tr s d)
   (nott (exists (deck)
                 (lambda* (c)
                     (andd (equall (suit c) s)
                          (equall (hand c d) p)
                          (unplayed c p tr d)))))))

(define trick-score
 (lambda* (p tr d)
  (*times* (win p tr d) (trick-value p tr d))))

(define win
 (lambda* (p tr d)
  (product (opponents p)
           (lambda* (op)
            (defeats (card p tr d) (card op tr d) (lead-suit tr d))))))

(define trick-value
 (lambda* (p tr d)
  (+plus+ (point-value (card p tr d))
     (sum (opponents p)
          (lambda* (pp) (point-value (card pp tr d)))))))

(define defeats
 (lambda* (c1 c2 s)
    (*times*  (iff (equall (suit c1) s) 1 0)
              (iff (orr (nott (equall (suit c2) (suit c1)))
                        (>gt> (rank-value (rank c1))
                              (rank-value (rank c2))))
                   1
                   0))))

(define card
  (lambda* (p tr d)
    (choice p
            tr
            (legal-choices p tr d)
```

```
            (private p d)
            (public p tr d))))

(define public
 (lambda* (p tr d)
  (iff (andd (equall tr (initial-trick))
           (equall p (initial-leader)))
      (null-given)
      (update-state (card p tr d)
                    p
                    tr
                    (public (prior-state-player p tr d)
                            (prior-state-trick p tr d))))))

(define update-state
  (lambda* (c p tr public)
    (and-givens (given (lambda* (d) (card p tr d)) c)
                public)))

(define unplayed
 (lambda* (c p tr d)
  (iff (equall tr (initial-trick))
      t
      (iff (equall (card p (prior-trick tr) d) c)
          nil
          (unplayed c p (prior-trick tr) d)))))

(define point-value
 (lambda* (c)
  (iff (equall (suit c) ’hearts)
      1
      (iff (andd (equall (suit c) ’spades)
               (equall (rank c) ’q))
          13
          0))))

(define lead-suit
 (lambda* (tr d)
  (suit (card (leader tr d) tr d))))

(define leader
 (lambda* (tr d)
  (iff (initial-trick tr)
      (initial-leader)
      (winner (prior-trick tr) d))))

(define winner
 (lambda* (tr d)
  (the-unique (players)
              (lambda* (p) (win p tr d)))))

(define players
  (lambda* () ’(tom dick harry moe)))

(define opponents
  (lambda* (p) (delete-element p (players))))

(define tricks
```

```
  (lambda* () '(1 2 3 4 5 6 7 8 9 10 11 12 13)))

(define hand
  (lambda* (c d) (assocc c d)))

(define prior-trick (lambda* (tr)  (-minus- tr 1)))

(define following-trick (lambda* (tr)  (+plus+ tr 1)))

(define initial-trick (lambda* () 1))

(define deck
  (lambda* ()
    '(
       (CD 2 SPADES) (CD 2 HEARTS) (CD 2 DIAMONDS) (CD 2 CLUBS)
       (CD 3 SPADES) (CD 3 HEARTS) (CD 3 DIAMONDS) (CD 3 CLUBS)
       (CD 4 SPADES) (CD 4 HEARTS) (CD 4 DIAMONDS) (CD 4 CLUBS)
       (CD 5 SPADES) (CD 5 HEARTS) (CD 5 DIAMONDS) (CD 5 CLUBS)
       (CD 6 SPADES) (CD 6 HEARTS) (CD 6 DIAMONDS) (CD 6 CLUBS)
       (CD 7 SPADES) (CD 7 HEARTS) (CD 7 DIAMONDS) (CD 7 CLUBS)
       (CD 8 SPADES) (CD 8 HEARTS) (CD 8 DIAMONDS) (CD 8 CLUBS)
       (CD 9 SPADES) (CD 9 HEARTS) (CD 9 DIAMONDS) (CD 9 CLUBS)
       (CD 10 SPADES) (CD 10 HEARTS) (CD 10 DIAMONDS) (CD 10 CLUBS)
       (CD J SPADES) (CD J HEARTS) (CD J DIAMONDS) (CD J CLUBS)
       (CD Q SPADES) (CD Q HEARTS) (CD Q DIAMONDS) (CD Q CLUBS)
       (CD K SPADES) (CD K HEARTS) (CD K DIAMONDS) (CD K CLUBS)
       (CD A SPADES) (CD A HEARTS) (CD A DIAMONDS) (CD A CLUBS)
     )))

(define suit (lambda* (c) (caddrr c)))

(define rank (lambda* (c) (cadrr c)))

(define rank-value
  (lambda* (r)
    (cadrr
     (assocc r
      '( (1  1) (2  2) (3  3) (4  4) (5  5) (6  6)
         (7  7) (8  8) (9  9) (10  10) (J 11) (Q 12)
         (K 13) (A 14))))))

(define range
 (lambda* (f)
  (cadrr
   (assocc f '(
    (card   (
     (CD 2 SPADES) (CD 2 HEARTS) (CD 2 DIAMONDS) (CD 2 CLUBS)
     (CD 3 SPADES) (CD 3 HEARTS) (CD 3 DIAMONDS) (CD 3 CLUBS)
     (CD 4 SPADES) (CD 4 HEARTS) (CD 4 DIAMONDS) (CD 4 CLUBS)
     (CD 5 SPADES) (CD 5 HEARTS) (CD 5 DIAMONDS) (CD 5 CLUBS)
     (CD 6 SPADES) (CD 6 HEARTS) (CD 6 DIAMONDS) (CD 6 CLUBS)
     (CD 7 SPADES) (CD 7 HEARTS) (CD 7 DIAMONDS) (CD 7 CLUBS)
     (CD 8 SPADES) (CD 8 HEARTS) (CD 8 DIAMONDS) (CD 8 CLUBS)
     (CD 9 SPADES) (CD 9 HEARTS) (CD 9 DIAMONDS) (CD 9 CLUBS)
     (CD 10 SPADES) (CD 10 HEARTS) (CD 10 DIAMONDS) (CD 10 CLUBS)
     (CD J SPADES) (CD J HEARTS) (CD J DIAMONDS) (CD J CLUBS)
     (CD Q SPADES) (CD Q HEARTS) (CD Q DIAMONDS) (CD Q CLUBS)
```

```
 (CD K SPADES) (CD K HEARTS) (CD K DIAMONDS) (CD K CLUBS)
 (CD A SPADES) (CD A HEARTS) (CD A DIAMONDS) (CD A CLUBS)
 ))
(lead-suit           (spades hearts diamonds clubs))
(defeats             (0 1))
(win                 (0 1))
(point-value         (0 1 13))
(trick-value         (0 1 2 3 4 13 14 15 16))
(trick-score         (0 1 2 3 4 13 14 15 16))
(cardinality         (0  1  2  3  4  5  6  7  8  9
                      10 11 12 13 14 15 16 17 18 19
                      20 21 22 23 24 25 26 27 28 29
                      30 31 32 33 34 35 36 37 38 39
                      40 41 42 43 44 45 46 47 48 49
                      51 52))
(hand                (tom dick harry moe))
(equall              (t nil))
(unplayed            (t nil)))))))


;;(define deals () ...)
;;(define private (p d) ...)
;;(define past-tricks (tr) ...)
;;(define future-tricks (tr) ...)
;;(define prior-state-trick (p tr d) ...)
;;(define prior-state-player (p tr d) ...)
;;(define initial-leader () ...)
```

Appendix D

The Initial Intractable Scheduling Theory

The scheduling theory defines a function *Evaluation(instance,time,given)* which can be used to evaluate partially specified schedules. The *instance* parameter describes a particular scheduling problem, i.e., a set of jobs, deadlines, precedence constraints, etc. The *time* parameter represents the first free *time* slot in the partially specified schedule. The *given* parameter represents the assignments of jobs to time slots that have already been made in the partially specified schedule. *Evaluation* operates by minimizing the expected value of the final schedule over all possible completions of the partially specified schedule.

An exact evaluation results from using the first definition of *Evaluation* to look ahead to the leaves of the search tree and directly evaluate fully specified schedules. Fully specified schedules are evaluated by computing *Exp[λ(e)Value(i,e)|g]* where *i* is the problem instance and *g* represents the fully specified schedule. Since all jobs are assigned in a fully specified schedule, this expectation value is just the actual value of the schedule. Approximations that look ahead exactly *d* levels can also be obtained. They result from unfolding *Evaluation* to some depth *d* using the first definition of this function. The second definition of *Evaluation* is then used to compute the expected value of the partially specified schedule. In such an intermediate state, the values of jobs already assigned can be directly determined. The values of jobs not yet assigned must be estimated using assumptions of *Equiprobable Random Variables (EP)*, *Functional Invariance (FI)* and *Probabilistic Independence (IN)*. Some scheduling functions are left undefined in the following section because they were never used in any derivations of approximate scheduling theories.

```
(define evaluation
  (lambda* (instance tm g)
    (iff (final-state? instance tm g)
         (exp (lambda* (e) (value instance e)) g)
         (minimize (options tm g)
                   (lambda* (op)
                     (evaluation instance
                                 (next-time tm)
                                 (next-state op tm g)))))))

(define evaluation
  (lambda* (instance tm g)
     (exp (lambda* (e) (value instance  e)) g)))
```

```
(define next-time
  (lambda* (instance tm op)
    (+plus+ tm 1)))

(define next-state
   (lambda* (op tm g)
       (and-givens g
                   (given (lambda* (e) (allocation-time op e)) tm)
                   (given (lambda* (e) (choice tm e)) op))))

(define value
   (lambda* (instance e)
      (sum (jobs instance)
           (lambda* (j) (job-value j instance e)))))

(define job-value
   (lambda* (j instance e)
      (*times* (preconditions j instance e)
               (*times* (penalty j
                                 instance
                                 (completion-time j instance e))
                        (weight j instance)))))

(define completion-time
  (lambda* (j instance e)
    (+plus+ (allocation-time j e)
            (working-time j instance))))


(define allocation-time
  (lambda* (j e)
     (the-unique (times instance)
                 (lambda* (tm) (equall (choice tm e) j)))))

(define choice
  (lambda* (tm e)
     (the-unique (jobs instance)
                 (lambda* (j) (equall (allocation-time j e) tm)))))


(define preconditions
   (lambda* (j instance e)
      (product (requirements j instance)
               (lambda* (jj)
                 (*times*
                   (iff (<le< (completion-time jj instance e)
                              (allocation-time j e))
                        1
                        0)
                   (preconditions jj instance e))))))

(define penalty
   (lambda* (j instance time)
     (iff (<le< time (deadline j instance))
          1
          0)))

(define options
```

```
    (lambda* (instance tm g)
      (if (job-finishes instance tm g)
          (set-of (jobs instance)
                  (lambda* (j)
                   (nott (job-scheduled instance j tm g))))
          (list 'NIL))))

(define job-scheduled
  (lambda* (instance j tm g)
    (exists (times instance)
            (lambda* (tm)
               (andd (known (lambda* (e) (choice tm e)) g)
                     (equall j (evaluate (lambda* (e) (choice tm e))
                             g)))))))

(define job-finishes
  (lambda* (instance tm g)
    (exists (times instance)
            (lambda* (tmm)
              (known (lambda* (e) (choice tm e)) g)
              (equall (+plus+ tmm
                              (working-time (evaluate
                                              (lambda* (e)
                                                (choice tm e))
                                              g)
                                             instance))
                      tm)))))

;;(define final-state?
;;  (lambda* (instance tm g) ...))

;;(define deadline
;;  (lambda* (j instance) ...))

;;(define requirements
;;  (lambda* (j instance) ...))

;;(define penalty-factor
;;   (lambda* (j instance) ...))

;;(define working-time
;;  (lambda* (j instance) ...))

;;(define weight
;;  (lambda* (j instance) ...))


;;(define range
;; (lambda* (f)
;;  (cadrr
;;   (assocc f '(
;;    (preconditions (0 1))
;;    (allocation-time (0 1 2 3 4 5 6 ....?..... ))
;;    (completion-time (0 1 2 3 4 5 6 ....?..... ))
;;    (penalty (0 1))
;;    (weight  (1 2 3 4 ....?.... )))))))
```

Appendix E

Primitive Functions for the *PT-Formalism*

Computation of Probabilities and Expectation Values:

- *Prob[λ(e) Boolean(e) | Given].*
- *Exp[λ(e) Numeric(e) | Given].*

The notation *Exp[λ(e)N(e)]|G]* signifies the expectation value of some numeric function *N(e),* viewed as a function of the event space. Likewise *Prob[λ(e)B(e)]|G]* is the probability that a boolean function *B(e)* is true. Both *Prob* and *Exp* have a second argument indicating the *givens* of a conditional expectation or probability. A *given* structure is conceptually a list of pairs. Each pair associates a random variable with a list of possible values. The random variable is constrained to take on one of the values on the list. Random variables not appearing on the list are free to take any value in their ranges. A given structure thus specifies a subset of the underlying event space. The subset consists of all events consistent with the constraints on values of random variables that are contained in the given structure. The given structures are actually implemented with hash tables that map λ-expressions representing random variables to lists of possible values.

Operations on Given Structures:

- *Known[λ(e) Function(e) | Given].*
- *Evaluate[λ(e) Function(e) | Given].*
- *Given[λ(e) f(e), v].*
- *And-Givens(G1,G2).*
- *Null-Given[].*

The function *Known* takes a random variable in the form of a λ-expression as its first argument, and a *given* structure as its second argument. It checks to see whether the random variable's value is directly determined by the given structure. The function *Evaluate* is the same as *Known* except that *Evaluate* returns the value of the random variable if the value is determined. The function *Evaluate* is not defined when the value of the random variable is not determined by the *given* structure. The function *Given* takes a random variable as its first argument and a value as its second argument. It returns a *given* structure that constrains the random value to have the specified value. The function *And-Givens* returns the logical conjunction of two *given* structures.

Reduction of Binary Operations:

- *Sum[Set, λ(x) Numeric(x)]*
- *Product[Set, λ(x) Numeric(x)]*

Operations on Sets Represented as Lists:

- *Minimize[Set, λ(x) Numeric(x)]*
- *Set-of[Set, λ(x) Boolean(x)]*
- *The-Unique[Set, λ(x) Boolean(x)]*
- *Delete-Element(Element, Set)*
- *Average(Numeric-Set)*
- *Cardinality(Set)*
- *Random-Member(Set)*

Quantification over Sets:

- *Exists[Set, λ(x) Boolean(x)]*
- *For-All[Set, λ(x) Boolean(x)]*

Standard LISP Functions:

- Arithmetic Functions: +, -, *, /.
- Relational Functions: <, ≤, >, ≥.
- Boolean Functions: *If, And, Or, Not, Equal.*
- List Operations: *List, Cons, Car, Cdr, Minimum, Member, Assoc.*
- Constants: *NIL, 0,1,...,N,..., 'ATOM, '(...).*
- Second Order Functions: *Apply, Mapcar*.

Appendix F

GSA Operators

A total of 10 operators are used to implement the *PT-GSA* family in *POLLYANNA*. The *FI* assumptions are implemented with three operators: The operator *Function-Invariance:Constant* implements *FI* version #1 shown in Figure 2-13. It uses a procedural implementation to query the user for a constant to replace an arbitrary expression or sub-expression. The operators *Function-Invariance:Sum* and *Function-Invariance:Product* respectively implement *FI* version #3 and *FI* version #4 in Figure 2-13. The *EP* assumptions are implemented by the three operators *Equiprobable-Boolean*, *Equiprobable-Equality* and *Equiprobable-Numeric*. These correspond directly to the three versions of *EP* shown in Figure 2-13. The *IN* assumptions are implemented by four operators. Two operators are used for each of the two versions of *IN* shown in Figure 2-13. *IN* version #1 is implemented by *Independence:Two-Conjuncts* and *Independence:Three-Conjuncts*. *IN* version #2 is implemented by *Independence-of-Product:Binary* and *Independence-of-Product:Nary*. These operators handle cases that differ in the number of conjuncts in a conjunctive expression, or the number of factors in product expression.

