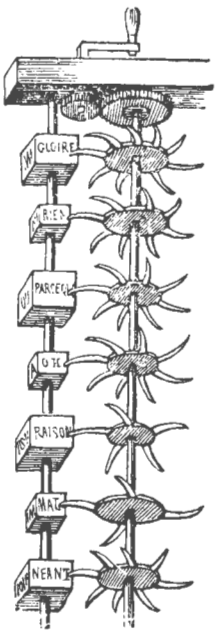
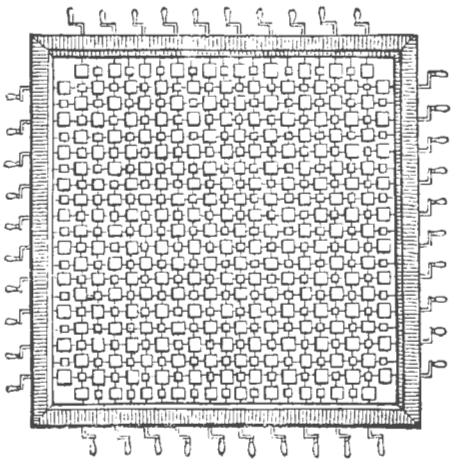
Canadian Society for Computational Studies of Intelligence

Société Canadienne pour Etudes d'Intelligence par Ordinateur





Proceedings of First CSCSI/SCEIO National Conference

University of British Columbia
Vancouver, B. C.

August 25 - 27, 1976

PROCEEDINGS

OF

FIRST CSCSI/SCEIO NATIONAL CONFERENCE

Sponsored by: Canadian Society for Computational Studies of Intelligence
/Société Canadienne pour Etudes d'Intelligence par Ordinateur

and Department of Computer Science
University of British Columbia

University of British Columbia
Vancouver, B. C., Canada

August 25 - 27, 1976

Additional copies of this volume may be purchased by prepaid order directed to CSCSI/SCEIO at the following address:

CSCSI/SCEIO
c/o Alan Mackworth/Richard Rosenberg
Department of Computer Science
University of British Columbia
Vancouver, B. C.
Canada V6T 1W5

Price (including postage): $8.00

Cover Illustration by Grandville [Jean Gérard]

The first professor I saw was in a very large room, with forty pupils about him. After salutation, observing me to look earnestly upon a frame, which took up the greatest part of both the length and breadth of the room, he said perhaps I might wonder to see him employed in a project for improving speculative knowledge by practical and mechanical operations. But the world would soon be sensible of its usefulness, and he flattered himself that a more noble exalted thought never sprang in any other man's head. Every one knew how laborious the usual method is of attaining to arts and sciences; whereas, by his contrivance, the most ignorant person at a reasonable charge, and with a little bodily labour, may write books in philosophy, poetry, politics, law, mathematics, and theology, without the least assistance from genius or study. He then led me to the frame, about the sides whereof all his pupils stood in ranks. It was twenty foot square, placed in the middle of the room. The superficies was composed of several bits of wood, about the bigness of a die, but some larger than others. They were all linked together by slender wires. These bits of wood were covered on every square with paper pasted on them, and on these papers were written all the words of their language, in their several moods, tenses, and declensions, but without any order. The professor then desired me to observe, for he was going to set his engine at work. The pupils at his command took each of them hold of an iron handle, whereof there were forty fixed round the edges of the frame and giving them a sudden turn, the whole disposition of the words was entirely changed. He then commanded six and thirty of the lads to read the several lines softly as they appeared upon the frame; and where they found three or four words together that might make part of a sentence, they dictated to the four remaining boys who were scribes. This work was repeated three or four times, and at every turn the engine was so contrived, that the words shifted into new places, as the square bits of wood moved upside down.

Six hours a day the young students were employed in this labour, and the professor showed me several volumes in large folio already collected, of broken sentences, which he intended to piece together, and out of those rich materials to give the world a complete body of all arts and sciences; which however might be still improved, and much expedited, if the public would raise a fund for making and employing five hundred such frames in Lagado, and oblige the managers to contribute in common their several collections.

He assured me that this invention had employed all his thoughts from his

youth, that he had emptied the whole vocabulary into his frame, and made the strictest computation of the general proportion there is in books between the numbers of particles, nouns, and verbs, and other parts of speech.

I made my humblest acknowledgment to this illustrious person for his great communicativeness, and promised if ever I had the good fortune to return to my native country, that I would do him justice, as the sole inventor of this wonderful machine; the form and contrivance of which I desired leave to delineate upon paper, as in the figure here annexed. I told him, although it were the custom of our learned in Europe to steal inventions from each other, who had thereby at least this advantage, that it became a controversy which was the right owner, yet I would take such caution, that he should have the honour entire without a rival.

Jonathan Swift, Gulliver's Travels

Preface

This is the first national conference of the Canadian Society for Computational Studies of Intelligence/Société Canadienne pour Etudes d'Intelligence par Ordinateur. It is preceeded by two workshops: the first held at the University of Western Ontario on May 23-25, 1973; the second in Ottawa, Ontario on May 28-29, 1975. The success of these workshops inspired us to organize a more formal conference. The fact that the first national conference is being held in Vancouver demonstrates that we do constitute a truly national society. We hope that this conference will be followed by biennial national conferences at other centres across the country.

We are most grateful to the following people: John Peck, Acting Head of the Department of Computer Science, U.B.C., and the rest of the department, faculty, staff and students, for financial and administrative support, in particular Heather Johnson, Emmy Mills and Sheila Whitt for their uncomplaining, never-failing help; our referees, Ted Elcock of the University of Western Ontario, John Mylopoulos of the University of Toronto, Bonnie Nash-Webber of Bolt, Beranek and Newman, and Ray Reiter of U.B.C. and Bolt, Beranek and Newman for being prompt and generous with their counsel; and Zenon Pylyshyn for his invited address to the conference, "Recent Trends in Artificial Intelligence".

Richard Rosenberg — General Chairman

Alan Mackworth — Programme Chairman

CONTENTS

* This paper was not available at time of initial printing. It may appear at the end of this volume.

EXPERIMENTS IN SPEECH UNDERSTANDING SYSTEM CONTROL

William H. Paxton

Artificial Intelligence Center
Stanford Research Institute
Menlo Park, California 94025

ABSTRACT

A series of experiments was performed concerning control strategies for a speech understanding system. The main experiment tested the effects on performance of four major choices: focus attention by inhibition or use an unbiased best-first method, "island-drive" or process left to right, use context checks in priority setting or do not, and map words all at once or map only as called for. Each combination of choices was tested with 60 simulated utterances of lengths varying from 0.8 to 2.3 seconds. The results include analysis of the effects and interactions of the design choices with respect to aspects of system performance such as overall sentence accuracy, processing time, and storage. Other experiments include tests of acoustic processing performance and a study of the effects of increased vocabulary and improved acoustic accuracy.

INTRODUCTION

This paper reports a series of experiments concerning control strategies for a speech understanding system.* The basic goal of the system was to perform a data-base management task using input in the form of continuous speech rather than isolated words, and simple English rather than an artificial command language.

One of the major problems in developing the system was to design a framework both to integrate the components and to provide an overall control strategy. For a speech system, the choice of a control strategy is particularly important because "false alarms," words incorrectly accepted as occurring in the input when they are not really there, present many opportunities to make mistakes. Rather than picking a particular control strategy, we designed the system framework so that it was possible to test the

---

* This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and monitored by the U.S. Army Research Office under Contract No. DAAG29-76-C-0011. The system used in the experiments was developed jointly by Stanford Research Institute (SRI) and System Development Corporation (SDC).

effects of several major design choices. The results of these tests are reported below. Details regarding the system framework are presented elsewhere (Paxton 1976 and forthcoming).

Several experiments were performed. Information regarding the acoustic processing was gathered in the first experiment. As well as being of interest in its own right, this information was used in simulating the acoustic processing for the other experiments. The second experiment dealt with the "fanout," both for the language alone and in combination with the acoustics. Fanout provides a quantitative measure of the difficulty of the task related to the average number of alternatives confronting the system. The third experiment measured the performance of two special test systems with extremely simple designs. In the fourth experiment, the main experiment of the series, the standard speech system was measured on a set of 60 test sentences for all combinations of 4 control strategy design choices. The performance of the best configuration from Experiment 4 was tested in the fifth experiment, allowing different sizes of gaps and overlaps between words in the simulated acoustic processing. The sixth and last experiment studied the performance of the most promising system configurations from Experiment 4, while varying vocabulary size and acoustic processing accuracy.

This paper will not discuss all of the experiments in detail. We will sketch the experiments and the main results with emphasis on the fourth experiment, concerning control-strategy design alternatives. A detailed discussion of the entire series of experiments will be given in Paxton (forthcoming).

EXPERIMENT 1--MAPPER PERFORMANCE

The first experiment deals with the performance of the system component called the "mapper" (described in Ritea, 1975). The mapper carries out acoustic tests. Given a predicted word and location in the input, the mapper either rejects the word, or accepts it and reports its beginning and ending boundaries rounded to the nearest 0.05 second. If the word is accepted, the mapper also gives it a score between 0 and 100, indicating how well it matches the input (100 indicates a perfect match). Words accepted by the mapper are either "hits," words really in the input sentence, or false alarms, words accepted although not in the input.

The mapper was tested by calling it for all of the words in the vocabulary at the start of an utterance and then at each position where a previously accepted word ended. This procedure resulted in testing the entire vocabulary at an average of about 16 out of 20 positions per second of speech (recall that word boundaries are rounded to multiples of .05 seconds, so there are 20 possible ending positions per second).* Overall, tests were made at 180 positions in 11 test utterances. For the 305-word vocabulary used in the following experiments, the mapper had 48 hits and 1564 distinct false alarms. The false alarms were distributed throughout the vocabulary (229 of the 305 words (75%) were falsely accepted at least once), with small words like "a" and "the" each accounting for more than 30 false alarms.

The false alarm rate for the mapper was determined by counting the number of false alarms that fell within a section of the input. For the 305 word vocabulary, the average rate was 114 false alarms per second of speech. Since there were about 3 hits per second of speech, this rate indicates that the mapper produced an average of almost 40 false alarms for each hit. As partial compensation for this result, there were no "misses" (cases in which the mapper failed to accept a correct word) and the mean hit score was higher than the mean false alarm score (73.5 versus 59.4), although both score distributions spread over the entire range, from near 100 down to the threshold of 45.

The remaining experiments use a simulation of the mapper based on the data gathered in the first experiment. To simulate the performance of the mapper on a particular sentence, the words of the sentence were first assigned lengths in seconds of speech. Each word was then assigned a score picked at random from the hit scores actually produced by the mapper. The words were concatenated to determine the length of the utterance, and the length was multiplied by the false alarm rate (114 false alarms per second of speech) to give the total number of false alarms to be simulated. The false alarms were

---

* This experiment was originally designed to record the results of all mapper calls that might be made in a left to right parse. The intention was to use this information in place of the mapper in tests of the entire system. However, technical difficulties made it impossible to gather enough information to satisfy the original goal. If the original goal had been to provide data for a simulation of the mapper, the mapper would have simply been tested on the entire vocabulary across each utterance at .05 second intervals. The change in goals may have resulted in slightly underestimating the false alarm rate for the mapper because of the untested positions where no word ended.

selected randomly from the 1564 false alarms produced by the mapper, and then positioned randomly in the sentence. As a check on our simulation, we calculated the correlations between the observed score distributions and the score distributions in the simulated utterances used in the following experiments. The hit score distributions had a .97 correlation, and the false alarm distributions had a .996 correlation. (There were many more false alarms, hence the higher correlation.) In computing the simulated processing time for the mapper in later experiments, we used figures of 0.25 seconds processing per word tested, 1.0 seconds per position of initial processing before trying any words, and 10.0 seconds per second of speech in the sentence if "island-driving" was being simulated (see discussion of Experiment 4). These figures are derived from rough measurements of the mapper running on an IBM System/370 Model 145.

EXPERIMENT 2--FANOUT

The second experiment dealt with the fanout in the language with and without acoustic constraints. "Fanout" is defined as the number of words that can be successfully appended to an initial substring of some sentence, to produce either a complete sentence or a string that can potentially be completed to form a sentence. The average fanout over a large number of initial substrings provides a measure of the uncertainty of each word, as indicated by the number of alternatives open to the system.