```
GENERIC OPERATOR DEFINITION:

LHS: A pattern describing the expression to be replaced.
RHS: A pattern describing the replacement expression.

Pattern Variables:
(? x)  Matches Exactly One S-Expression.
(*s x) Matches Zero or More S-Expressions.

NO-OCCURRENCES: (......( EXP-1 EXP-2 ).......)
 Indicates that the instantiation of EXP-2 may not appear anywhere
 within the instantiation of EXP-2. This constraint is used to keep
 Lambda variables within their proper scope.

...................................................................

EP-OPERATORS:

Equiprobable:Boolean:

:LHS ’(PROB (? B) (? GIVEN))

:RHS ’(/div/ 1 2)

Equiprobable:Equality:
```

```
:LHS '(PROB (LAMBDA* ((? D))
              (EQUALL  ((? F) (*s ARGS)) (? V)))
            (? G))

:RHS '(/div/ 1 (CARDINALITY (RANGE '(? F))))
```

Equiprobable:Numeric:

```
:LHS '(EXP (LAMBDA* ((? D)) ((? F) (*s ARGS))) (? GIVEN))

:RHS '(AVERAGE (RANGE '(? F)))
```

....

IN-OPERATORS.

Independence-of-Product:Binary:

```
:LHS '(EXP (LAMBDA* ((? D)) (*TIMES* (? G) (? H))) (? GIVEN))

:RHS '(*TIMES* (EXP (LAMBDA* ((? D)) (? G)) (? GIVEN))
               (EXP (LAMBDA* ((? D)) (? H)) (? GIVEN)))
```

Independence-of-Product:Nary:

```
:LHS '(EXP (LAMBDA* ((? D))
              (PRODUCT (? S) (LAMBDA* ((? V)) (? F))))
           (? GIVEN))

:RHS '(PRODUCT (? S)
               (LAMBDA* ((? V))
                 (EXP (LAMBDA* ((? D)) (? F)) (? GIVEN))))
```

Independence:Two-Conjuncts:

```
:LHS '(PROB (LAMBDA* ((? D)) (ANDD (? F) (? G))) (? GIVEN))

:RHS '(*TIMES* (PROB (LAMBDA* ((? D)) (? F)) (? GIVEN))
               (PROB (LAMBDA* ((? D)) (? G)) (? GIVEN)))
```

Independence:Three-Conjuncts:

```
:LHS '(PROB (LAMBDA* ((? D)) (ANDD (? F) (? G) (? H))) (? GIVEN))

:RHS '(*TIMES* (PROB (LAMBDA* ((? D)) (? F)) (? GIVEN))
               (PROB (LAMBDA* ((? D)) (? G)) (? GIVEN))
               (PROB (LAMBDA* ((? D)) (? H)) (? GIVEN)))
```

....

FI OPERATORS:

Function-Invariance: Sum:

```
:LHS '(+plus+ (*s TS1) (? Term) (*s TS2))

:RHS '(+plus+ (*s TS1) 0 (*s TS2))
```

Function-Invariance: Product:

```
:LHS '(*times* (*s FS1) (? Factor) (*s FS2))

:RHS '(*times* (*s FS1) 1 (*s FS2))
```

Function-Invariance:Constant:

Given an expression E, replace E with a constant value supplied by the interactive user. This operator has a purely procedural implementation. It does not use the generic operator format defined above. The procedure simply prompts the interactive user for a suitable constant.

Appendix G

Reformulation Operators

A total of 29 reformulation operators were implemented in *POLLYANNA*. These include 15 implementing laws of probability theory; 6 implementing identities of algebra; 2 implementing rules of logic; and 1 implementing a theorem of set theory. *Fold* and *UnFold* were implemented using 3 and 2 different operators each. A procedural implementation was used to implement *Fold* and *UnFold*. All the other reformulation operators were implemented as declarative rewrite rules. *Memoization* was implemented by defining demons to perform the tasks of table storage and lookup. The demons get invoked prior to selected function calls. The approximation generator interface allows the user to indicate which function/versions should be memoized.

```
GENERIC OPERATOR DEFINITION:

LHS: A pattern describing the expression to be replaced.
RHS: A pattern describing the replacement expression.

Pattern Variables:
(? x)  Matches Exactly One S-Expression.
(*s x) Matches Zero or More S-Expressions.

NO-OCCURRENCES: (......( EXP-1 EXP-2 ).......)
 Indicates that the instantiation of EXP-2 may not appear anywhere
 within the instantiation of EXP-2. This constraint is used to keep
 Lambda variables within their proper scope.

...................................................................

EXP-SUM-OPERATORS:

Expectation-of-SUM:Binary:

:LHS '(EXP (LAMBDA* ((? D)) (+PLUS+ (? X) (? Y)))
           (? GIVEN))

:RHS '(+PLUS+ (EXP (LAMBDA* ((? D)) (? X)) (? GIVEN))
              (EXP (LAMBDA* ((? D)) (? Y)) (? GIVEN)))

Expectation-of-SUM:Nary:

:LHS '(EXP (LAMBDA* ((? D))
              (SUM (? SET) (LAMBDA* ((? X)) (? BODY))))
           (? GIVEN))
```

```
:RHS '(SUM (? SET)
           (LAMBDA* ((? X))
              (EXP (LAMBDA* ((? D)) (? BODY))
                   (? GIVEN))))
```

....

COMPOSITE PROBABILITIES AND EXPECTATIONS:

Composed-Exp: One Arg:

```
:LHS '(EXP (LAMBDA* ((? D))
             ((? F) (*s a1) ((? G) (*s ARGS)) (*s a2)))
           (? GIVEN))

:RHS '(SUM (RANGE '(? G))
        (LAMBDA* (V1)
          (*TIMES* ((? F) (*s a1) V1 (*s a2))
                   (PROB (LAMBDA* ((? D))
                            (EQUALL  ((? G) (*s ARGS)) V1))
                         (? GIVEN)))))

:no-occurrences '( ((? D) (*s a1)) ((? D) (*s a2)) )
```

Composed-Exp: Two Args:

```
:LHS '(EXP (LAMBDA* ((? D)) ((? F) (*s a1)
                                   ((? G) (*s GS))
                                   (*s a2)
                                   ((? H) (*s HS))
                                   (*s a3)))
           (? GIVEN))

:RHS '(SUM (RANGE '(? G))
        (LAMBDA* (V1)
          (SUM (RANGE '(? H))
            (LAMBDA* (V2)
              (*TIMES* ((? F) (*s a1) v1 (*s a2) v2 (*s a3))
                       (PROB (LAMBDA* ((? D))
                               (ANDD (EQUALL  ((? G) (*s GS)) V1)
                                     (EQUALL  ((? H) (*s HS)) V2)))
                             (? GIVEN)))))))

:no-occurrences '( ((? D) (*s a1)) ((? D) (*s a2)) ((? D) (*s a3)) )
```

Composed-Exp: Three Args:

```
:LHS '(EXP (LAMBDA* ((? D)) ((? F) (*s a1)
                                   ((? G) (*s GS))
                                   (*s a2)
                                   ((? H) (*s HS))
                                   (*s a3)
                                   ((? I) (*s IS))
                                   (*s a4)))
           (? GIVEN))

:RHS '(SUM (RANGE '(? G))
```

```
        (LAMBDA* (V1)
          (SUM (RANGE '(? H))
            (LAMBDA* (V2)
              (SUM (RANGE '(? I))
                (LAMBDA* (V3)
                  (*TIMES* ((? F) (*s a1)
                                  v1
                                  (*s a2)
                                  v2
                                  (*s a3)
                                  v3
                                  (*s a4))
                           (PROB (LAMBDA* ((? D))
                                   (ANDD (EQUALL  ((? G) (*s GS)) V1)
                                         (EQUALL  ((? H) (*s HS)) V2)
                                         (EQUALL  ((? I) (*s IS)) V3)))
                                 (? GIVEN)))))))))

:no-occurrences '( ((? D) (*s a1))
                   ((? D) (*s a2))
                   ((? D) (*s a3))
                   ((? D) (*s a4)) )

Composed-Prob: One Arg:

:LHS '(PROB (LAMBDA* ((? D))
              (EQUALL  ((? F) (*s a1) ((? G) (*s GS)) (*s a2))
                       (? V)))
            (? GIVEN))

:RHS '(SUM (RANGE '(? G))
        (LAMBDA* (V1)
          (*TIMES* (IFF (EQUALL  ((? F) (*s a1) v1 (*s a2))
                                 (? V))
                        1
                        0)
                   (PROB (LAMBDA* ((? D)) (EQUALL  ((? G) (*s GS)) V1))
                         (? GIVEN)))))

:no-occurrences '( ((? D) (*s a1)) ((? D) (*s a2))  )

Composed-Prob: Two Args:

:LHS '(PROB (LAMBDA* ((? D))
              (EQUALL  ((? F) (*s a1)
                              ((? G) (*s GS))
                              (*s a2)
                              ((? H) (*s HS))
                              (*s a3))
                       (? V)))
            (? GIVEN))

:RHS '(SUM (RANGE '(? G))
        (LAMBDA* (V1)
          (SUM (RANGE '(? H))
            (LAMBDA* (V2)
              (*TIMES* (IFF (EQUALL  ((? F) (*s a1)
```

```
                                             v1
                                             (*s a2)
                                             v2
                                             (*s a3))
                                      (? V))
                             1
                             0)
                        (PROB (LAMBDA* ((? D))
                                (ANDD (EQUALL  ((? G) (*s GS)) V1)
                                      (EQUALL  ((? H) (*s HS)) V2)))
                              (? GIVEN)))))))

:no-occurrences '( ((? D) (*s a1)) ((? D) (*s a2)) ((? D) (*s a3)) )

Composed-Prob: Three Args:

:LHS '(PROB (LAMBDA* ((? D))
              (EQUALL  ((? F) (*s a1)
                               ((? G) (*s GS))
                               (*s a2)
                               ((? H) (*s HS))
                               (*s a3)
                               ((? I) (*s IS))
                               (*s a4))
                        (? V)))
            (? GIVEN))

:RHS '(SUM (RANGE '(? G))
        (LAMBDA* (V1)
          (SUM (RANGE '(? H))
            (LAMBDA* (V2)
              (SUM (RANGE '(? I))
                (LAMBDA* (V3)
                 (*TIMES* (IFF (EQUALL  ((? F) (*s a1)
                                               V1
                                               (*s a2)
                                               V2
                                               (*s a3)
                                               V3
                                               (*s a4))
                                         (? V))
                                  1
                                  0)
                           (PROB (LAMBDA* ((? D))
                                   (ANDD (EQUALL  ((? G) (*s GS)) V1)
                                         (EQUALL  ((? H) (*s HS)) V2)
                                         (EQUALL  ((? I) (*s IS)) V3)))
                                 (? GIVEN)))))))))

:no-occurrences '( ((? D) (*s a1))
                   ((? D) (*s a2))
                   ((? D) (*s a3))
                   ((? D) (*s a4))  )

...................................................................

OPERATORS FOR FACTORING SUMMATIONS:
```

```
Factor-Summation:

:LHS '(SUM (? S)
        (LAMBDA* ((? V))
           (*times* (*s FS1) (? F) (*s FS2))))

:RHS '(*TIMES* (? F)
               (SUM (? S)
                 (LAMBDA* ((? V))
                   (*times* (*s FS1) (*s FS2)))))

:no-occurrences '( ((? V) (? F)) )

Reverse-Summation-Order:

:LHS '(SUM (? S1)
           (LAMBDA* ((? V1))
                    (SUM (? S2)
                         (LAMBDA* ((? V2)) (? EXP)))))

:RHS '(SUM (? S2)
           (LAMBDA* ((? V2))
                    (SUM (? S1)
                         (LAMBDA* ((? V1)) (? EXP)))))

...................................................................

MANIPULATION OF COMMUTATIVE OPERATIONS:

Commute-Summation:

:LHS '(+plus+ (*s TS1) (? T1) (*s TS2) (? T2) (*s TS3))

:RHS '(+plus+ (*s TS1) (? T2) (*s TS2) (? T1) (*s TS3))

Commute-Product:

:LHS '(*times* (*s TS1) (? T1) (*s TS2) (? T2) (*s TS3))

:RHS '(*times* (*s TS1) (? T2) (*s TS2) (? T1) (*s TS3))

Commute-Conjunction:

:LHS '(andd (*s TS1) (? T1) (*s TS2) (? T2) (*s TS3))

:RHS '(andd (*s TS1) (? T2) (*s TS2) (? T1) (*s TS3))

...................................................................

MANIPULATION OF ASSOCIATIVE OPERATIONS:

Associate-Summation:

:LHS '(+plus+ (*s TS1)
              (+plus+ (*s TS2))
              (*s TS3))

:RHS '(+plus+ (*s TS1)
```

```
            (*s TS2)
            (*s TS3))
```

Associate-Product:

```
:LHS '(*times* (*s FS1)
               (*times* (*s FS2))
               (*s FS3))

:RHS '(*times* (*s FS1)
               (*s FS2)
               (*s FS3))
```

Associate-Conjunction:

```
:LHS '(ANDD (*s CS1)
            (ANDD (*s CS2))
            (*s CS3))

:RHS '(ANDD (*s CS1)
            (*s CS2)
            (*s CS3))
```

....

EQUIVALENT FORMS OF PROBABILITIES AND EXPECTATIONS:

Probability-to-Expectation:

```
:LHS '(PROB (LAMBDA* ((? D)) (? FORM)) (? GIVEN))

:RHS '(EXP (LAMBDA* ((? D)) (IFF (? FORM) 1 0))
           (? GIVEN))
```

Prob-Boolean-to-Prob-Equality:

```
:LHS '(PROB (LAMBDA* ((? D)) (? FORM)) (? GIVEN))

:RHS '(PROB (LAMBDA* ((? D)) (EQUALL  (? FORM) T))
            (? GIVEN))
```

....

PROBABILITIES IF KNOWN:

If the value of a random variable is KNOWN under a set of givens, then the probability or expectation value can be determined by EVALUATING the variable under the givens.

Probability-If-Known:Boolean:

```
:LHS '(PROB (? FOO) (? GIVEN))

:RHS '(IFF (KNOWN (? FOO) (? GIVEN))
           (IF (EVALUATE (? FOO) (? GIVEN))
               1
               0)
```

```
           (PROB (? FOO) (? GIVEN)))
```

Probability-If-Known:Equality:

```
:LHS '(PROB (LAMBDA* ((? D)) (EQUALL  (? FOO) (? V)))
            (? GIVEN))

:RHS '(IFF (KNOWN (LAMBDA* ((? D)) (? FOO)) (? GIVEN))
           (IFF (EQUALL  (EVALUATE (LAMBDA* ((? D)) (? FOO)) (? GIVEN))
                         (? V))
                1
                0)
           (PROB (LAMBDA* ((? D)) (EQUALL  (? FOO) (? V))) (? GIVEN)))
```

Expectation-If-Known:

```
:LHS '(EXP (? FOO) (? GIVEN))

:RHS '(IFF (KNOWN (? FOO) (? GIVEN))
           (EVALUATE (? FOO) (? GIVEN))
           (EXP (? FOO) (? GIVEN)))
```

....

MANIPULATION OF SETS:

Member of Set:

```
:LHS '(MEMBER (? X) (SET-OF (? S) (? TEST)))

:RHS '(ANDD (MEMBER (? X) (? S)) (APPLYY (? TEST) (LIST (? X))))
```

....

PROBABILITIES OF SET MEMBERSHIP:

Prob of Random Member:

The function (RANDOM-MEMBER R) selects a random member from the set R. A random selection from the set R of members of S that meet TEST is likely to be VALUE with probability P/Q where: P = The odds that VALUE is contained in R. Q = The expected cardinality of R given that VALUE is in R.