The fanout was measured for 11 sentences, together containing a total of 67 words. The fanout was measured only for initial substrings of the actual sentences; it was not measured along false paths. The distribution of the size of the fanout was bimodal. Using the 305-word vocabulary and ignoring acoustic constraints, 24 positions (36%) had a fanout of less than 30 words, while 33 positions (49%) had a fanout of more than 173 words. The small fanout positions were places allowing only vocabulary classes with a small number of members, classes such as preposition or verb. The large fanout positions corresponded to places where a noun could be expected. The mean fanout was 117, with a standard deviation of 90 and a maximum of 219. The fanout at the beginning of sentences was 206.

The fanout with acoustic constraints is based on the simulated mapper data. It is calculated by counting the number of words: (a) that are accepted by the simulated mapper at a position starting plus or minus 0.05 seconds from

the end of an initial substring of hits, and (b) that are also in the fanout set without acoustic constraints for that substring. In addition to recording the size of the fanout, we ordered the set of words by mapper scores and computed the rank of the hit. For example, if 2 false alarms had scores higher than the hit, the rank of the hit would be 3. For the 305 word vocabulary, the mean fanout with speech was 18, and the average hit rank was 3.7. The fact that the hit rank is smaller than half of the fanout reflects the previously mentioned difference between the score distributions for hits and false alarms.

The results of this experiment help to show why the control strategy problem for speech understanding is so difficult. The results suggest that, on the average, there will be between 2 and 3 false alarms with higher scores than the actual hit to tempt the system down false paths. Luckily, there are compensating factors tending to bring the system back to the correct path. One aid is that the fanout following a false alarm is probably smaller than the fanout following a hit, and false paths thus often lead to dead ends. (This speculation needs to be tested empirically.) The decrease in fanout is most pronounced near the boundaries of an utterance, where many words are eliminated because their minimum duration is greater than the available time. Similarly, false paths may be impossible to complete because too much speech remains. For example, a path will be a dead end if it requires a one-syllable word to fill a four-syllable section of the input. Finally, even if there are complete false paths, the system may still get the sentence right if the correct path is found and is given a higher overall score than any incorrect path. The difference in hit and false alarm score distributions makes this more likely. These factors, and perhaps others not yet recognized, may offset the effect of the large number of high scoring false alarms, but speech understanding is still a difficult task, as indicated by the results of the next experiment.

EXPERIMENT 3--TWO (TOO) SIMPLE SYSTEMS

In the third experiment, two systems with extremely simple designs were tested on a set of 60 sentences. The sentences covered a wide range of vocabulary and included questions, commands, and elliptical sentence fragments. There were 10 sentences at each length of simulated speech from 0.8 to 2.3 seconds at intervals of 0.3 seconds. The sentences averaged 5.9 words in length, with a maximum of 9 words.

The first system tested used a dynamic programming method to find the sequence of words accepted by the simulated acoustic processing with the highest cumulative score. The sequence was constrained to start and end within 0.05 seconds of the utterance boundaries, and to have between-word gaps and overlaps of no more than 0.05 seconds. There were no syntactic or semantic constraints on the word sequences chosen. This system selected the complete correct sequence of words in 3 sentences (5%) and found 86 of the 323 hits (27%).

The second test system used a context-free parsing algorithm to find the sequence of words with the best cumulative score that met the same gap and overlap constraint of 0.05 seconds and also satisfied some simple phrase structure rules for a subset of English.* The additional constraint eliminated many of the false paths open to the first test system, but the algorithm still selected the correct sequence in only 6 cases (10%) and found only 100 hits (31%).

The poor performance of these simple test systems tends to confirm the need for more extensive linguistic knowledge (or large improvements in acoustic processing accuracy) and helps to justify the greater complexity of the standard system, which, as the following experiments show, is able to achieve much better results.

EXPERIMENT 4--CONTROL STRATEGY DESIGN CHOICES

In the fourth experiment, the performance of the standard speech system was measured on the 60 test sentences described above, while varying four major control-strategy design choices. The choices used as experimental variables were the following:

Island-Drive or Not -- Go in both directions from arbitrary starting points in the input versus proceed strictly left to right from the beginning: Island-driving allows the system to use words that match well anywhere in an input and to build up an interpretation around them. Left to right processing is simpler and less flexible but may still be more accurate and efficient than island-driving.

---

* The rules were taken from the language definition for the standard speech system used in the later experiments.

Map All or One -- Test all the words at once at a given location versus try them one at a time and delay further testing when a good match is found: Mapping all at once lets the system know the best candidates from the acoustics and reduces the chances of following a false path. Mapping one at a time avoids exhaustive testing and will be more efficient than mapping all at once if the system does not encounter too many false alarms.

Context Checks -- Take into account the restrictions of the possible sentential contexts as part of setting priorities versus ignore the contextual restrictions except for use in eliminating already formed structures: Context checking should give more information for setting priorities and should lead to better predictions. However, the checks can be expensive and therefore may not lead to an overall improvement in performance.

Focus by Inhibition -- Focus the system on selected alternatives by inhibiting competition versus employ an unbiased best-first strategy: Focusing allows the system to concentrate on a particular set of potential interpretations rather than thrashing among a large number of alternatives. However, if the focus of attention is too often wrong, the net effect may be harmful to system performance.

All combinations of the 4 control-strategy variables were tested on the 60 sentences. This experimental design allows us to compare the 16 combinations of control choices and to evaluate, by analysis of variance, the main effects and interactions of the control strategy variables. The main effect of a variable is the change in performance it produces, averaged over all the possibilities for the other variables. The interaction of two variables tells whether the effect of one variable is the same for all possibilities of the other. The interaction of three variables tells whether the interaction of two of them is the same for all possibilities of the third, and so on. Analysis of variance is a statistical technique for computing the probability that the observed effects or interactions are really caused by the experimental variables, rather than the result of random variation (see e.g. Winer 1971). In other words, this method aids in evaluating results of experiments influenced by substantial random factors. In our case, the random factors include the random choices of false alarms and hit scores in simulating the mapper, and the selection of a particular sample of sentences

from the much larger population of possible sentences. The statistical results for a main effect or interaction are given in a form such as "F(1,5)=6.9, p < .05." This means that the F ratio (a statistic for comparing variances) for the effect or interaction has 1 and 5 degrees of freedom and has a value equal to 6.9. This in turn implies that the probability is less than .05 that the observed effect or interaction was caused by random variation alone. If the probability is given by itself in the following discussion, it is based on the these values: F(1,5)>=16.3 for p < .01, F(1,5)>=6.6 for p < .05, and F(1,5)>=4.1 for p < .10.

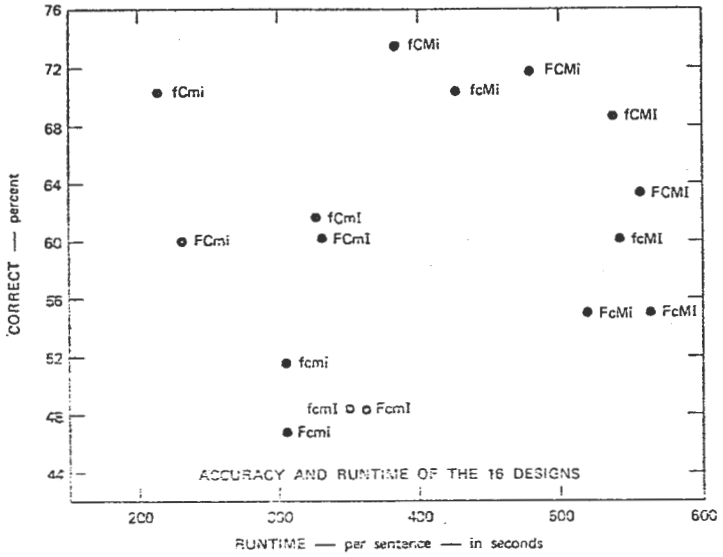


Figure 1. Accuracy and Runtime

The most important performance measures for the system are accuracy (the percentage of sentences for which the correct sequence of words is found) and runtime (the computation required by the system, including simulated acoustic processing time). For these measures, the control strategy variables had large, significant effects. Before discussing the effects, we need to introduce a notation for naming the experimental designs. The capital letter "F" will refer to focus by inhibition, lower case "f" for no focus by inhibition; "C" stands for context checks, "c" for no context checks; "M" for map all at once, "m" for not map all at once; "I" for island-driving, and "i"

for no island-driving. This notation will indicate the different combinations of choices. For example, "fCMi" refers to the system that does not use focus by inhibition, does use context checks, does map all at once, and does not island-drive.

Using this notation, Figure 1 shows the accuracy and runtime of the 16 experimental systems. Notice the range of values for both measures, from 46.7% to 73.3% for accuracy, and from 221 to 559 seconds processing per sentence for runtime. These wide ranges confirm the importance of control strategy in determining system performance. With respect to the individual control variables, comparing the C-systems to the correspondng c-systems shows that context checks for priority setting result in better accuracy and faster runtimes. Similar comparisons show that mapping all at once improves accuracy but increases runtime, while focus by inhibition and island-driving both reduce the accuracy and increase the runtime. In the course of this paper, we discuss these effects and propose explanations for them.

Table 1
MAIN EFFECTS OF VARIABLES ON PERCENT CORRECT

	WITH	WITHOUT	DIFFERENCE	
F	57.5	62.9	-5.4	*
C	66.0	54.4	11.6	*
M	64.6	55.8	8.8	*
I	58.1	62.3	-4.2	

* p < 0.05

Table 2
FOCUS AND ISLAND-DRIVING INTERACTION

	I	i	I-i
F	56.7	58.3	-1.6
f	59.6	66.3	-6.7
F-f	-2.9	-8.0	5.1

Table 1 shows the effect of the control variables on accuracy. For the purposes of analysis of variance, we pooled the results on the 10 sentences of equal length to get 6 accuracy measures per system. The interaction with length was then used as the error term for calculating statistical significance. Results are reported for analysis of the raw percentages; analysis after an arcsin transformation to improve homogeneity of variance was also performed and gave the same levels of significance. As inspection of Figure 1 suggests, context checks and map all improve accuracy, while focus and island-driving make it worse. The island-driving effect was not significant statistically because of a large interaction with sentence length. For the long sentences, 1.7 to 2.3 seconds, island-driving decreased accuracy by 15.8%, but for the short ones, 0.8 to 1.4 seconds, it actually increased accuracy by 7.5%. There was a significant interaction ($p < 0.05$) between

focus and island-driving. As shown in Table 2, the effect of island-driving is less with focus, and the effect of focus is less with island-driving. To explain this collection of results we must first consider how accuracy is influenced by control strategy.

The control strategy affects accuracy indirectly. All the strategies are "complete" in the sense that they only reorder, and never eliminate, alternatives. If there were no false alarms, all the systems would get 100% of the test sentences correct. Even with false alarms, the strategies would get an equal percent correct, if all the possible alternatives could be tried before the system picked an interpretation. Errors would only occur when false alarms had high enough scores to displace hits in the highest rated interpretations. However, in the actual system, the large number of alternatives makes it impossible to consider all of them in the space and time available. As a result, the order in which the alternatives are considered can affect the accuracy, and so can the demands on space and time. Control strategy thus affects accuracy indirectly by reordering alternatives and by modifying space and time requirements. To explain the accuracy effects, we must look at these other factors.

In this experiment, the storage limit was an important factor for accuracy. In the 960 tests (60 sentences times 16 systems), 578 (60.2%) were correct and 383 (39.8%) were wrong. Of the errors, 175 (46%) had an incorrect interpretation, while 207 (54%) had no interpretation at all. Since the systems could potentially get the correct answer, and no time limit was imposed until at least one interpretation had been found, all of the 207 sentences with no interpretation were a result of running out of storage.

The storage limit used in the tests was based on the number of phrases constructed. When the total reached 500, the system would stop trying new alternatives and, if any interpretation had been found, pick the highest rated interpretation as its answer. The average number of phrases constructed was 204 nonterminal and 63 terminal. The system with the best accuracy, fCMi, had the lowest average (113 nonterminals and 45 terminals), while the system with the worst accuracy, Fcmi, had one of the highest averages (260 nonterminals and 63 terminals). Overall, there was a strong negative correlation (-.93) between system accuracy and average number of phrases constructed.

Table 3
MAIN EFFECTS ON STORAGE
(number of phrases)

	WITH	WITHOUT	DIFFERENCE	
F	281	253	28	*
C	240	294	-54	**
M	244	290	-46	**
I	287	247	40	

** $p < .01$ * $p < .05$

Table 3 shows the effects of the control variables on the number of phrases. The pattern is the same as for accuracy; context checks and map all have good effects, while focus and island-driving have bad effects. Again, because of a large interaction with length, the island-driving effect is not significant statistically. There are significant interactions, $p < 0.05$, between focus and island-driving for storage, as seen in Table 4, and between context checking and mapping all at once, as seen in Table 5.