```
:LHS  '(PROB (LAMBDA* ((? D))
               (EQUALL  (RANDOM-MEMBER (SET-OF (? S) (? TEST)))
                        (? VALUE)))
             (? GIVEN))

:RHS  '(/DIV/ (PROB (LAMBDA* ((? D))
                      (ANDD (MEMBER (? VALUE) (? S))
                            (APPLYY (? TEST) (LIST (? VALUE)))))
                    (? GIVEN))
              (EXP (LAMBDA* ((? D))
                     (CARDINALITY (SET-OF (? S) (? TEST))))
                   (AND-GIVENS (? GIVEN)
                               (GIVEN (LAMBDA* ((? D))
```

```
                                         (MEMBER (? VALUE)
                                                 (SET-OF (? S)
                                                         (? TEST))))
                                        T))))
```

Prob of Unique Member:

The function (THE-UNIQUE S TEST) returns the unique element of S that satisfies TEST. The function is not defined when more than one element of S satisfies TEST. VALUE is the unique member of S satisfying TEST if VALUE is a member of S and VALUE satisfies TEST.

```
:LHS  '(PROB (LAMBDA* ((? D))
               (EQUALL  (THE-UNIQUE (? SET) (? TEST))
                        (? VALUE)))
             (? GIVEN))

:RHS  '(PROB (LAMBDA* ((? D))
               (ANDD (MEMBER (? VALUE) (? SET))
                     (APPLYY (? TEST) (LIST (? VALUE)))))
             (? GIVEN))
```

....

FOLD-OPERATORS:

All the fold operators behave in the following manner. Given an expression E, they rewrite the expression in the form: (F ARG1 ... ARGN) and define the function F with a lambda expression such that (F ARG1 ... ARGN) expands into expression E. For a given initial expression E, the fold operation can be performed in more than one way, depending on how the arguments ARGi are chosen. The following three versions differ on how the arguments are chosen. These operators have purely procedural implementations. They do not use the generic operator format defined above.

Fold Function Definition: Atomic Args:

Choose only atomic s-expressions as arguments.

Fold Function Definition: Selected Args:

Allow the user to interactively select the arguments to be used.

Fold Existing Function Definition:

Find a previously defined function that can expand into the given expression. If several previous definitions can be so expanded, this operation can be done in several ways.

....

UNFOLD-OPERATORS:

Unfold Function Definition:

Given an expression (F ARG1 ... ARGN), expand the expression using

the body of a definition of F, replacing the formal parameters in the definition with ARG1 ... ARGN. If function F has multiple definitions, this operation can be done in several different ways. This operator has a purely procedural implementation. It does not use the generic operator format defined above.

Unfold Lambda Definition:

Given an expression: (APPLY (LAMBDA* PARMS BODY) ARGS) replace it with the result of substituting ARGS for PARMS in BODY. This operator has a purely procedural implementation. It does not use the generic operator format defined above.

Appendix H

Outline of Automatic Control

The proposed automatic control procedure *GENERATE* is defined below. This definition is equivalent to the one given in Section 2.7; however, it includes a greater level of detail. It shows the specific *GSA* and reformulation operators that would be used at each step. It also shows the results of each operator application. The final state of the approximation generator resulting from use of *GENERATE* is also shown below.

```
GENERATE(F(x)):

0. Initially F has the form:

   F-1: F(x) = Prob[λ(e) v=h(e) |G]

1. Apply Equiprobable:Equality to F-1:

*  F-2: F(x) = 1/|Range(h)|.

2. Apply Probability-if-Known:Equality to F-1:

   F-3: F(x) = If Known[λ(e) h(e) |G]
               then [If v = Evaluate[λ(e) h(e) |G]
                        then 1
                        else 0]
               else Prob[λ(e) v=h(e) |G]

3. Apply Equiprobable:Equality to F-3:

*  F-4: F(x) = If Known[λ(e) h(e) |G]
               then [If v = Evaluate[λ(e) h(e) |G]
                        then 1
                        else 0]
               else 1/|Range(h)|


4. Apply UnFold to h(e) in F-1:

   F-5: F(x) = Prob[λ(e) v=s(t1(e),...,tn(e)) |G]

5. Apply Composed-Prob: N-Args to F-5:

   F-6: F(x) = Σ (v1 in Range(t1))
             ...
              Σ (vn in Range(tn))
                  If v=s(v1,...,vn)
```

```
                        then Prob[λ(e) v1=t1(e) ∧ ...
                                           ... ∧ vn=tn(e)
                                  |G]
                        else 0
```

6. Apply *Independence:N-Conjuncts* **to F-6:**

```
   F-7: F(x) = Σ (v1 in Range(t1))
                ...
                 Σ (vn in Range(tn))
                    If v=s(v1,...,vn)
                       then Prob[λ(e) v1=t1(e) |G] * ...
                            ... * Prob[λ(e) vn=tn(e) |G]
                       else 0
```

7. Apply *Fold* **N times in F-7:**

```
*  F-8: F(x) = Σ (v1 in Range(t1))
                ...
                 Σ (vn in Range(tn))
                    If v=s(v1,...,vn)
                       then G1(x,v1) * ... * Gn(x,vn)
                       else 0


   G1-1: G1(x,v1) = Prob[λ(e) v1=t1(e) |G]

                  ...

   Gn-1: Gn(x,vn) = Prob[λ(e) vn=tn(e) |G]
```

8. Repeat Steps 4 through 7 starting with version F-3 rather than version F-1.

```
*  F-9: F(x) = If Known[λ(e) h(e) |G]
                  then [If v = Evaluate[λ(e) h(e) |G]
                           then 1
                           else 0]
                  else Σ (v1 in Range(t1))
                        ...
                         Σ (vn in Range(tn))
                           If v=s(v1,...,vn)
                              then G1(x,v1) * ... * Gn(x,vn)
                              else 0
```

9. For (i = 1 .. N) Call GENERATE(Gi(x,vi)).

FINAL STATE:
(Selecting starred versions only.)

Version1: Return a constant.

F(x) = 1/|Range(h)|.

Version2: If *h(e)* **is known given** *G*, **return** *0* **or** *1* **depending on whether it has value** *v*. **Otherwise return** *1/|Range(h)|*.

```
F(x) = If Known[λ(e) h(e) |G]
          then [If v = Evaluate[λ(e) h(e) |G]
                   then 1
                   else 0]
          else 1/|Range(h)|
```

Version3: **Since** *h(e)* **is a function of** *t1(e),...,tn(e)*, **the odds that** *v=h(e)* **can be computed by summing over combinations of possible values of each** *ti(e)*. *Gi(x,vi)* **computes the odds that variable** *ti(e)* **has value** *vi*.

```
F(x) = Σ (v1 in Range(t1))
        ...
         Σ (vn in Range(tn))
               If v=s(v1,...,vn)
                  then G1(x,v1) * ... * Gn(x,vn)
                  else 0
```

Version4: **If** *h(e)* **is known given** *G*, **return** *0* **or** *1* **depending on whether it has value** *v*. **Otherwise the odds that** *v=h(e)* **can be computed by summing over combinations of possible values of each** *ti(e)*. *Gi(x,vi)* **computes the odds that variable** *ti(e)* **has value** *vi*.

```
F(x) = If Known[λ(e) h(e) |G]
          then [If v = Evaluate[λ(e) h(e) |G]
                   then 1
                   else 0]
          else Σ (v1 in Range(t1))
                ...
                 Σ (vn in Range(tn))
                   If v=s(v1,...,vn)
                      then G1(x,v1) * ... * Gn(x,vn)
                      else 0
```

Appendix I

Sample Approximation Generator Operation

I.1. Sample Approximation Generator Trace

The behavior of *POLLYANNA*'s approximation generator is illustrated by the following sample execution trace. The trace can best be understood by referring to the hearts random variable tree shown in Figure 2-10. The domain theory is initially expressed in terms of the random variable *Game-Score* appearing at the root of the tree. The operators *UnFold*, *Probability/Expectation of a Composite Function (PC/EC)* and *Probabilistic Independence (IN)* are used to re-express the theory in terms of variables appearing at lower points in the tree. Once the theory is expressed in terms of a given random variable, an assumption of *Equiprobable Random Variables (EP)* can be formulated for that variable. *EP* assumptions are generated for each random variable appearing above a limited depth in the tree. The execution trace thus follows a path corresponding roughly to the automatic control procedure *GENERATE*. (See Figure 2-23 in Section 2.7 and Appendix H.)

The simplified hearts theory shown in Figure 2-9 was used as the input for this run of the approximation generator. The sample trace is similar in flavor, but different in details from the trace that results from using the more complex hearts theory representation found in Appendix C. The complex version was used to generate the results presented in Chapter 3. (See Section 3.3.1.) The algebraic expressions shown here describe actual outputs of the approximation generator; however they were translated by hand from LISP into ordinary function notation.

1. The initial theory is an evaluation function computing an expectation value of the random variable *Game-Score*.

```
Exp-Game-Score(p,t,hand,state,c) =
 = Exp[λ(d) Game-Score(p,d) | Next-State(p,t,hand,state,c)]
```

2. Apply *Equiprobable Random Variables (EP)* to *Game-Score*.

```
Exp-Game-Score(p,t,hand,state,c) = Average(Range(Game-Score))
```

3. Use *UnFold* to write *Game-Score (1)* as a sum over *Trick-Score* for each trick.

```
Exp-Game-Score(p,t,hand,state,c) =
```

```
 = Exp[λ(d) ∑ (t’ in TRICKS) Trick-Score(p,t’,d)
       | Next-State(p,t,hand,state,c)]
```

4. Use *Expectation-of-Sum:Nary* to reformulate the expectation value of sum into a sum of expectation values.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t’ in TRICKS)
     Exp[λ(d) Trick-Score(p,t’,d) | Next-State(p,t,hand,state,c)]
```

5. Apply *Fold* to define the new function *Exp-Trick-Score*.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t’ in TRICKS)
     Exp-Trick-Score(p,t’,Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = Exp[λ(d) Trick-Score(p,t,d) | state]
```

6. Use *Expectation-If-Known* to insert a test checking whether *Trick-Score* is known.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t’ in TRICKS)
     Exp-Trick-Score(p,t’,Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = If Known(λ(d) Trick-Score(p,t,d) , state)
      then Evaluate(λ(d) Trick-Score(p,t,d) , state)
      else Exp[λ(d) Trick-Score(p,t,d) | state]
```

7. Apply *Equiprobable Random Variables (EP)* to *Trick-Score* after the test.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t’ in TRICKS)
     Exp-Trick-Score(p,t’,Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = If Known(λ(d) Trick-Score(p,t,d) , state)
      then Evaluate(λ(d) Trick-Score(p,t,d) , state)
      else Average(Range(Trick-Score))
```

8. Apply *UnFold* to write *Trick-Score (5)* in terms of *Win* and *Trick-Value*.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t’ in TRICKS)
     Exp-Trick-Score(p,t’,Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = Exp[λ(d) If Win(p,t,d) then Trick-Value(t,d) else 0 | state]
```

9. Apply *Expectation of Composite Function (EC)* to write *Exp-Trick-Score* as a sum over combinations of values of *Win* and *Trick-Value*.

```
Exp-Game-Score(p,t,hand,state,c) =
```

```
 = ∑ (t' in TRICKS)
     Exp-Trick-Score(p,t',Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = ∑ (v1 in Range(Win))
     ∑ (v2 in Range(Trick-Value))
       (If v1 Then v2 Else 0) *
       Prob[λ(d) (Win(p,t,d) = v1) ∧ (Trick-Value(t,d) = v2)
            | state ]
```

10. Apply *Probabilistic Independence (IN)* to *Win* and *Trick-Value*.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t' in TRICKS)
     Exp-Trick-Score(p,t',Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = ∑ (v1 in Range(Win))
     ∑ (v2 in Range(Trick-Value))
       (If v1 Then v2 Else 0)
       * Prob[λ(d) Win(p,t,d) = v1 | state]
       * Prob[λ(d) Trick-Value(t,d) = v2 | state]
```

11. Use *Fold* to define the new functions *P-Win* and *P-Trick-Value*.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t' in TRICKS)
     Exp-Trick-Score(p,t',Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = ∑ (v1 in Range(Win))
     ∑ (v2 in Range(Trick-Value))
       (If v1 Then v2 Else 0)
       * P-Win(p,t,state,v1)
       * P-Trick-Value(t,state,v2)

P-Win(p,t,state,v) = Prob[λ(d) Win(p,t,d) = v | state]

P-Trick-Value(t,state,v) = Prob[λ(d) Trick-Value(t,d) = v | state]
```

12. Use *Probability-If-Known:Equality* to insert tests to check if either of *Win* or *Trick-Value* is known. Apply *Equiprobable Random Variables (EP)* to *Win* and to *Trick-Value* after the tests.

```
Exp-Game-Score(p,t,hand,state,c) =
 = ∑ (t' in TRICKS)
     Exp-Trick-Score(p,t',Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state) =
 = ∑ (v1 in Range(Win))
     ∑ (v2 in Range(Trick-Value))
       (If v1 Then v2 Else 0)
       * P-Win(p,t,state,v1)
       * P-Trick-Value(t,state,v2)

P-Win(p,t,state,v) =
```

```
  = If Known(λ(d) Win(p,t,d)  , state)
       then {If v = Evaluate(λ(d) Win(p,t,d) , state)
                 then 1
                 else 0}
       else 1/|Range(Win)|

P-Trick-Value(t,state,v) =
  = If Known(λ(d) Trick-Value(t,d)  , state)
       then {If v = Evaluate(λ(d) Trick-Value(t,d) , state)
                 then 1
                 else 0}
       else 1/|Range(Trick-Value)|
```

13. Deriving a New Version of *P-Win*:

```
P-Win(p,t,state,v) = Prob[λ(d) Win(p,t,d) = v | state]
```

14. Use *UnFold* to write *Win* in terms of *Defeats*:

```
P-Win(p,t,state,v) =
 = Prob[λ(d) v = Defeats(Card(p,t,d),
                         Card(Left(p),t,d),
                         Lead-Suit(t,d)),
                 ∧ Defeats(Card(p,t,d),
                           Card(Across(p),t,d),
                           Lead-Suit(t,d)),
                 ∧ Defeats(Card(p,t,d),
                           Card(Right(p),t,d),
                           Lead-Suit(t,d)),
         | state]
```

15. Use *Probability of a Composite Function (PC)* to write the *P-Win* as a sum over combinations of values of *Defeats* for each of three opponents:

```
P-Win(p,t,state,v) =
 = ∑ (v1 in Range(Defeats))
    ∑ (v2 in Range(Defeats))
     ∑ (v3 in Range(Defeats))
        If v = v1 ∧ v2 ∧ v3
           then {Prob[λ(d) v1 = Defeats(Card(p,t,d),
                                        Card(Left(p),t,d),
                                        Lead-Suit(t,d))
                           ∧ v2 = Defeats(Card(p,t,d),
                                          Card(Across(p),t,d),
                                          Lead-Suit(t,d))
                           ∧ v3 = Defeats(Card(p,t,d),
                                          Card(Right(p),t,d),
                                          Lead-Suit(t,d))
                      | state]}
           else 0
```