Table 4
FOCUS AND ISLAND-DRIVING INTERACTION
(number of phrases)

	I	i	I-i
F	290	272	18
f	284	222	62
F-f	6	50	-44

Table 5
CONTEXT AND MAP-ALL INTERACTION
(number of phrases)

	M	m	M-m
C	221	259	-38
c	267	322	-55
C-c	-46	-63	17

The beneficial effect of mapping all at once on the storage requirements and accuracy is caused by a reduction in the proportion of false alarm terminal phrases. Mapping all at once significantly reduces the proportion of terminal phrases that are false alarms--from 88.0% to 85.7%, $p < .01$. The false terminal proportion is in turn significantly correlated with the number of phrases (.72) and the accuracy (-.75). When the words are all mapped at once, the system is able to take advantage of the difference in false alarm and hit score distributions to reduce the likelihood of constructing false terminal phrases. Notice that a relatively small change in false terminal percentage has a large effect on system performance.

Surprisingly, context checking also results in a significant reduction in the false terminal percentage--from 87.5% to 86.2%, $p < .01$. This reduction may be evidence that context checking is giving lower priority to looking for words adjacent to false alarms than it gives to looking next to hits. This

change could affect the false terminal likelihood, since there is always a hit adjacent to a hit, while false alarms often have nothing but other false alarms next to them. In addition to its effect on false terminals, context checking may also be improving the storage requirements and accuracy by generally improving the priority setting, thereby reducing the likelihood of following false paths.

Focus by inhibition slightly increases the proportion of false alarm terminal phrases (from 86.3% to 87.3%), but this increase is too small to be statistically significant. The explanation of the ill effects of focus is essentially the converse of the explanation of the effects of context checking. Context checking makes performance better by improving priorities, while focus makes it worse by distorting priorities. Focus too often changes priorities to bias the system in favor of a false alarm instead of a hit. Overall, focus changed priorities in favor of a false alarm 112 times per sentence and in favor of a hit, only 15 times per sentence. Thus the priorities, and the system performance, were better with the unbiased best-first strategy than with focus by inhibition.

Island-driving did not affect the false terminal proportion, but it did have bad effects on storage and accuracy for the longer sentences. To get a sentence correct, island-driving must start at least one island with a hit. If all the islands are false alarms, the sentence will not be interpreted correctly. The overall average was 3.7 false alarm islands per sentence and 0.9 hit islands. There were one or more hit islands in 82% of the tests using island-driving. The bad effects of island-driving on long sentences was not caused by a greater likelihood of false alarm islands. The average rank of the first hit in the sequence of words for use in forming islands was 4.8, and this did not increase with sentence length. (The correlation between rank and length was .04). For sentences 1.7 seconds or longer, instead of an increase in the number of islands necessary to get a hit, there was an increase in the amount of storage consumed per island. Perhaps the greater length allowed islands to grow in both directions, whereas in shorter sentences the sentence boundaries blocked one direction or the other.

The interaction of focus and island-driving can be explained as the result of the storage limit. The limit put a ceiling on the size of the possible combined effect. Thus the combined effect was less than the sum of the individual effects. Similarly, the interaction between context checking

and mapping all at once is a result of overlapping good effects, which consequently fail to add. The same pattern of context and map-all interaction appears in false terminal proportion, $p < .05$, and in accuracy, $F=4.00$ versus $F(1,5)=4.06$ for $p < .10$.

We now turn to a brief analysis of the sentences that got one or more interpretations, but were incorrect because their highest rated interpretation was wrong. As mentioned previously, this happened in 175 tests. In 109 of these (62%), the chosen interpretation was reasonable linguistically but contained incorrect words. In 10 tests (6%), the chosen interpretation could have been eliminated by a better language definition ("Was feet one builder of the Farragut?" is an example from these 10). Finally, 56 of the errors (32%) were harmless, in that the system would probably produce the same answer as if it had found the correct sequence of words (e.g., "What reactor does it have?" instead of "What reactors does it have?" was one of these harmless errors). If the harmless errors are counted as correct in calculating the accuracy, the most accurate system, fCMi, increases in percent correct from 73.3% to 80.0%, and the average accuracy for all the systems goes up 5.8%. The ten to one preponderance of linguistically reasonable errors over linguistically bad errors suggests that, although there is still room for improvement in the language definition, the major way to improve accuracy is to reduce the number of high scoring false alarms. The effects of such acoustic improvements are explored in Experiment 6.

The accuracy effects have been explained in terms of storage requirements, proportions of false terminal phrases, and priorities. The important role of storage limitations raises the question of whether the accuracy effects would have disappeared if more storage had been available. I speculate that the effects would have been smaller but still important. The effects on the proportion of false terminal phrases would remain, as would the presumed effects on priorities. Moreover, even if the storage limit were relaxed, the limit on runtime would remain to penalize inefficient systems. The effects of control strategy choices on accuracy would only vanish if space and runtime limitations were both removed, a most unlikely event in view of the current performance of speech understanding systems.

The system runtime is another important performance measure. Here, we will use the phrase "total runtime" to refer to the simulated acoustic processing, plus the actual processing time (on a DEC PDP KA-10) for the

executive and the semantic components. The executive time is mainly spent setting priorities and parsing. The semantics time is used in constructing semantic translations and in dealing with anaphoric references and ellipsis. The reported total runtime does not include acoustic preprocessing or question answering, since neither are affected by the experimental variables. In analysis of variance of the runtimes, interaction with length was used as the error term, and significance levels were confirmed by reanalysis after a square root transformation to improve homogeneity of variance.

The main effects of the control variables on total runtime are given in Table 6. All except context checking increase the runtime. Partitioning the sentences into a short group (0.8 to 1.4 seconds) and a long group (1.7 to 2.3 seconds) shows that island-driving has a much worse effect on runtime for long than for short sentences. For short sentences, island-driving increased the mean runtime from 262 to 290 seconds, a difference of 28. For long sentences, the increase was from 457 to 598 seconds, a difference of 141. Recall that island-driving also had worse effects for long sentences on accuracy and storage.