16. Apply *Probabilistic Independence (IN)* to assume the three instances of *Defeats* are independent:

```
P-Win(p,t,state,v) =
 = ∑ (v1 in Range(Defeats))
```

```
    ∑ (v2 in Range(Defeats))
     ∑ (v3 in Range(Defeats))
        If v = v1 ∧ v2 ∧ v3
           then {Prob[λ(d) v1=Defeats(Card(p,t,d),
                                      Card(Left(p),t,d),
                                      Lead-Suit(t,d))
                      | state]
                 * Prob[λ(d) v2=Defeats(Card(p,t,d),
                                        Card(Across(p),t,d),
                                        Lead-Suit(t,d))
                      | state]
                 * Prob[λ(d) v3=Defeats(Card(p,t,d),
                                        Card(Right(p),t,d),
                                        Lead-Suit(t,d))
                      | state]}
           else 0
```

17. Use *Fold* to define the new function *P-Defeats*:

```
 P-Win(p,t,state,v) =
  = ∑ (v1 in Range(Defeats))
     ∑ (v2 in Range(Defeats))
      ∑ (v3 in Range(Defeats))
         If v = v1 ∧ v2 ∧ v3
            then {P-Defeats(p,t,state,Left(p),v1)
                  * P-Defeats(p,t,state,Across(p),v2)
                  * P-Defeats(p,t,state,Right(p),v3)}
            else 0

P-Defeats(p,t,state,p',v) =
  = Prob[λ(d) v = Defeats(Card(p,t,d),
                          Card(p',t,d),
                          Lead-Suit(t,d))
          | state]
```

18. Use *Probability-If-Known:Equality* to insert test checking whether *Defeats* is known. Apply *Equiprobable Random Variables (EP)* to *Defeats* after the test:

```
P-Defeats(p,t,state,p',v) =
  = If Known(λ(d) Defeats(Card(p,t,d),
                          Card(p',t,d),
                          Lead-Suit(t,d)),
             state)
       then {If [v = Evaluate(λ(d) Defeats(Card(p,t,d),
                                           Card(p',t,d),
                                           Lead-Suit(t,d)),
                              state)]
               then 1
               else 0}
       else 1 / |Range(Defeats)|
```

19. Deriving New Version of *P-Trick-Value*:

```
P-Trick-Value(t,state,v) = Prob[λ(d) Trick-Value(t,d) = v |state]
```

20. Use *UnFold* to write *Trick-Value* as a sum over *Point-Value* for each player:

```
P-Trick-Value(t,state,v) =
  = Prob[λ(d) v = Point-Value(Card(Tom,t,d))
                  + Point-Value(Card(Dick,t,d))
                  + Point-Value(Card(Harry,t,d))
                  + Point-Value(Card(Moe,t,d))
         | state]
```

21. Use *Probability of a Composite Function (PC)* to write the *P-Trick-Value* as a sum over combinations of values of *Point-Value* for each of player:

```
P-Trick-Value(t,state,v) =
  = ∑ (v1 in Range(Point-Value))
     ∑ (v2 in Range(Point-Value))
      ∑ (v3 in Range(Point-Value))
       ∑ (v4 in Range(Point-Value))
         If v = v1 + v2 + v3 + v4
            then Prob[λ(d) v1 = Point-Value(Card(Tom,t,d))
                           ∧ v2 = Point-Value(Card(Dick,t,d))
                           ∧ v3 = Point-Value(Card(Harry,t,d))
                           ∧ v4 = Point-Value(Card(Moe,t,d))
                      | state]
            else 0
```

22. Apply *Probabilistic Independence (IN)* to assume the four instances of *Point-Value* are independent:

```
P-Trick-Value(t,state,v) =
  = ∑ (v1 in Range(Point-Value))
     ∑ (v2 in Range(Point-Value))
      ∑ (v3 in Range(Point-Value))
       ∑ (v4 in Range(Point-Value))
         If v = v1 + v2 + v3 + v4
            then Prob[λ(d) v1=Point-Value(Card(Tom,t,d)) |state]
                 * Prob[λ(d) v2=Point-Value(Card(Dick,t,d)) |state]
                 * Prob[λ(d) v3=Point-Value(Card(Harry,t,d)) |state]
                 * Prob[λ(d) v4=Point-Value(Card(Moe,t,d)) |state]
            else 0
```

23. Use *Fold* to define the new function *P-Point-Value*:

```
P-Trick-Value(t,state,v) =
  = ∑ (v1 in Range(Point-Value))
     ∑ (v2 in Range(Point-Value))
      ∑ (v3 in Range(Point-Value))
       ∑ (v4 in Range(Point-Value))
         If v = v1 + v2 + v3 + v4
            then {P-Point-Value(Tom,t,state,v1)
                  * P-Point-Value(Dick,t,state,v2)
                  * P-Point-Value(Harry,t,state,v3)
                  * P-Point-Value(Moe,t,state,v4)}
            else 0

P-Point-Value(t,state,p,v) =
  = Prob[λ(d) v = Point-Value(Card(p,t,d)) |state]
```

24. Use *Probability-If-Known:Equality* to insert test checking whether *Point-Value* is known. Apply *Equiprobable Random Variables (EP)* to *Point-Value* after the test:

```
P-Point-Value(t,state,p,v) =
  = If Known(λ(d) Point-Value(Card(p,t,d)) , state)
       then {If [v = Evaluate(λ(d) Point-Value(Card(p,t,d)),
                                 state)]
                then 1
                else 0}
       else 1/|Range(Point-Value)|
```

I.2. Sample Approximation Generator Final State

The final state of the sample approximation generator trace is shown below. The final state described the six functions *Exp-Game-Score*, *Exp-Trick-Score*, *P-Win*, *P-Defeats*, *P-Trick-Value* and *P-Point-Value*. Each function is defined by one or more versions. Not all the functions appearing the final state are shown below For those functions shown, only some of their versions are listed. The omitted versions are all computationally intractable. They each require evaluation of expressions that directly calculate probabilities *Prob[]* or expectation values *Exp[]*.

```
FINAL STATE OF THE SAMPLE APPROXIMATION GENERATOR TRACE:

Exp-Game-Score(p,t,hand,state,c):

EGS-0: = Average(Range(Game-Score))

EGS-1: = Σ (t’ in TRICKS)
           Exp-Trick-Score(p,t’,Next-State(p,t,hand,state,c))

Exp-Trick-Score(p,t,state):

ETS-0: = If Known(λ(d) Trick-Score(p,t,d) , state)
            then Evaluate(λ(d) Trick-Score(p,t,d) , state)
            else Average(Range(Trick-Score))

ETS-1: = Σ (v1 in Range(Win))
           Σ (v2 in Range(Trick-Value))
             (If v1 Then v2 Else 0)
             * P-Win(p,t,state,v1)
             * P-Trick-Value(t,state,v2)

P-Win(p,t,state,v):

PWN-0: = If Known(λ(d) Win(p,t,d)  , state)
            then {If v = Evaluate(λ(d) Win(p,t,d) , state)
                     then 1
                     else 0}
            else 1/|Range(Win)|

PWN-1: = Σ (v1 in Range(Defeats))
          Σ (v2 in Range(Defeats))
           Σ (v3 in Range(Defeats))
             If v = v1 ∧ v2 ∧ v3
```

```
                 then {P-Defeats(p,t,state,Left(p),v1)
                       * P-Defeats(p,t,state,Across(p),v2)
                       * P-Defeats(p,t,state,Right(p),v3)}
                 else 0

P-Defeats(p,t,state,p',v):

PDF-0: = If Known(λ(d) Defeats(Card(p,t,d),
                               Card(p',t,d),
                               Lead-Suit(t,d)),
                  state)
            then {If [v = Evaluate(λ(d) Defeats(Card(p,t,d),
                                                Card(p',t,d),
                                                Lead-Suit(t,d)),
                                   state)]
                     then 1
                     else 0}
            else 1 / |Range(Defeats)|


P-Trick-Value(t,state,v):

PTV-0: = If Known(λ(d) Trick-Value(t,d)  , state)
            then {If v = Evaluate(λ(d) Trick-Value(t,d) , state)
                     then 1
                     else 0}
            else 1/|Range(Trick-Value)|

PTV-1: = ∑ (v1 in Range(Point-Value))
          ∑ (v2 in Range(Point-Value))
           ∑ (v3 in Range(Point-Value))
            ∑ (v4 in Range(Point-Value))
              If v = v1 + v2 + v3 + v4
                 then {P-Point-Value(Tom,t,state,v1)
                       * P-Point-Value(Dick,t,state,v2)
                       * P-Point-Value(Harry,t,state,v3)
                       * P-Point-Value(Moe,t,state,v4)}
                 else 0

P-Point-Value(p,t,state,v):

PPV-0: = If Known(λ(d) Point-Value(Card(p,t,d)) , state)
            then {If v = Evaluate(λ(d) Point-Value(Card(p,t,d)),
                                  state)
                     then 1
                     else 0}
            else 1/|Range(Point-Value)|
```

Appendix J

Approximation Generator Output: Hearts

A summary of one final state produced by *POLLYANNA*'s approximation generator for the hearts domain is found below. The principal functions in the final state listed in a diagram that describes referencing relationships. The diagram also indicates which functions have multiple versions, and the number of versions of each. The actual output of *POLLYANNA*'s approximation generator for the hearts domain follows after the referencing diagram. Only a partial description of the final state given. All versions computing exact probabilities or expectation values are intractable, and were therefore omitted. This data was generated using the complex hearts theory representation found in Appendix C

```
Exp-Game-Score
 |
 |----Exp-Past-Score
 |----Exp-Future-Score
 |
 |----Exp-Current-Score
 |      |----Exp-Win* [2]
 |      |      |----Exp-Defeats
 |      |             |----Prob-Known-Card
 |      |             |
 |      |             |----Exp-Defeats-Sum-Sum
 |      |                    |----Prob-Lead-Suit
 |      |                    |      |----Prob-Lead-Card* [2]
 |      |                    |
 |      |                    |----Exp-Defeats-Sum
 |      |                           |----Prob-Card-Invariant
 |      |
 |      |----Exp-Trick-Value* [2]
 |             |----Exp-Point-Value* [2]
 |             |      |----Prob-Card* [2]
 |             |
 |             |----Exp-Point-Value-Invariant
 |                    |----Prob-Card-Invariant
 |
 |----Exp-Next-Score
        |----Exp-Next-Win* [2]
        |      |----Exp-Next-Defeats
        |             |----Prob-My-Next-Card
        |             |----Prob-Next-Defeats-In-Suit-Invariant
        |             |----Prob-Next-In-Lead-Suit-Invariant
        |
        |----Exp-Next-Trick-Value* [2]
```

```
             |----Exp-My-Next-Point-Value* [2]
             |     |----Prob-My-Next-Card
             |
             |----Exp-Op-Next-Point-Value
```

```
FUNCTION:                       CODE:

Exp-Win:                        EWN-0, EWN-1
Prob-Lead-Card:                 PLC-0, PLC-1
Exp-Trick-Value:                ETV-0, ETV-1
Exp-Point-Value:                EPV-0, EPV-1
Prob-Card:                      PCD-0, PCD-1
Exp-Next-Win:                   ENW-0, ENW-1
Exp-Next-Trick-Value:           ENT-0, ENT-1
Exp-My-Next-Point-Value:        ENP-0, ENP-1
```

....

```
Function: EXP-GAME-SCORE
Id:       V8732

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (+PLUS+
          (EXP-PAST-SCORE C P TR PRIVATE PUBLIC)
          (EXP-CURRENT-SCORE C P TR PRIVATE PUBLIC)
          (EXP-NEXT-SCORE C P TR PRIVATE PUBLIC)
          (EXP-FUTURE-SCORE C P TR PRIVATE PUBLIC)))
```

....

```
Function: EXP-PAST-SCORE
Id:       V8701

(LAMBDA* (C P TR PRIVATE PUBLIC) 0)
```

....

```
Function: EXP-FUTURE-SCORE
Id:       V8699

(LAMBDA* (C P TR PRIVATE PUBLIC) 0)
```

....

```
Function: EXP-CURRENT-SCORE
Id:       V8741

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (*TIMES* (EXP-WIN C P TR PRIVATE PUBLIC)
                  (EXP-TRICK-VALUE C P TR PRIVATE PUBLIC)))
```

....

```
Function: EXP-WIN
Id:       V8754
Code:     EWN-0

(LAMBDA* (C P TR PRIVATE PUBLIC) (AVG-WIN))
```

```
Function: EXP-WIN
Id:       V8749
Code:     EWN-1

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (PRODUCT (OPPONENTS P)
                  (LAMBDA* (OP1)
                    (EXP-DEFEATS C P TR PRIVATE PUBLIC OP1))))

.......................................................................

Function: EXP-DEFEATS
Id:       V8670

(LAMBDA* (C P TR PRIVATE PUBLIC OP1)
         (SUM (RANGE 'CARD)
          (LAMBDA* (V12)
              (*TIMES* (PROB-KNOWN-CARD C P TR PRIVATE PUBLIC P V12)
               (EXP-DEFEATS-SUM-SUM C P TR PRIVATE PUBLIC OP1 V12)))))

.......................................................................

Function: PROB-KNOWN-CARD
Id:       V8685

(LAMBDA* (C P TR PRIVATE PUBLIC PP1 V11)
    (IFF (KNOWN (LAMBDA* (D) (CARD PP1 TR D))
                (NEXT-STATE C P TR PRIVATE PUBLIC))
         (IFF (EQUALL (EVALUATE (LAMBDA* (D) (CARD PP1 TR D))
                                (NEXT-STATE C P TR PRIVATE PUBLIC))
                      V11)
              1
              0)
     (P-CARD C P TR PRIVATE PUBLIC PP1 V11)))

.......................................................................

Function: P-CARD
Id:       V8759

(LAMBDA* (C P TR PRIVATE PUBLIC PP1 V11) (ONE-DIV-CARD))

.......................................................................

Function: EXP-DEFEATS-SUM-SUM
Id:       V8735

(LAMBDA* (C P TR PRIVATE PUBLIC OP1 V12)
     (SUM (RANGE 'LEAD-SUIT)
      (LAMBDA* (V31)
               (*TIMES* (PROB-LEAD-SUIT C P TR PRIVATE PUBLIC V31)
                (EXP-DEFEATS-SUM P TR PRIVATE PUBLIC OP1 V12 V31)))))

.......................................................................

Function: PROB-LEAD-SUIT
Id:       V8760
```

```
(LAMBDA* (C P TR PRIVATE PUBLIC V31)
         (SUM (RANGE 'CARD)
          (LAMBDA* (V13)
                   (*TIMES* (IFF (EQUALL (SUIT V13) V31) 1 0)
                    (PROB-LEAD-CARD C P TR PRIVATE PUBLIC V13)))))
```

....

```
Function: PROB-LEAD-CARD
Id:       V8771
Code:     PLC-0

(LAMBDA* (C P TR PRIVATE PUBLIC V13) (ONE-DIV-CARD))

Function: PROB-LEAD-CARD
Id:       V8767
Code:     PLC-1

(LAMBDA* (C P TR PRIVATE PUBLIC V13)
    (IFF
     (KNOWN (LAMBDA* (D) (CARD (LEADER TR D) TR D))
            (NEXT-STATE C P TR PRIVATE PUBLIC))
     (IFF (EQUALL (EVALUATE (LAMBDA* (D) (CARD (LEADER TR D) TR D))
                            (NEXT-STATE C P TR PRIVATE PUBLIC))
                  V13)
          1
          0)
     (P-LEAD-CARD C P TR PRIVATE PUBLIC V13)))
```

....

```
Function: P-LEAD-CARD
Id:       V8774

(LAMBDA* (C P TR PRIVATE PUBLIC V13) (ONE-DIV-CARD))
```

....

```
Function: EXP-DEFEATS-SUM
Id:       V8736

(LAMBDA* (P TR PRIVATE PUBLIC OP1 V12 V31)
         (SUM (RANGE 'CARD)
          (LAMBDA* (V21)
              (*TIMES* (DEFEATS V12 V21 V31)
               (PROB-CARD-INVARIANT P TR PRIVATE PUBLIC OP1 V21)))))
```

....

```
Function: PROB-CARD-INVARIANT
Id:       V8733

(LAMBDA* (P TR PRIVATE PUBLIC PP1 V11)
    (IFF (KNOWN (LAMBDA* (D) (CARD PP1 TR D))
                (NEXT-STATE NIL P TR PRIVATE PUBLIC))
         (IFF (EQUALL (EVALUATE (LAMBDA* (D) (CARD PP1 TR D))
                                (NEXT-STATE NIL P TR PRIVATE PUBLIC))
```

```
                      V11)
              1
              0)
     (P-CARD NIL P TR PRIVATE PUBLIC PP1 V11)))

.......................................................................

Function: EXP-TRICK-VALUE
Id:       V8659
Code:     ETV-0

(LAMBDA* (C P TR PRIVATE PUBLIC) (AVG-TRICK-VALUE))

Function: EXP-TRICK-VALUE
Id:       V8658
Code:     ETV-1

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (+PLUS+ (EXP-POINT-VALUE C P TR PRIVATE PUBLIC P)
          (SUM (OPPONENTS P)
           (LAMBDA* (PP1)
             (EXP-POINT-VALUE-INVARIANT P TR PRIVATE PUBLIC PP1)))))

.......................................................................

Function: EXP-POINT-VALUE
Id:       V8669
Code:     EPV-0

(LAMBDA* (C P TR PRIVATE PUBLIC PP1) (AVG-POINT-VALUE))

Function: EXP-POINT-VALUE
Id:       V8663
Code:     EPV-1

(LAMBDA* (C P TR PRIVATE PUBLIC PP1)
         (SUM (RANGE 'CARD)
          (LAMBDA* (V11)
            (*TIMES* (POINT-VALUE V11)
                     (PROB-CARD C P TR PRIVATE PUBLIC PP1 V11)))))

.......................................................................

Function: PROB-CARD
Id:       V8693
Code:     PCD-0

(LAMBDA* (C P TR PRIVATE PUBLIC PP1 V11) (ONE-DIV-CARD))

Function: PROB-CARD
Id:       V8687
Code:     PCD-1

(LAMBDA* (C P TR PRIVATE PUBLIC PP1 V11)
    (IFF (KNOWN (LAMBDA* (D) (CARD PP1 TR D))
                (NEXT-STATE C P TR PRIVATE PUBLIC))
         (IFF (EQUALL (EVALUATE (LAMBDA* (D) (CARD PP1 TR D))
```

```
                            (NEXT-STATE C P TR PRIVATE PUBLIC))
                  V11)
              1
              0)
     (P-CARD C P TR PRIVATE PUBLIC PP1 V11)))
```

....

```
Function: EXP-POINT-VALUE-INVARIANT
Id:       V8737

(LAMBDA* (P TR PRIVATE PUBLIC PP1)
    (SUM (RANGE 'CARD)
     (LAMBDA* (V11)
       (*TIMES* (POINT-VALUE V11)
                (PROB-CARD-INVARIANT P TR PRIVATE PUBLIC PP1 V11)))))
```

....

```
Function: EXP-NEXT-SCORE
Id:       V8642

(LAMBDA* (C P TR PRIVATE PUBLIC)
    (*TIMES* (EXP-NEXT-WIN C P TR PRIVATE PUBLIC)
             (EXP-NEXT-TRICK-VALUE C P TR PRIVATE PUBLIC)))
```

....

```
Function: EXP-NEXT-WIN
Id:       V8623
Code:     ENW-0

(LAMBDA* (C P TR PRIVATE PUBLIC) (AVG-WIN))

Function: EXP-NEXT-WIN
Id:       V8620
Code:     ENW-1

(LAMBDA* (C P TR PRIVATE PUBLIC)
    (PRODUCT (OPPONENTS P)
      (LAMBDA* (OP1) (EXP-NEXT-DEFEATS C P TR PRIVATE PUBLIC OP1))))
```

....

```
Function: EXP-NEXT-DEFEATS
Id:       V8573

(LAMBDA* (C P TR PRIVATE PUBLIC OP1)
    (SUM (RANGE 'CARD)
     (LAMBDA* (V12)
       (*TIMES* (PROB-MY-NEXT-CARD C P TR PRIVATE PUBLIC V12)
                (PROB-NEXT-DEFEATS-IN-SUIT-INVARIANT V12)
                (PROB-NEXT-IN-LEAD-SUIT-INVARIANT V12)))))
```

....

```
Function: PROB-MY-NEXT-CARD
```

```
Id:       V8552

(LAMBDA* (C P TR PRIVATE PUBLIC V11)
 (/DIV/
    (*TIMES* (PROB-MEMBER-LEGAL-CHOICES C P TR PRIVATE PUBLIC V11)
      (PROB-OPTIMAL C P TR PRIVATE PUBLIC V11))
      (EXP-CARDINALITY C P TR PRIVATE PUBLIC V11)))

.......................................................................

Function: PROB-NEXT-DEFEATS-IN-SUIT-INVARIANT
Id:       V8596

(LAMBDA* (V12)
         (SUM (RANGE 'CARD)
              (LAMBDA* (V21)
                (*TIMES* (ONE-DIV-CARD)
                         (DEFEATS-IN-SUIT V12 V21)))))

.......................................................................

Function: PROB-NEXT-IN-LEAD-SUIT-INVARIANT
Id:       V8603

(LAMBDA* (V12)
         (SUM (RANGE 'LEAD-SUIT)
          (LAMBDA* (V31) (*TIMES* (IN-SUIT V12 V31)
                                  (ONE-DIV-LEAD-SUIT)))))

.......................................................................

Function: EXP-NEXT-TRICK-VALUE
Id:       V8634
Code:     ENT-0

(LAMBDA* (C P TR PRIVATE PUBLIC) (AVG-TRICK-VALUE))

Function: EXP-NEXT-TRICK-VALUE
Id:       V8636
Code:     ENT-1

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (+PLUS+
           (EXP-MY-NEXT-POINT-VALUE C P TR PRIVATE PUBLIC)
           (SUM (OPPONENTS P)
            (LAMBDA* (PP1)
               (EXP-OP-NEXT-POINT-VALUE C P TR PRIVATE PUBLIC PP1)))))

.......................................................................

Function: EXP-MY-NEXT-POINT-VALUE
Id:       V8626
Code:     ENP-0

(LAMBDA* (C P TR PRIVATE PUBLIC) (AVG-POINT-VALUE))

Function: EXP-MY-NEXT-POINT-VALUE
```

```
Id:       V8628
Code:     ENP-1

(LAMBDA* (C P TR PRIVATE PUBLIC)
         (SUM (RANGE 'CARD)
          (LAMBDA* (V11)
            (*TIMES*
              (POINT-VALUE V11)
              (PROB-MY-NEXT-CARD C P TR PRIVATE PUBLIC V11)))))

.......................................................................

Function: EXP-OP-NEXT-POINT-VALUE
Id:       V8631

(LAMBDA* (C P TR PRIVATE PUBLIC PP1) (AVG-POINT-VALUE))

.......................................................................

Function: PROB-MEMBER-LEGAL-CHOICES
Id:       V8532

(LAMBDA* (C P TR PRIVATE PUBLIC V11)
         (*TIMES*
          (PROB-NEXT-IN-HAND C P TR PRIVATE PUBLIC V11)
          (PROB-NEXT-UNPLAYED C P TR PRIVATE PUBLIC V11)
          (PROB-IN-DECK C P TR PRIVATE PUBLIC V11)
          (PROB-NEXT-LEGAL C P TR PRIVATE PUBLIC V11)))

.......................................................................

Function: PROB-NEXT-IN-HAND
Id:       V8525

(LAMBDA* (C P TR PRIVATE PUBLIC V11)
     (IFF (KNOWN (LAMBDA* (D) (HAND V11 D))
                 (NEXT-STATE C P TR PRIVATE PUBLIC))
          (IFF (EQUALL (EVALUATE (LAMBDA* (D) (HAND V11 D))
                                 (NEXT-STATE C P TR PRIVATE PUBLIC))
                       P)
               1
               0)
      (P-NEXT-IN-HAND C P TR PRIVATE PUBLIC V11)))

.......................................................................

Function: P-NEXT-IN-HAND
Id:       V8522

(LAMBDA* (C P TR PRIVATE PUBLIC V11) (AVG-HAND))

.......................................................................

Function: PROB-NEXT-UNPLAYED
Id:       V8489

(LAMBDA* (C P TR PRIVATE PUBLIC V11)
```

```
  (SUM (RANGE 'UNPLAYED)
   (LAMBDA* (V23)
     (*TIMES*
       (PROB-EQUP-INVARIANT P TR PRIVATE PUBLIC V11 V23)
       (SUM (RANGE 'EQUALL)
            (LAMBDA* (V14)
             (*TIMES* (IF-A-0-IF-B-1-0 V23 V14)
                      (PROB-EQCD C P TR PRIVATE PUBLIC V11 V14))))))))
```

....

```
Function: PROB-EQUP-INVARIANT
Id:       V8485

(LAMBDA* (P TR PRIVATE PUBLIC V11 V23)
    (IFF
     (KNOWN (LAMBDA* (D)
              (UNPLAYED V11 P (PRIOR-TRICK (FOLLOWING-TRICK TR)) D))
            (NEXT-STATE NIL P TR PRIVATE PUBLIC))
     (IFF (EQUALL (EVALUATE
                    (LAMBDA* (D)
                     (UNPLAYED V11
                               P
                               (PRIOR-TRICK (FOLLOWING-TRICK TR))
                               D))
                    (NEXT-STATE NIL P TR PRIVATE PUBLIC))
                 V23)
           1
           0)
     (P-EQUP-INVARIANT P TR PRIVATE PUBLIC V11 V23)))
```

....

```
Function: P-EQUP-INVARIANT
Id:       V8482

(LAMBDA* (P TR PRIVATE PUBLIC V11 V23) (/DIV/ 1 2))
```

....

```
Function: PROB-EQCD
Id:       V8503

(LAMBDA* (C P TR PRIVATE PUBLIC V11 V14)
    (IFF (KNOWN (LAMBDA* (D)
                 (EQUALL (CARD P (PRIOR-TRICK (FOLLOWING-TRICK TR)) D)
                         V11))
                (NEXT-STATE C P TR PRIVATE PUBLIC))
         (IFF (EQUALL
                (EVALUATE
                  (LAMBDA* (D)
                     (EQUALL (CARD P
                                  (PRIOR-TRICK (FOLLOWING-TRICK TR))
                                  D)
                             V11))
                  (NEXT-STATE C P TR PRIVATE PUBLIC))
                V14)
```

```
              1
              0)
     (P-EQCD C P TR PRIVATE PUBLIC V11 V14)))
```

....

```
Function: P-EQCD
Id:       V8506

(LAMBDA* (C P TR PRIVATE PUBLIC V11 V14) (/DIV/ 1 2))
```

....

```
Function: PROB-IN-DECK
Id:       V8545

(LAMBDA* (C P TR PRIVATE PUBLIC V11) 1)
```

....

```
Function: PROB-NEXT-LEGAL
Id:       V8547

(LAMBDA* (C P TR PRIVATE PUBLIC V11) (/DIV/ 1 2))
```

....

```
Function: PROB-OPTIMAL
Id:       V8549

(LAMBDA* (C P TR PRIVATE PUBLIC V11) (/DIV/ 1 2))
```

....

```
Function: EXP-CARDINALITY
Id:       V8563

(LAMBDA* (C P TR PRIVATE PUBLIC V11) (AVG-CARDINALITY))
```

....

```
Function: AVG-POINT-VALUE
Id:       V8775

(LAMBDA* NIL (AVERAGE (RANGE ’POINT-VALUE)))
```

....

```
Function: AVG-WIN
Id:       V8755

(LAMBDA* NIL (AVERAGE (RANGE ’WIN)))
```

....

```
Function: AVG-TRICK-SCORE
Id:       V8740
```

```
(LAMBDA* NIL (AVERAGE (RANGE 'TRICK-SCORE)))

..............................................................

Function: AVG-DEFEATS
Id:       V8739

(LAMBDA* NIL (AVERAGE (RANGE 'DEFEATS)))

..............................................................

Function: AVG-TRICK-VALUE
Id:       V8738

(LAMBDA* NIL (AVERAGE (RANGE 'TRICK-VALUE)))

..............................................................

Function: AVG-CARDINALITY
Id:       V8562

(LAMBDA* NIL (AVERAGE (RANGE 'CARDINALITY)))

..............................................................

Function: AVG-HAND
Id:       V8530

(LAMBDA* NIL (/DIV/ 1 (CARDINALITY (RANGE 'HAND))))

..............................................................

Function: ONE-DIV-LEAD-SUIT
Id:       V8766

(LAMBDA* NIL (/DIV/ 1 (CARDINALITY (RANGE 'LEAD-SUIT))))

..............................................................

Function: ONE-DIV-CARD
Id:       V8756

(LAMBDA* NIL (/DIV/ 1 (CARDINALITY (RANGE 'CARD))))

..............................................................

Function: DEFEATS-IN-SUIT
Id:       V8610

(LAMBDA* (C1 C2)
         (IFF
          (ORR (NOTT (EQUALL (SUIT C2) (SUIT C1)))
           (>GT> (RANK-VALUE (RANK C1)) (RANK-VALUE (RANK C2))))
          1 0))

..............................................................

Function: IN-SUIT
```

```
Id:       V8614

(LAMBDA* (C1 S) (IFF (EQUALL (SUIT C1) S) 1 0))

.......................................................................

Function: IF-A-0-IF-B-1-0
Id:       V8508

(LAMBDA* (V23 V14) (IFF (EQUALL (IFF NIL T (IFF V14 NIL V23)) T) 1 0))
```

Appendix K

Semantics of Approximate Hearts Theories

K.1. Equivalent forms of Approximate Hearts Functions

An algebraic analysis of approximate hearts functions generated by *POLLYANNA* is presented in this section. The analysis begins with the definitions of functions appearing in the approximation generator final state, as shown in Appendix J. The definitions are then simplified using straightforward, but tedious, algebraic manipulations. These manipulations were carried out manually. They have not been mechanized in the *POLLYANNA* system itself. This analysis demonstrates that *POLLYANNA*'s output is equivalent to the simpler algebraic expressions shown in Figure 3-5. (See Section 3.3.1.1.) These simpler expressions are used in Chapter 3 to carry out a semantic analysis of approximate hearts theories. The semantic analysis demonstrates equivalence between approximate hearts theories and verbal heuristics.

K.1.1. Analysis of *Exp-Win*

1. Naive Version of *Exp-Win (EWN-0)*: Results from applying *Equiprobable:Numeric* to the random variable *λ(d)Win(p,tr,d)*. This random variable indicates whether player *p* wins trick *tr*, the current trick.

   ```
   Exp-Win(c,p,tr,private,public)

    = Average(Range(Win))

    = 1 / 2
   ```

2. Sophisticated Version of *Exp-Win (EWN-1)*: Results from using *Equiprobable:Equality* to assume the opponents yet to play in the current trick are equally likely to play any card in the deck. It also uses *Independence:Nary* to assume that the odds of defeating each of the opponents are probabilistically independent. This function is a product of three probabilities, each representing the odds of defeating a single opponent in the current trick. It can be rewritten as a rule that tests whether player *p*'s card choice *c* is defeated by some opponent's card already played. If so, then *Exp-Win* is zero. Otherwise, *Exp-Win* reduces to a polynomial function of degree *N* in *Rank(c)*, where *N* is the number of opponents yet to play in the current trick.