Table 6
MAIN EFFECTS ON TOTAL RUNTIME
(seconds per sentence)

	WITH	WITHOUT	DIFFERENCE	
F	417	386	31	*
C	383	421	-38	**
M	498	305	193	**
I	444	359	85	#

* p < .05 ** p < .01 # p < .10

Table 7
EFFECTS ON EXECUTIVE RUNTIME
(seconds per sentence)

	WITH	WITHOUT	DIFFERENCE	
F	120	106	14	**
C	109	117	-8	#
M	90	135	-45	**
I	127	98	29	#

Table 8
EFFECTS ON ACOUSTICS RUNTIME
(seconds per sentence)

	WITH	WITHOUT	DIFFERENCE	
F	276	260	16	#
C	254	282	-28	**
M	389	147	242	**
I	295	241	54	#

** p < .01 * p < .05 # p < .10

Tables 7 and 8 show the main effects on executive runtime and simulated acoustics runtime respectively. In both cases, context checks decrease the runtime, while focus and island-driving increase it. Mapping all at once

improves the executive runtime but leads to a huge increase in acoustic processing time. As usual, examination of the results according to sentence length shows that island-driving is worse for longer sentences. Since the average executive and acoustic times together account for 95% of the average total, we do not report separate effects for semantics.

Analysis of variance for total, executive, and acoustic runtimes reveals a significant interaction between context checking and mapping all at once ($p < .01$ for total and acoustics; $p < .05$ for executive). For total and acoustic runtime, the good effect of context checking was reduced when words were mapped all at once, and the increase in runtime caused by map-all was greater when also context checking. For executive runtime, both context and map-all had good effects, and there was actually a synergistic relation; context checking helped more when mapping all at once, and vice versa.

There was also a significant three way interaction among focus, map-all, and island-driving ($p < .01$ for total and acoustic runtimes; $p < .05$ for executive). When not mapping all at once, there was negligible interaction between focus and island-driving. However, when mapping all at once, the combined bad effect of focus and island-driving was significantly less than the sum of their individual bad effects.

The runtime results follow basically the same pattern as the accuracy and storage results. Focus and island-driving have bad effects, with worse results from island-driving for longer sentences, while context checking has consistently good effects. Map-all has a good effect on executive runtime, but, unfortunately, it causes large increases in acoustic and total runtimes. The only inconsistency with the previous pattern of effects for accuracy and storage is the bad effect of map all on the acoustic runtime. This fact is explained by pointing out that the mapper was designed for mapping words one at a time and, in the simulation, does not accumulate or share information to make subsequent tests more efficient. Finally, it is noteworthy that the extra effort for context checking resulted in a net decrease in processing time. For example, fCMi required an average of 6.3 seconds more per sentence to do context checks, but was still 41 seconds per sentence faster than fcMi.

The runtime figures above are in units of seconds used to process a sentence. A more common unit for runtime is seconds per second of speech. This is a reasonable scale if the runtime is essentially a linear function of sentence length and has a zero intercept. Both assumptions are consistent

with our data. No significant nonlinearity was found by an F test of the variance of the mean for each length about the regression line, relative to the combined variance of the sentences within a given length (for instance, the data for fCMi gave F=1.37 versus F(4,54)=1.41 for $p < .25$). Moreover, the 95% confidence interval for the intercept of the regression line included the origin. With this justification, we used zero-intercept linear regression to calculate the processing times per second of speech and their 95% confidence intervals.

The results for the fastest system, fCmi, were 141, plus or minus 14 seconds processing per second of speech for total runtime; 66, plus or minus 9 for executive runtime; and 63, plus or minus 7 for acoustic runtime. The results for the most accurate system, fCMi, were 247, plus or minus 21 for total runtime; 34, plus or minus 6 for executive runtime; and 205, plus or minus 14 for acoustic runtime. Thus, for fCMi, 83% of the total runtime slope comes from acoustic processing, 14% from the executive, and the remaining 3% from the semantics. Clearly, the best approach to improving fCMi runtime is to redesign the mapper for mapping all at once. The potential is large for sharing work to improve efficiency in the mapper, since the data show that fCMi is mapping all the words at an average of 15 out of the 20 possible positions per second of speech.

The results of Experiment 4 have relatively clear implications for the control-strategy design choices. The effects of focus by inhibition were all bad, too often focusing the system on false alarms instead of hits. An unbiased best-first strategy is better. The effects of island-driving were also bad, and they were particularly bad for longer sentences. Island-driving was hurt by false alarm islands, especially when the sentence was long enough for the islands to grow in both directions. Perhaps island-driving can be modified to overcome this problem, but until then, simple left-to-right processing is better. Context checking had uniformly good effects. For both accuracy and runtime, it was worth the extra effort in order to get better priority setting. The only ambiguous control choice is whether or not to map all at once. Mapping all at once improves accuracy and executive runtime, but at a large cost in acoustic and total runtime. Redesign of the mapper could undoubtedly resolve this choice in favor of mapping all at once. For example, just cutting the acoustic processing in half would make the fCMi system about as fast as the fCmi system. The choice whether to map all or not, is

explored further in Experiment 6. Finally, note that changes in the mapper appear to offer the best chances for significant improvements in both accuracy and runtime.

EXPERIMENT 5--GAPS AND OVERLAPS

The data from Experiment 1 do not aid us in simulating the mapper's performance when called on to test whether two words it has accepted individually are also acceptable as a contiguous pair. Such tests are necessary whenever words and phrases are combined to form larger units. In the basic simulation of the mapper, we simply allowed gaps and overlaps between words of up to 0.05 seconds of speech. Experiment 5 tests the effect of different values of the gap-overlap parameter on the performance of the fCMi system from Experiment 4. Table 9 gives the results for a variety of measures with gap-overlap sizes of 0.00, 0.05, and 0.10 seconds.

Table 9
EFFECTS OF GAP-OVERLAP PARAMETER

	GAP-OVERLAP SIZE		
	0.00	0.05	0.10
Fanout with acoustics (words)	6.6	18.0	29.4
Rank of hit in fanout	1.9	3.7	5.6
Raw accuracy %	96.7	73.3	48.3
Forgiving accuracy %	98.3	80.0	58.3
False terminal %	58.2	83.2	89.1
Number of nonterminals	31	113	217
Total runtime (sec/sec-speech)	140	247	333
Executive runtime "	10	34	69
Acoustics runtime "	128	205	243

The performance is much better for 0.00 and much worse for 0.10 seconds of gap-overlap. This is strong evidence for the importance of special acoustic tests to verify word-pair junctions. Such tests can lead to a large reduction in the average hit rank and, consequently, to significant improvements in both accuracy and runtime.