   ```
   Exp-Win(c,p,tr,private,public) =

    = Π (p’ in Opponents(c))
        Exp-Defeats(c,p,tr,private,public,p’)
   ```

```
= Π (p’ in Opponents(c))
    Σ (c’ in Deck)
      Prob-Known-Card(c,p,tr,private,public,p,c’)
      * Exp-Defeats-Sum-Sum(c,p,tr,private,public,p’,c’)

= Π (p’ in Opponents(c))
    Exp-Defeats-Sum-Sum(c,p,tr,private,public,p’,c)

= Π (p’ in Opponents(c))
    Σ (s in Suits)
      Prob-Lead-Suit(c,p,tr,private,public,s)
      * Exp-Defeats-Sum(c,p,tr,private,public,p’,c,s)

= Π (p’ in Opponents(c))
    1. Let s = The lead suit for trick tr.
    2. Return Exp-Defeats-Sum(c,p,tr,private,public,p’,c,s)

= Π (p’ in Opponents(c))
    1. Let s = The lead suit for trick tr.
    2. Return
       {Σ (c’ in Deck)
          Prob-Card-Invariant(c,p,tr,private,public,p’,c’)
          * Defeats(c,c’,s)}

= Π (p’ in Opponents(c))
    1. Let s = The lead suit for trick tr.
    2. Return
       {Σ (c’ in Deck)
          (If (Player p’ already played c’’ in trick tr.)
               then (If c’ = c’’ then 1 else 0)
              else 1/|Range(Card)|)
          * Defeats(c,c’,s)}

= Π (p’ in Opponents(c))
    1. Let s = The lead suit for trick tr.
    2. If (Player p’ already played in trick tr.)
          then a. Let c’ = The card played by player p’.
               b. Return Defeats(c,c’,s).
          else [Average (c’ in Deck) Defeats(c,c’,s)]

= Π (p’ in Opponents(c))
    1. Let s = The lead suit for trick tr.
    2. If (Player p’ already played in trick tr.)
          then a. Let c’ = The card played by player p’.
               b. Return Defeats(c,c’,s).
          else If Suit(c) = s
                  then Return {A1 * [A2 + Rank(c)]}^N
                  else Return 0

= If (Card c defeats all cards on the table)
     then 1. Let N = The number of opponents yet to play.
          2. Return {A1 * [A2 + Rank(c)]}^N
     else Return 0

= {A1 * [A2 + Rank(c)]}^N
  * {If (Card c defeats all cards on the table)
        then 1
```

```
        else 0}

A1 = (1 / |Range(Card)|) = 1 / 52
A2 = 3 * (Number of Cards in a Suit) - (Minimum Rank) = 37
N  = The number of opponents yet to play.
```

K.1.2. Analysis of *Exp-Trick-Value*

1. Naive Version of *Exp-Trick-Value (ETV-0)*: Results from applying *Equiprobable:Numeric* to the random variable *λ(d)Trick-Value(tr,d)*, which measures the value of trick *tr*, the current trick.

```
Exp-Trick-Value(c,p,tr,private,public)

 = Average(Range(Trick-Value))

 = Average({0,1,2,3,4,13,14,15,16})

 = (0+1+2+3+4+13+14+15+16)/9 = 68 / 9
```

2. Sophisticated Version of *Exp-Trick-Value (ETV-1)*: Results from using *Equiprobable:Equality* to assume the opponents yet to play in the current trick are equally likely to play any card in the deck. This function is a sum of four terms, each representing the expected point value of the card of one of four players. The terms corresponding to opponents are all constants, independent of player *p*'s card choice *c*. The function can be written as an increasing linear function of *Point-Value(c)*.

```
Exp-Trick-Value(c,p,tr,private,public) =

 = Exp-Point-Value(c,p,tr,private,public,p)
   + Σ (p' in Opponents(p))
       Exp-Point-Value-Invariant(c,p,tr,private,public,p')

 = {Σ (c' in Deck)
      Prob-Card(c,p,tr,private,public,p,c')
      * Point-Value(c')}
   + {Σ (p' in Opponents(p))
       Exp-Point-Value-Invariant(c,p,tr,private,public,p')}

 = Point-Value(c)
   + {Σ (p' in Opponents(p))
       Exp-Point-Value-Invariant(c,p,tr,private,public,p')}

 = Point-Value(c)
   + {Σ (p' in Opponents(p))
       Σ (c' in Deck)
         Prob-Card-Invariant(c,p,tr,private,public,p',c')
         * Point-Value(c')}

 = Point-Value(c)
   + {Σ (p' in Opponents(p))
       Σ (c' in Deck)
         (If (Player p' already played c'' in trick tr.)
             then (If (c'' = c') then 1 else 0)
             else 1/|Range(Card)|)
         * Point-Value(c')}

 = Point-Value(c)
```

```
  + ∑ (p' in Opponents(p))
      If (Player p' already played in trick tr.)
         then 1. Let c' be the card played by player p'.
              2. Return Point-Value(c')
         else [Average (c' in Deck) Point-Value(c')]

= B1 + (N * B2) + Point-Value(c)

B1 = Sum of points on the table.
B2 = [Average (c' in Deck) Point-Value(c')] = 1 / 2.
N  = Number of opponents yet to play.
```

K.1.3. Analysis of *Exp-Next-Win*

1. Naive Version of *Exp-Next-Win (ENW-0)*: Results from applying *Equiprobable:Numeric* to the random variable λ*(d)Win(p,tr+1,d)*, which indicates whether player *p* wins trick *tr+1*, the next trick.

```
Exp-Next-Win(c,p,tr,private,public)

 = Average(Range(Win))

 = 1 / 2
```

2. Sophisticated Version of *Exp-Next-Win (ENW-1)*: Results from using *Equiprobable:Equality* three times to make three key assumptions: (1) Opponents are equally likely to play any card in the deck in the next trick. (2) The lead suit for the next trick is equally likely to be any suit. (3) In the next trick, player *p* is equally likely to play any card in *Hand(p)-{c}*, i.e. any card remaining in his hand. This function is a product of three probabilities, each representing the odds of defeating one of three opponents. Each factor computes the probability that player *p* defeats opponent *p*' in the next trick. This function can be rewritten as a monotonically increasing cubic polynomial function of the ranks of all cards in *Hand(p)-{c}*, i.e., the cards remaining in player *p*'s hand.

```
Exp-Next-Win(c,p,tr,private,public) =

 = Π (p' in Opponents(c))
     Exp-Next-Defeats(c,p,tr,private,public,p')

 = Π (p' in Opponents(c))
     ∑ (c' in Deck)
       Prob-My-Next-Card(c,p,tr,private,public,c')
       * Prob-Next-Defeats-In-Suit-Invariant(c')
       * Prob-Next-In-Lead-Suit-Invariant(c')

 = Π (p' in Opponents(c))
     ∑ (c' in Deck)
       K0 * [If Member(c',Hand(p)-{c}) then 1 else 0]
       * Prob-Next-Defeats-In-Suit-Invariant(c')
       * Prob-Next-In-Lead-Suit-Invariant(c')

 = Π (p' in Opponents(c))
     ∑ (c' in Hand(p)-{c})
       K0 * Prob-Next-Defeats-In-Suit-Invariant(c')
       * Prob-Next-In-Lead-Suit-Invariant(c')

 = Π (p' in Opponents(c))
     ∑ (c' in Hand(p)-{c})
```

```
      K0 * Prob-Next-Defeats-In-Suit-Invariant(c’)
      * ∑ (s in Suits)
          [If Suit(c’)=s then 1/|Range(Suit)| else 0]

= Π (p’ in Opponents(c))
    ∑ (c’ in Hand(p)-{c})
      K0 * Prob-Next-Defeats-In-Suit-Invariant(c’)
      * 1/|Range(Suit)|

= Π (p’ in Opponents(c))
    ∑ (c’ in Hand(p)-{c})
      K0 * K1
      * Prob-Next-Defeats-In-Suit-Invariant(c’)

= Π (p’ in Opponents(c))
    ∑ (c’ in Hand(p)-{c})
      K0 * K1
      * ∑ (c’’ in Deck)
          Defeats-In-Suit(c’,c’’) * 1/|Range(Card)|

= Π (p’ in Opponents(c))
    ∑ (c’ in Hand(p)-{c})
      K0 * K1 * K2 * [K3 + Rank(c’)]
```

$= \{\sum$ `(c’ in Hand(p)-{c}) K0*K1*K2* [K3 + Rank(c’)]`$\}^3$

$= \{$`C1 + C2 *` $\sum$ `(c’ in Hand(p)-{c})  Rank(c’)` $\}^3$

$= \{$`C1 + C2 * S` $\}^3$

```
K0 = (1 / 2) * (1 / 52) = 1 / 104
K1 = 1 / |Range(Suit)| = 1 / 4
K2 = 1 / |Range(Card)| = 1 / 52
K3 = (Number of Cards in other Suits) - Minimum Rank = 37
C1 = K0 * K1 * K2 * K3 * |Hand(p)-{c}|
   = 37 * |Hand(p)-{c}| / (4 * 104 * 52)
   = 37 * |Hand(p)-{c}| / 21632
C2 = K0 * K1 * K2
   = 1 / (4 * 104 * 52)
   = 1 / 21632
S  = Sum of Ranks in Hand(p)-{c}.
```

K.1.4. Analysis of *Exp-Next-Trick-Value*

1. Naive Version of *Exp-Next-Trick-Value (ENT-0):* Results from applying *Equiprobable:Numeric* to the random variable λ*(d)Trick-Value(tr+1,d)*, which measures the value of trick *tr+1*, the next trick.

```
Exp-Next-Trick-Value(c,p,tr,private,public)

 = Average(Range(Trick-Value))

 = Average({0,1,2,3,4,13,14,15,16})

 = (0+1+2+3+4+13+14+15+16)/9 = 68 / 9
```

2. Sophisticated Version of *Exp-Next-Trick-Value (ENT-1):* Results from using *Equiprobable:Equality* twice to to make two key assumptions: (1) Opponents are equally likely to play any card in the deck in the next trick. (2) In the next trick, player *p* is equally likely to play any card in *Hand(p)-{c}*, i.e. any card remaining in his hand. This function is a sum of four terms, each representing the expected point value of the card played in the next trick by one four players. The terms corresponding to opponents are all constants, independent of player *p*'s card choice *c* in trick *tr*, the current trick. It can be written as a increasing linear function of the total point value of all cards in *Hand(p)-{c}*, i.e., all cards remaining in player *p*'s hand.

```
Exp-Next-Trick-Value(c,p,tr,private,public) =

 = Exp-My-Next-Point-Value(c,p,tr,private,public)
   + ∑ (p' in Opponents(p))
       Exp-Op-Next-Point-Value(c,p,tr,private,public,p')

 = Exp-My-Next-Point-Value(c,p,tr,private,public)
   + ∑ (p' in Opponents(p))
       Average(Range(Point-Value))

 = D0 + Exp-My-Next-Point-Value(c,p,tr,private,public)

 = D0 + ∑ (c' in Deck)
          Point-Value(c')
          * Prob-My-Next-Card(c,p,tr,private,public,c')

 = D0 + ∑ (c' in Deck)
          Point-Value(c')
          * D1 * [If Member(c',Hand(p)-{c}) then 1 else 0]

 = D0 + D1 * { ∑ (c' in Hand(P)-{c}) Point-Value(c') }

 = D0 + D1 * S

 S  = Sum of Points in Hand(p)-{c}
 D0 = (Number of Opponents) * Average(Range(Point-Value))
    = 3 * (0 + 1 + 13) / 3 = 14.
 D1 = 1 / 104
```

K.1.5. Additional Key Functions

Analysis of *Prob-My-Next-Card(c,p,tr,private,public,c')*: This function computes the odds that player *p* will play card *c*' in trick *tr+1* if he plays card *c* in trick *tr*. The approximate version results from assuming: (1) All cards are equally likely to be optimal or sub-optimal for the next trick, (2) The number of optimal choices for player *p* in trick *tr+1* is equally likely to be any number in *Range(Cardinality)*, the possible sizes of card sets. (3) All cards that are in player *p*'s hand at trick *tr+1* are equally likely to be legal or illegal choices for trick *tr+1*.

```
Prob-My-Next-Card(c,p,tr,private,public,c') =

  = Prob-Member-Legal-Choices(c p tr private public c')
    * Prob-Optimal(c p tr private public c')
    * Exp-Cardinality(c p tr private public c')

  = K1 * K2 * Prob-Member-Legal-Choices(c p tr private public c')
```

```
= K1 * K2 *
  * Prob-Next-In-Hand(c p tr private public c')
  * Prob-Next-Unplayed(c p tr private public c')
  * Prob-In-Deck(c p tr private public c')
  * Prob-Next-Legal(c p tr private public c')

= K1 * K2 *
  * Prob-Next-In-Hand(c p tr private public c')
  * Prob-Next-Unplayed(c p tr private public c')
  * Prob-In-Deck(c p tr private public c')
  * Prob-Next-Legal(c p tr private public c')

= K1 * K2 * K3 * K4
  * Prob-Next-In-Hand(c p tr private public c')
  * Prob-Next-Unplayed(c p tr private public c')

= K1 * K2 * K3 * K4
  * (If Member(c',Hand(p)) then 1 else 0)
  * Prob-Next-Unplayed(c p tr private public c')

= K1 * K2 * K3 * K4
  * (If Member(c',Hand(p)) then 1 else 0)
  * (If c=c' then 0 else 1)

= K1 * K2 * K3 * K4
  * (If Member(c',Hand(p)-{c}) then 1 else 0)

= K5 * (If Member(c',Hand(p)-{c}) then 1 else 0)

K1 = 1 / 2
K2 = Average(Range(Cardinality)) = 1 / 26
K3 = 1
K4 = 1 / 2
K5 = K1 * K2 * K3 * K4 = 1 / 104
```

K.2. Assumptions used in Approximate Hearts Functions

1. *Function-Invariance:Constant*: Applied to *Exp-Past-Score*.
2. *Function-Invariance:Constant*: Applied to *Exp-Future-Score*.
3. *Independence-of-Product:Binary*: Applied to *Win* and *Trick-Value* in the current trick.
4. *Independence-of-Product:Binary*: Applied to *Win* and *Trick-Value* in the next trick.
5. *Equiprobable:Numeric*: Applied to *Win* of current trick.
6. *Equiprobable:Numeric*: Applied to *Trick-Value* of current trick.
7. *Equiprobable:Numeric*: Applied to *Win* in next trick.
8. *Equiprobable:Numeric*: Applied to *Trick-Value* of next trick.
9. *Independence-Of-Product:Nary*: Applied to defeating each of three opponents in the current trick:
10. *Independence-Of-Product:Nary*: Applied to defeating each of three opponents in the next trick:
11. *Equiprobable:Equality*: Applied to each player's current trick *Card*, if his card choice is not known.
12. *Equiprobable:Equality*: Applied to each opponent's next trick *Card*.

13. *Equiprobable:Equality*: Applied current trick leader's *Card*, if his card choice is not known.
14. *Function-Invariance:Constant*: Applied to ignore player *p*'s current choice card *c*, when computing odds of opponents' card choices in the current trick.
15. *Function-Invariance:Constant*: Applied to assume (correctly) that the next trick is never the initial trick.
16. *Independence:Three-Conjuncts*: Applied to (a) player *p*'s current trick *Card*, (b) an opponent's current trick *Card* and (c) the *Lead-Suit* of the current trick.
17. *Independence:Three-Conjuncts*: Applied to (a) player *p*'s next trick *Card*, (b) an opponent's next trick *Card* and (c) the *Lead-Suit* of the next trick.
18. *Equiprobable:Equality*: Applied to *Lead-Suit* for the next trick.
19. *Equiprobable:Numeric*: Applied to *Point-Value* of each opponent's next trick card.
20. *Independence:Two-Conjuncts*: Applied to (a) legality of player *p*'s hypothetical next trick card choice and (b) optimality of that card choice.
21. *Independence:Two-Conjuncts* and *Independence:Three-Conjuncts*: Applied to the fact of *p*'s hypothetical next card choice being (a) in the deck, (b) in *p*'s initial hand, (c) not played by *p* at start of next trick and (d) a legal choice given the next lead suit.
22. *Independence:Two-Conjuncts*: Applied to the fact of *p*'s hypothetical next card choice being (a) unplayed at start of current trick and (b) not played during the current trick.
23. *Function-Invariance:Constant*: Applied to (correctly) set the odds of *p*'s hypothetical next card choice being in the deck to one.
24. *Function-Invariance:Constant*: Applied to ignore the dependence on player *p*'s current choice *c* of the odds that *p*'s hypothetical next card choice was unplayed at start of current trick.
25. *Equiprobable:Numeric*: Applied to the number of card choices that are optimal for player *p* in the next trick.
26. *Equiprobable:Boolean*: Applied to the optimality of player *p*'s hypothetical next card choice.
27. *Equiprobable:Boolean*: Applied to legality of player *p*'s hypothetical next card choice.
28. *Equiprobable:Equality*: Applied to the player who was dealt player *p*'s hypothetical next card choice, if the player holding that card is not known.
29. *Equiprobable:Boolean*: Applied to whether player *p*'s hypothetical next card choice is unplayed at start of current trick, if that fact is not known.
30. *Equiprobable:Boolean*: Applied to whether player *p*'s hypothetical next card choice is played in the current trick, if not known.

Function	**Version**	**Assumptions Used**
Exp-Win	**EWN-0**	**5**
	EWN-1	**9,11,13,14,16**
Exp-Trick-Value	**ETV-0**	**6**
	ETV-1	**11,14**
Exp-Next-Win	**ENW-0**	**7**

	ENW-1	10,12,15,17,18,20-30
Exp-Next-Trick-Value	ENT-0	8
	ENT-1	15,19,20 - 30

T1: (EWN-0, ETV-1, ENW-0, ENT-0)
Assumptions: 1-4,5,7,8,11,14.

T2: (EWN-0, ETV-0, ENW-0, ENT-1)
Assumptions: 1-4,5,6,7,15,19,20 - 30.

T3: (EWN-0, ETV-0, ENW-1, ENT-0)
Assumptions: 1-4,5,6,8,10,12,15,17,18,20-30

T4: (EWN-1, ETV-0, ENW-0, ENT-0)
Assumptions: 1-4,6,7,8,9,11,13,14,16.

T5: (EWN-1, ETV-0, ENW-1, ENT-0)
Assumptions: 1-4,6,8,9,10,11,12,13,14,15,16,17,18,20-30.

T6: (EWN-1, ETV-0, ENW-0, ENT-1)
Assumptions: 1-4,6,7,9,11,13,14,15,16,19,20 - 30.

Appendix L

Approximation Generator Output: Scheduling

A summary of one final state produced by *POLLYANNA*'s approximation generator for the scheduling domain is found below. The principal functions in the final state listed in a diagram that describes referencing relationships. The diagram also indicates which functions have multiple versions, and the number of versions of each. The actual output of *POLLYANNA*'s approximation generator for the scheduling domain follows after the referencing diagram. Only a partial description of the final state given. All versions computing exact probabilities or expectation values are intractable, and were therefore omitted. This data was generated using the scheduling theory found in Appendix D

```
Evaluation* [3]
 |
 |---Eval-Static
 Or
 |---Evaluation-A
 |    |---Eval-Static
 Or
 |---Evaluation-B
      |---Evaluation-A
           |---Eval-Static

Eval-Static
 |
 |---Exp-Job-Value
      |---Exp-Preconditions* [2]
      |    |---Exp-Check-Precond
      |         |---Prob-Prec-Comp-Alloc
      |              |---Prob-Prec
      |              |---Prob-Comp* [2]
      |              |    |---Prob-Alloc
      |              |         |---Prob-Member-Times
      |              |         |---Prob-Choice
      |              |              |---P-Choice
      |              |                   |---P-Alloc
      |              |
      |              |---Prob-Alloc
      |                   |---Prob-Member-Times
      |                   |---Prob-Choice
      |                        |---P-Choice
      |                             |---P-Alloc
      |
      |---Exp-Penalty* [2]
```

```
    |    |---Prob-Comp* [2]
    |         |---Prob-Alloc
    |              |---Prob-Member-Times
    |              |---Prob-Choice
    |                   |---P-Choice
    |                        |---P-Alloc
    |
    |---Exp-Weight
```

```
FUNCTION:                VERSIONS:

Evaluation               EVL-0, EVL-1, EVL-2
Exp-Preconditions        EPC-0, EPC-1
Prob-Comp                PCM-0, PCM-1
Exp-Penalty              EPN-0, EPN-1
```

....

```
Function: EVALUATION
Id:       V8077
Code:     EVL-0

(LAMBDA* (INSTANCE TM G) (EVAL-STATIC INSTANCE G))

Function: EVALUATION
Id:       V8080
Code:     EVL-1

(LAMBDA* (INSTANCE TM G) (EVALUATION-A INSTANCE TM G))

Function: EVALUATION
Id:       V8083
Code:     EVL-2

(LAMBDA* (INSTANCE TM G) (EVALUATION-B INSTANCE TM G))
```

....

```
Function: EVAL-STATIC
Id:       V8078

(LAMBDA* (INSTANCE G)
   (SUM (JOBS INSTANCE)
     (LAMBDA* (J1) (EXP-JOB-VALUE INSTANCE G J1))))
```

....

```
Function: EVALUATION-A
Id:       V8106

(LAMBDA* (INSTANCE TM G)
  (IFF (FINAL-STATE? INSTANCE TM G) (KNOWN-STATE-VALUE INSTANCE G)
       (MINIMIZE (OPTIONS TM G)
         (LAMBDA* (OP)
           (EVAL-STATIC INSTANCE (NEXT-STATE OP TM G))))
   ))
```

....

```
Function: EVALUATION-B
Id:       V8108

(LAMBDA* (INSTANCE TM G)
  (IFF (FINAL-STATE? INSTANCE TM G) (KNOWN-STATE-VALUE INSTANCE G)
       (MINIMIZE (OPTIONS TM G)
       (LAMBDA* (OP)
           (EVALUATION-A INSTANCE
                         (NEXT-TIME TM)
                         (NEXT-STATE OP TM G))))))

..................................................................

Function: EXP-JOB-VALUE
Id:       V8040

(LAMBDA* (INSTANCE G J1)
         (*TIMES* (EXP-PRECONDITIONS INSTANCE G J1)
                  (EXP-PENALTY INSTANCE G J1)
                  (EXP-WEIGHT INSTANCE G J1)))

..................................................................

Function: EXP-PRECONDITIONS
Id:       V8087
Code:     EPC-0

(LAMBDA* (INSTANCE G J1) (AVERAGE (RANGE 'PRECONDITIONS)))

Function: EXP-PRECONDITIONS
Id:       V8065
Code:     EPC-1

(LAMBDA* (INSTANCE G J1)
         (PRODUCT (REQUIREMENTS J1 INSTANCE)
          (LAMBDA* (JJ2) (EXP-CHECK-PRECOND INSTANCE G J1 JJ2))))

..................................................................

Function: EXP-CHECK-PRECOND
Id:       V8068

(LAMBDA* (INSTANCE G J1 JJ2)
  (SUM (RANGE 'PRECONDITIONS)
   (LAMBDA* (V11)
    (SUM (RANGE 'COMPLETION-TIME)
     (LAMBDA* (V21)
      (SUM (RANGE 'ALLOCATION-TIME)
       (LAMBDA* (V31)
        (*TIMES* (CHECK-PRECOND V11 V21 V31)
          (PROB-PREC-COMP-ALLOC INSTANCE G J1 JJ2 V11 V21 V31))))
                     )))))

..................................................................

Function: PROB-PREC-COMP-ALLOC
Id:       V8074
```

```
(LAMBDA* (INSTANCE G J1 JJ2 V11 V21 V31)
         (*TIMES* (PROB-PREC INSTANCE G JJ2 V11)
                  (PROB-COMP INSTANCE G JJ2 V21)
                  (PROB-ALLOC G J1 V31)))

....................................................................

Function: PROB-PREC
Id:       V8091

(LAMBDA* (INSTANCE G JJ2 V11) (/DIV/ 1 2))

....................................................................

Function: PROB-COMP
Id:       V8114
Code:     PCM-0

(LAMBDA* (INSTANCE G JJ2 V21)
  (SUM (RANGE ’ALLOCATION-TIME)
    (LAMBDA* (V15)
      (*TIMES* (IFF (EQUALL (+PLUS+ V15 0) V21)
                    1
                    0)
               (PROB-ALLOC G JJ2 V15)))))

Function: PROB-COMP
Id:       V8020
Code:     PCM-1

(LAMBDA* (INSTANCE G JJ2 V21)
  (SUM (RANGE ’ALLOCATION-TIME)
   (LAMBDA* (V15)
    (*TIMES* (IFF (EQUALL (+PLUS+ V15
                                  (WORKING-TIME JJ2 INSTANCE))
                          V21)
                  1
                  0)
             (PROB-ALLOC G JJ2 V15)))))

....................................................................

Function: PROB-ALLOC
Id:       V8105

(LAMBDA* (G J1 V31)
 (IFF (KNOWN (LAMBDA* (E) (ALLOCATION-TIME J1 E)) G)
      (IFF (EQUALL (EVALUATE (LAMBDA* (E) (ALLOCATION-TIME J1 E))
                             G)
                   V31)
           1
           0)
      (*TIMES* (PROB-MEMBER-TIMES G V31)
               (PROB-CHOICE G J1 V31))))

....................................................................

Function: P-ALLOC
```

```
Id:       V8097

(LAMBDA* (G J1 V31)
    (/DIV/ 1 (CARDINALITY (RANGE 'ALLOCATION-TIME))))

....................................................................

Function: PROB-MEMBER-TIMES
Id:       V8577

(LAMBDA* (G V31) 1)

....................................................................

Function: PROB-CHOICE
Id:       V8101

(LAMBDA* (G J1 V31)
  (IFF (KNOWN (LAMBDA* (E) (CHOICE V31 E)) G)
       (IFF (EQUALL (EVALUATE (LAMBDA* (E) (CHOICE V31 E)) G)
                    J1)
            1
            0)
       (P-CHOICE G J1 V31)))

....................................................................

Function: P-CHOICE
Id:       V8560

(LAMBDA* (G J1 V31)
   (*TIMES* (PROB-MEMBER-JOBS G J1) (P-ALLOC G J1 V31)))

....................................................................

Function: PROB-MEMBER-JOBS
Id:       V8576

(LAMBDA* (G J1) 1)

....................................................................

Function: EXP-PENALTY
Id:       V8088
Code:     EPN-0

(LAMBDA* (INSTANCE G J1) (AVERAGE (RANGE 'PENALTY)))

Function: EXP-PENALTY
Id:       V8024
Code:     EPN-1

(LAMBDA* (INSTANCE G J1)
 (SUM (RANGE 'COMPLETION-TIME)
  (LAMBDA* (V14)
   (*TIMES* (PENALTY J1 INSTANCE V14)
            (PROB-COMP INSTANCE G J1 V14)))))
```

```
.....................................................................

Function: EXP-WEIGHT
Id:       V8089

(LAMBDA* (INSTANCE G J1) (AVERAGE (RANGE 'WEIGHT)))

.....................................................................

Function: KNOWN-STATE-VALUE
Id:       V8109

(LAMBDA* (INSTANCE G)
  (IFF (KNOWN (LAMBDA* (E) (VALUE INSTANCE E)) G)
       (EVALUATE (LAMBDA* (E) (VALUE INSTANCE E)) G)
       (EXP (LAMBDA* (E) (VALUE INSTANCE E)) G)))

.....................................................................

Function: CHECK-PRECOND
Id:       V8036

(LAMBDA* (X1 X2 X3) (*TIMES* (IFF (<LE< X2 X3) 1 0) X1))
```

Appendix M

Semantics of Approximate Scheduling Theories

M.1. Equivalent forms of Approximate Scheduling Theories

An algebraic analysis of approximate scheduling functions generated by *POLLYANNA* is presented in this section. The analysis begins with the definitions of functions appearing in the approximation generator final state, as shown in Appendix L. The definitions are then simplified using straightforward, but tedious, algebraic manipulations. These manipulations were carried out manually. They have not been mechanized in the *POLLYANNA* system itself. This analysis demonstrates that *POLLYANNA*'s output is equivalent to the simpler algebraic expressions shown in Figure 3-5. (See Section 3.3.1.1.) These simpler expressions are used in Chapter 3 to carry out a semantic analysis of approximate scheduling theories. The semantic analysis demonstrates equivalence between approximate scheduling theories and verbal heuristics.