EXPERIMENT 6--INCREASED VOCABULARY AND IMPROVED ACOUSTICS

Experiment 6 studies the effect on system performance of increased vocabulary size and improved acoustic-processing accuracy. As test systems, we use fCMi and fCmi from Experiment 4. These were the best systems for accuracy and speed, respectively, and would also give us more information

about the map-all control strategy choice. Thus there were three experimental variables: vocabulary size, acoustics, and map-all. Data for two of the eight combinations, map-all or not for smaller vocabulary and regular acoustics, came from Experiment 4. For Experiment 6, the other six combinations were tested to provide a complete set of data for analysis of the effects of the variables.

The large vocabulary is a 451-word superset of the 305-word vocabulary used in the other experiments. The data gathered in Experiment 1 showed that with the 451-word vocabulary the mapper made 2026 false alarms and had a false alarm rate of 142 false alarms per second of speech (compared to 114 for the 305-word vocabulary). Using this information, the mapper performance was simulated for the large vocabulary on the same set of 60 test sentences.

Improved acoustic-processing accuracy was simulated by a 7% downward stretch of the false alarm score distribution, while leaving the hit scores unchanged. In other words, a false alarm score X, in the range 45 to 100, was replaced by 1.07X-7. If the result was below the threshold of 45, the false alarm was eliminated. This process reduced the number of false alarms for the 305-word vocabulary from 1564 to 1204, and for the 451-word vocabulary, from 2026 to 1541. Because the subthreshold scores were eliminated, the simulated improvement left the average false alarm score almost unchanged: for the 305-word vocabulary, it went from 59.4 to 60.2, and for the 451-word vocabulary, it went from 58.2 to 58.8. We feel that an improvement in acoustic accuracy of the magnitude simulated here could have been achieved by careful tuning of the mapper.

Table 10 records the accuracy results using the notation "M" for tests with mapping all at once, "m" for those without, "A" for systems with improved acoustics, "a" for those without, "V" for systems with increased vocabulary, and "v" for those without. Improved acoustics raises fCMi accuracy from 73.3% to 85.0%, or from 80.0% to 95.0% if harmless errors are forgiven. However, if vocabulary size is also increased, accuracy drops slightly from 73.3% to 71.6%. Thus, in this experiment, a 7% improvement in acoustics almost compensates for a 48% increase in vocabulary. Comparison of the M-results to the m-results shows that map-all consistently helps accuracy.

Table 10
ACCURACY RESULTS
(percent)

	AMv	Amv	aMv	AMV	amv	AmV	aMV	amV
Raw	85.0	78.3	73.3	71.6	70.0	68.3	68.3	53.3
Forgiving	95.0	85.0	80.0	78.3	76.7	76.7	75.0	58.3

The main effects on accuracy and several other measures are given in Table 11. Improved acoustics leads to big gains in accuracy, storage, and runtime. Increased vocabulary makes performance worse, but at least the system does not collapse. As in Experiment 4, mapping all at once improves everything except acoustic and total runtimes.

Table 11
MAIN EFFECTS OF ACOUSTICS, VOCABULARY, AND MAP-ALL

	WITH	WITHOUT	DIFFERENCE	
Raw Accuracy (percent)				
A	75.8	66.3	9.5	**
V	65.4	76.7	-11.3	#
M	74.6	67.5	7.1	*
Phrases (total number terminal and nonterminal)				
A	155	208	-53	**
V	204	159	45	**
M	156	206	-51	**
False Terminals (percent)				
A	80.6	85.9	-5.3	**
V	84.3	82.1	2.2	
M	81.7	84.8	-3.1	**
Total Runtime (seconds/sentence)				
A	266	320	-54	**
V	312	275	37	*
M	383	204	179	**
Acoustic Runtime (seconds/sentence)				
A	187	213	-26	**
V	205	195	10	
M	315	84	231	**
Executive Runtime (seconds/sentence)				
A	66	89	-23	**
V	88	67	21	**
M	55	101	-46	**

** $p < .01$ * $p < .05$ # $p < .10$

There were few significant interactions. Vocabulary size and mapping all at once interacted significantly for acoustic runtime ($p < .05$) and for total runtime ($p < .10$). Table 12 shows that the increase caused by map-all is greater for the bigger vocabulary, and, surprisingly, that the increase in vocabulary size leads to a reduction in processing, if the system is not mapping all at once.

Table 12
VOCABULARY AND MAP-ALL INTERACTION
for ACOUSTICS RUNTIME
(seconds/sentence)

	M	m	M-m
V	335	75	260
v	296	94	202
V-v	39	-19	58

Table 13
ACOUSTICS AND MAP-ALL INTERACTION
for FALSE TERMINALS
(percent)

	M	m	M-m
A	78.6	82.6	-4.0
a	84.8	87.0	-2.2
A-a	-6.2	-4.4	-1.8

Mapping all at once also interacted significantly with acoustics for acoustic runtime ($p < .01$), total runtime ($p < .01$), and false terminal percentage ($p < .05$). All cases were similar to the one shown in Table 13. There was a synergistic interaction causing mapping all at once to be more effective with better acoustics, and vice versa. This result is readily explained since map all is designed to take advantage of the difference between false alarm and hit score distributions, and improving the acoustics enhances that difference by reducing the number of high scoring false alarms.

In addition to the main tests for Experiment 6, we also ran two other tests to study the effect of improved acoustics on systems using island-driving and focus by inhibition. The best island-driving system from Experiment 4 was fCMI. When tested on the 305-word vocabulary with 7% simulated improvement in acoustics, fCMI gained in accuracy from 68.3% to 78.3%. It was still below the non-island-driving fCMi, and the gap between them remained large. (Recall that fCMi went from 73.3% to 85.0%.) The best focus by inhibition system was FCMi. Improved acoustics raised its accuracy and reduced its runtime, but it was still less accurate than fCMi (80.0% versus 85.0%) and also slower (235 seconds per sentence, versus 200). The accuracy difference vanished with a 24% simulated improvement in acoustics, but, even with a 51% simulated improvement, a slight runtime advantage for fCMi remained.