M.1.1. Analysis of *Soonest Scheduling Deadline First*

```
Scheduling Theory #1:    (EVL-0, EPN-1, EPC-0, PCM-1)

Heuristic:               "Soonest Scheduling Deadline First"
```

1. Start with the definition of *Exp-Job-Value*. Substitute the definitions of *Exp-Preconditions (EPC-0)* and *Exp-Weight*, which are just constants:

```
Exp-Job-Value(i,g,j) =

 =  Exp-Preconditions(i,g,j)
    * Exp-Penalty(i,g,j)
    * Exp-Weight(i,g,j)

 = K0 * K1 * Exp-Penalty(i,g,j)
```

2. Substitute the definitions of the function *Exp-Penalty (EPN-1)* and the function *Penalty*:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t in  Range(Completion-Time))
       Penalty(j,i,t) *  Prob-Comp(i,g,j,t)
```

```
 = K0 * K1
   * ∑ (t in  Range(Completion-Time))
       Prob-Comp(i,g,j,t)
       * (If t ≤ Deadline(j,i) then 1 else 0)
```

3. All summation terms are zero unless *t≤Deadline(j,i)*. The summation range can be restricted to these values:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       Prob-Comp(i,g,j,t)
```

4. Substitute the definition of *Prob-Comp (PCM-1)*:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       ∑ (t’ in Range(Allocation-Time))
         If [t = t’ + Working-Time(j,i)]
            then Prob-Alloc(g,j,t’)
            else 0
```

5. The inner summation can be eliminated, since only one term is non zero:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       Prob-Alloc(g,j, [t - Working-Time(j,i)] )
```

6. Substitute the definition of *Prob-Alloc*. Then paraphrase the definition:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If Known(λ(e) Allocation-Time(j,e), g)
          then If {[t - Working-Time(j,i)]
                     = Evaluate(λ(e) Allocation-Time(j,e), g)}
                  then 1
                  else 0
          else Prob-Member-Times(g,[t - Working-Time(j,i)])
                 * Prob-Choice(G,J,[t - Working-Time(j,i)])

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If (Job j is allocated some slot t’ in state g)
          then If t’ = [t - Working-Time(j,i)]
                  then 1
                  else 0
          else Prob-Member-Times(g,[t - Working-Time(j,i)])
                 * Prob-Choice(G,J,[t - Working-Time(j,i)])
```

7. The function *Prob-Member-Times* is just a constant:

```
Exp-Job-Value(i,g,j) =
```

```
 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If (Job j is allocated some slot t′ in state g)
          then If t′ = [t - Working-Time(j,i)]
                  then 1
                  else 0
          else K2 * Prob-Choice(g,j,[t - Working-Time(j,i)])
```

8. Substitute and paraphrase the definition of *Prob-Choice*:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If (Job j is allocated some slot t′ in state g)
          then If t′ = [t - Working-Time(j,i)]
                  then 1
                  else 0
          else K2 * If (Time slot [t - Working-Time(j,i)]
                        is assigned some job j′ in state g)
                      then If j = j′ then 1 else 0
                      else P-Choice(g,j,[t-Working-Time(j,i)])
```

9. Substitute the definition of *P-Choice*:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If (Job j is allocated some slot t′ in state g)
          then If t′ = [t - Working-Time(j,i)]
                  then 1
                  else 0
          else K2 * If (Time slot [t - Working-Time(j,i)]
                        is assigned some job j′ in state g)
                     then If j = j′ then 1 else 0
                     else Prob-Member-Jobs(g,j)
                          * P-Alloc(g,j,[t-Working-Time(j,i)])
```

10. The functions *Prob-Member-Jobs* and *P-Alloc* are just constants:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
       If (Job j is allocated some slot t′ in state g)
          then If t′ = [t - Working-Time(j,i)]
                  then 1
                  else 0
          else K2 * If (Time slot [t - Working-Time(j,i)]
                        is assigned some job j′ in state g)
                       then If j = j′ then 1 else 0
                       else K3 * K4
```

11. The outermost *If-Then-Else* can be moved outside of the summation.

```
Exp-Job-Value(i,g,j) =
```

```
 = K0 * K1
   If (Job j is allocated some slot t’ in state g)
      then ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
             If t’ = [t - Working-Time(j,i)]
                then 1
                else 0
      else ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
             K2 * If (Time slot [t - Working-Time(j,i)]
                        is assigned some job j’ in state g)
                       then If j = j’ then 1 else 0
                       else K3 * K4
```

12. The summation in the first branch has a non-zero term only if the allocation time *t*’ plus *Working-Time(j,i)* less than or equal to *Deadline(j,i)*:

```
Exp-Job-Value(i,g,j) =

 = K0 * K1
   * If (Job j is allocated to some slot t in state g)
        then If t + Working-Time(j,i) ≤ Deadline(j,i)
                then 1
                else 0
        else ∑ (t ≤ Deadline(j,i) in Range(Completion-Time))
               K2 * If (Time slot [t - Working-Time(j,i)]
                          is assigned some job j’ in state g)
                         then If j = j’ then 1 else 0
                         else K3 * K4
```

13. The summation in the second branch has as many non zero terms as there are time slots open in state *g* up to the scheduling deadline for job *j*:

```
Exp-Job-Value(i,g,j) =

 = If (Job j is allocated to some slot t in state g)
      then If t + Working-Time(j,i) ≤ Deadline(j,i)
              then K0 * K1
              else 0
      else [S-Deadline(j,i) + 1 - Now(g)]
           * K0 * K1 * K2 * K3 * K4

 Deadline(j,i) = Job j’s completion deadline.
 Working-Time(j,i) = The time to perform job j.
 S-Deadline(j,i) = Deadline(j,i) - Working-Time(j,i)
 Now(g) = The time of the first open slot in state g.

 K0 = Average(Range(Preconditions))
 K1 = Average(Range(Weight))
 K2 = 1
 K3 = 1/|Range(Allocation-Time)|
 K4 = 1
```

M.1.2. Analysis of *Most Critical Job First*

```
Scheduling Theory #2:    (EVL-0, EPN-0, EPC-1, PCM-0)

Heuristic:               "Most Critical Job First"
```

1. Start with the definition of *Exp-Job-Value*. Substitute the definitions of *Exp-Penalty (EPN-0)* and *Exp-Weight*, which are just constants.

```
Exp-Job-Value(i,g,j) =

  =  Exp-Preconditions(i,g,j)
     * Exp-Penalty(i,g,j)
     * Exp-Weight(i,g,j)

  =  K0 * K1 * Exp-Preconditions(i,g,j)
```

2. Substitute the definitions of *Exp-Preconditions (EPC-1)* and *Exp-Check-Precond*. The resulting expression checks all triples *(p,at,ct)* where *at* is an allocation time for for job *j*, *ct* is a completion time for precondition *j'* and *p* is a member of *{0,1}* indicating whether the preconditions of *j'* are met. The function *Prob-Prec-Comp-Alloc* computes the probability of each triple.

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1 * Π (j' in Requirements(j,i))
                 Exp-Check-Precond(i,g,j,j')

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (p in Range(Preconditions))
          Σ (ct in Range(Completion-Time))
           Σ (at in Range(Allocation-Time))
             Check-Precond(p,ct,at) *
             * Prob-Prec-Comp-Alloc(i,g,j,j',p,ct,at)
```

3. Substitute the definition of *Check-Precond*:

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (p in Range(Preconditions))
          Σ (ct in Range(Completion-Time))
           Σ (at in Range(Allocation-Time))
             (If (ct ≤ at) then p else 0)
             * Prob-Prec-Comp-Alloc(i,g,j,j',p,ct,at)
```

4. The summation over *p* can be eliminated since only those terms for which *p=1* are non zero. The summation over *ct* can be restricted to terms for which *ct≤at*, since only these are non-zero:

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (at in Range(Allocation-Time))
```

```
            Σ (ct ≤ at in Range(Completion-Time))
              Prob-Prec-Comp-Alloc(i,g,j,j',1,ct,at)
```

5. Substitute for *Prob-Prec-Comp-Alloc*, then substitute for *Prob-Prec* which is just a constant:

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (at in Range(Allocation-Time))
           Σ (ct ≤ at in Range(Completion-Time))
             Prob-Prec(i,g,j',1) *
             * Prob-Comp(i,g,j',ct)
             * Prob-Alloc(g,j,at)

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (at in Range(Allocation-Time))
           Σ (ct ≤ at in Range(Completion-Time))
             K2 *
             * Prob-Comp(i,g,j',ct)
             * Prob-Alloc(g,j,at)
```

6. Substitute for *Prob-Comp (PCM-0)*. Notice that this version assumes all jobs have constant (zero) working time.

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1
     * Π (j' in Requirements(j,i))
         Σ (at in Range(Allocation-Time))
           Σ (ct ≤ at in Range(Completion-Time))
             K2 *
             * Prob-Alloc(g,j,at)
             * Σ (t in Range(Allocation-Time))
                 If [ct = t + 0 ]
                    then Prob-Alloc(g,j',t)
                    else 0
```

7. The innermost summation can be eliminated since only the term for which $t=ct$ is non zero:

```
Exp-Job-Value(i,g,j) =

  =  K0 * K1 * (K2^|Requirements(j,i)|)
     * Π (j' in Requirements(j,i))
         Σ (at in Range(Allocation-Time))
           Σ (ct ≤ at in Range(Completion-Time))
             Prob-Alloc(g,j,at)
             * Prob-Alloc(g,j', ct )
```

8. Substitute for both occurrences of *Prob-Alloc* and then substitute for *Prob-Member-Times*, which is just a constant:

```
Exp-Job-Value(i,g,j) =
```

```
=  K0 * K1 * (K2^|Requirements(j,i)|)
   * Π (j' in Requirements(j,i))
       Σ (at in Range(Allocation-Time))
         Σ (ct ≤ at in Range(Completion-Time))
           {If (Job j is allocated time t in state g)
               then If (t = at) then 1 else 0
               else Prob-Member-Times(g,at)
                    * Prob-Choice(g,j,at)}
           * {If (Job j' is allocated time t in state g)
                 then If (t = ct) then 1 else 0
                 else Prob-Member-Times(g,ct)
                      * Prob-Choice(g,j',ct)}

=  K0 * K1 * (K2^|Requirements(j,i)|)
   * Π (j' in Requirements(j,i))
       Σ (at in Range(Allocation-Time))
         Σ (ct ≤ at in Range(Completion-Time))
           {If (Job j is allocated time t in state g)
               then If (t = at) then 1 else 0
               else K3 * Prob-Choice(g,j,at)}
           * {If (Job j' is allocated time t in state g)
                 then If (t = ct) then 1 else 0
                 else K3 * Prob-Choice(g,j',ct)}
```

9. Substitute for *Prob-Choice* and then for *Prob-Member-Jobs* and *P-Alloc* which are just constants: which is just a constant:

```
Exp-Job-Value(i,g,j) =

=  K0 * K1 * (K2^|Requirements(j,i)|)
   * Π (j' in Requirements(j,i))
       Σ (at in Range(Allocation-Time))
         Σ (ct ≤ at in Range(Completion-Time))
           {If (Job j is allocated time t in state g)
               then (If (t = at) then 1 else 0)
               else K3 * [If (Slot at is assigned job j''
                              in state g)
                             then If j = j'' then 1 else 0
                             else K4 * K5]
           * {If (Job j' is allocated time t in state g)
                 then (If (t = ct) then 1 else 0)
                 else K3 * [If (Slot ct is assigned job j''
                                in state g)
                               then If j' = j'' then 1 else 0
                               else K4 * K5]
```

10. The embedded *If-Then-Else* statements can be moved outside of the summations. They can also we rewritten as a case statement:

```
Exp-Job-Value(i,g,j) =

= K0 * K1 * (K2^|Requirements(j,i)|)
  * Π (j' in Requirements(j,i))

      CaseA: Jobs j & j' are allocated at t & t' in g.
             Σ (at in Range(Allocation-Time))
```

```
            ∑ (ct ≤ at in Range(Completion-Time))
              [If (t = at) then 1 else 0]
              * [If (t' = ct) then 1 else 0]

CaseB: Only job j is allocated (at time t) in g.
          ∑ (at in Range(Allocation-Time))
            ∑ (ct ≤ at in Range(Completion-Time))
              K3 * [If (t = at) then 1 else 0]
              * [If (Slot ct is assigned job j'' in g)
                    then [If j' = j'' then 1 else 0]
                    else K4 * K5]

CaseC: Only job j' is allocated (at time t') in g.
          ∑ (at in Range(Allocation-Time))
            ∑ (ct ≤ at in Range(Completion-Time))
              K3 *
              * [If (Slot at is assigned job j'' in g)
                    then [If j = j'' then 1 else 0]
                    else K4 * K5]
              * [If (t' = ct) then 1 else 0]

CaseD: Neither j nor j' is allocated in g.
          ∑ (at in Range(Allocation-Time))
            ∑ (ct ≤ at in Range(Completion-Time))
              [K3^2]
              * [If (Slot at is assigned job j'' in g)
                    then [If j = j'' then 1 else 0]
                    else K4 * K5]
              * [If (Slot ct is assigned job j'' in g)
                    then [If j' = j'' then 1 else 0]
                    else K4 * K5]
```

11. Consider each of the four cases:
    a. *Case-A*: Both job *j* and precondition *j*' are allocated, at times *t* and *t*', in state *g*. The summation is taken over all pairs such that *ct*≤*at*. Only the terms for which *ct*=*t*' and *at*=*t* can be non zero. The summation has a (single) non zero term only if *t*'≤*t*.
    b. *Case-B*: Only job *j* is allocated, at time *t*, in state *g*. The summation is taken over all pairs such that *ct*≤*at*. Only the terms for which *at*=*t* can be non zero. Since time slots are allocated in order, every *ct*≤*t* is assigned a job *j''* that is distinct from *j'*. These terms must therefore be zero as well. The summation thus has no non zero terms.
    c. *Case-C*: Only precondition *j'* is allocated, at time *t*', in state *g*. The summation is taken over all pairs such that *ct*≤*at*. Only the terms for which *ct*=*t*' can be non zero. Only the unassigned slots *at*≥*t'* will lead to non zero terms. Since time slots are allocated in order, this must include all free slots. The summation thus has one non zero term for each free time slot. If *Slots(g)* is the number of slots open in state *g*, then the summation is proportional to *Slots(g)*.
    d. *Case-D*: Neither job *j* nor precondition *j'* is allocated in state *g*. The summation is taken over all pairs such that *ct*≤*at*. Only the unassigned pairs will be non zero. The summation thus has one non zero term for each pair of open time slots. If *Slots(g)* is the number of slots open in state *g*, then the summation is proportional to *Slots(g)* choose two.

```
Exp-Job-Value(i,g,j) =

 =  K0 * K1 * (K2^|Requirements(j,i)|)
    * Π (j' in Requirements(j,i))
        Case1: Jobs j and j' are allocated at t and t' in g.
```

```
        Return {If t’ ≤ t then 1 else 0}
Case2: Only job j is allocated (at time t) in g.
        Return 0
Case3: Only job j’ is allocated (at time t’) in g.
        Return K3 * K4 * K5 * Slots(g)
Case4: Neither j nor j’ is allocated in g.
        Return {(K3*K4*K5)^2}*Slots(g)*(Slots(g)-1)/2.
```

```
Definitions:

 Requirements(j,i) = The preconditions of job j.
 Slots(g) = The number of open time slots in state g.

 K0 = Average(Range(Penalty))
 K1 = Average(Range(Weight))
 K2 = 1 / 2
 K3 = 1
 K4 = 1
 K5 = 1 / |Range(Allocation-Time)|
```

M.2. Assumptions used in Approximate Scheduling Theories

1. *Equiprobable:Numeric:* Applied assume all *Preconditions* are as likely to be met as unmet.
2. *Equiprobable:Numeric:* Applied to the *Penalty* factor of each job.
3. *Equiprobable:Numeric:* Applied to the *Weight* of each job.
4. *Equiprobable:Boolean:* Applied to the fact of whether (recursive) *Preconditions* of a precondition are met.
5. *Equiprobable:Equality:* Applied to assume *Allocation-Time* of a job *j* is equally likely to be time *t* in *Range(Allocation-Time)* if the allocation time of *j* is not known, and if the job allocated to time *t* is not known.
6. *Functional-Invariance:Constant:* Applied to assume (correctly) that any time *t* is in the set *Range(Allocation-Time)*.
7. *Functional-Invariance:Constant:* Applied to assume (correctly) that any job *j* is in the set *Range(Choice)*.
8. *Independence-of-Product:Binary:* Applied to assume independence of (a) satisfying *Preconditions*, (b) the *Penalty* factor and (c) the *Weight* of a job. (Applied Twice.)
9. *Independence-of-Product:Nary:* Applied to assume independence of the fact of satisfying each precondition of a job.
10. *Independence:Three-Conjuncts:* Applied to assume independence of (a) recursive *Preconditions* of a precondition, (b) the *Completion-Time* of a precondition and (c) the *Allocation-Time* of a job.
11. *Independence:Two-Conjuncts:* Applied to assume independence of (a) whether the *Choice* at some time *t* was a job *j*, and whether time *t* is a member of *Range(Allocation-Time)*.
12. *Independence:Two-Conjuncts:* Applied to assume independence of (a) whether the *Allocation-Time* of some job *j* is time *t*, and whether job *j* is a member of *Range(Choice)*.
13. *Functional-Invariance:Constant:* Applied to assume that all jobs have the same *Working-Time*.

Scheduling Theory #1:	**(EVL-0, EPN-1, EPC-0, PCM-1)**
Heuristic:	**"Soonest Scheduling Deadline First"**
Assumptions:	**1,3,5,6,8a,8b,11.**

Scheduling Theory #2:	**(EVL-0, EPN-0, EPC-1, PCM-0)**
Heuristic:	**"Most Critical Job First"**
Assumptions:	**2,3,4,5,6,7,8a,8b,9,10,11,12,13.**

Junk Page.

Table of Contents

List of Figures