In summary, this experiment has given us information about how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustics. With respect to the control-strategy design choices, further evidence appeared in favor of mapping all at once, and against island-driving and focus by inhibition.

CONCLUSION

Reviewing the series of experiments, the first experiment showed that the acoustic processing component called the mapper" had a high false alarm rate, but tended to give better scores to hits than to false alarms. In the second experiment, we measured the number of alternatives open to the system for extending segments of sentences. The size of the fanout helps to explain the difficulty of speech understanding. The third experiment found that two simple system designs were too simple, a result that helps to justify the complexity of the standard system. The fourth experiment studied the effects on system performance of four control-strategy design choices. Focus by inhibition and island-driving had bad effects, while context checks for priority setting had good effects. Mapping all at once had good effects on everything except acoustic and total runtime, and these bad effects could probably be eliminated by redesign of the mapper. In fact, mapper changes appear to offer the best hope for large gains in both accuracy and runtime. The fifth experiment varied the size of allowed gaps and overlaps between words and showed the potential value of special acoustics tests to verify word-pair junctions. Finally, the sixth experiment gave quantitative measures of how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustic accuracy. The experiment also provided more information about the control choices. Overall, the series of experiments gives insights into system performance and control strategy that should be useful in designing future speech understanding systems.

REFERENCES

Paxton, William H. A Framework for Language Understanding. In COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Canada, 28 June - 2 July 1976. [Technical Note 131, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1976.]

Paxton, William H. A Framework for Speech Understanding. Stanford University Ph.D. dissertation, Stanford, California. forthcoming.

Ritea, H. Barry. Automatic Speech Understanding Systems. Proceedings, Eleventh Annual IEEE Computer Society Conference, Washington, D.C., 9-11 September 1975.

Winer, B. J. Statistical Principles in Experimental Design, second ed. McGraw-Hill Book Company. New York, New York. 1971.

A Word Based Bi-Directional Speech Parser

Corot C. Reason
University of Toronto

ABSTRACT

This paper is a brief description of the design of a syntax based speech parsing system. The motivation came from work by Riesbeck and grew into a scheme that resembles an Augmented Transition Network in many ways. Included is a discussion of some of the problems encountered in an implementation of the design.

The design of the speech parser described here was motivated by the natural language analyser built by Riesbeck [RIESBECK 74]. His notion that a single word or phrase can be used to develop expectations about other words and phrases that are likely to appear in the text seems to be extensible to the area of speech parsing at a syntactic level. Since identification of words in a speech signal is very difficult, any expectations about words that may appear may be very useful.

The parsing scheme that I will describe here leads to programs that are vocabulary dependent. By that I mean any "fine tuning" of a program using this scheme will be related more to the words used, rather than to the context the words are used in. As a result it is best to describe the scheme as syntactically

based. The scheme does not preclude the use of a body of contextual and semantic knowledge but a speech system with such knowledge would use this parsing scheme as one module, the products of which would then be evaluated and used by other modules, perhaps on an interactive basis.

This type of modular structure was used in the BBN speech system [WOODS 74]. In that system the syntactic parsing was handled by an ATN. The parsing scheme I will describe here is logically very similar to an ATN but has some organizational advantages.

The parse activity is primarily verb based. Verbs carry with them, in the lexicon, case frames (called Assemblies) that specify the different environments the verbs may appear in. The collection of all case frames for verbs in the lexicon determines the range of sentence structure that the parser recognizes. The syntactic structure of phrases mentioned in the assemblies are not specified by the assemblies. For example a particular assembly may specify the expectation of a noun phrase or to-complement. The recognition of these syntactic units is handled by 'finders', small programs that contain (procedurally) the definitions of the units they locate. Thus the grammar of the language recognized can be changed to a certain extent without altering the assemblies. This parallels the ability to change the definitions of phrases in an ATN. The ATN, when referencing a language construct, uses a PUSH arc and this allows for recursive language constructs. Finder programs may also operate

recursively, although recursive language constructs do not necessarily have to be recognized in this way.

The assemblies are linear and hence may be read either right-to-left or left-to-right, a feature which is necessary in speech parsing. This feature makes finder programs more complicated since they may be called upon to locate a phrase of some sort either "on the left of" or "on the right of" some cursor position. This also necessitates a system for storing complete or incomplete syntactic units in a way that operates with no left-right preference. Each identified structure must be stored by its known boundary locations, which may change as processing continues. As the parse proceeds these phrases may become part of larger phrases, but their original location must not be forgotten so that in case of error the inner phrase can be recovered. In fact using assemblies allows for two or more parses to be developed for the same utterance, or part of it. So the sharing of small phrases among competing parses is facilitated.

Assemblies are associated with particular verb forms in the lexicon. For example the form "gave" would have a list of assemblies as would the form "given" or "give". The lists of assemblies may have common parts and these parts can be shared in order to achieve storage economies. As the vocabulary size grows the number of different new assemblies will fall off so that with a very large vocabulary a nucleus of assemblies will be used by all the verb forms in the vocabulary.

An assembly is, effectively, a position ordered sequence of function calls. Assemblies can be represented by programs or some other data structure. In the implementation I have carried out assemblies are a combination of character string and program, a combination which is dependent on the language the implementation was written in. I write assemblies on one line, with a * marking the verb position. There is always a verb in each assembly, but this does not exclude the possibility of parsing verb-free sentences such as idioms or expressives. As an example:

NG(Subj)-THAT-*-NP(Indobj)-NP(Dobj) = NP

is an assembly that can be associated with 'bought' (also: 'gives', 'sold', etc.) and encodes the fact that: A noun group (determiner, adjectives, noun) followed by 'that' followed by the verb and two noun phrases form a noun phrase. The noun group is the subject of the verb. The other noun phrases are the indirect object and direct object of the verb.

Additional information can be added to calls to the finder programs. For example a noun phrase can be searched for with: NP(subject, human, singular). The additional phrase markers can be used to shorten lexicon searches and also they provide additional information on which to base evaluation of the located phrase.

Naturally in processing a speech signal, location of a verb with any degree of confidence may be a rare event. It is important that the discovery of any word in the signal can

initiate parsing. This can be achieved by identifying the word type (adjective, preposition etc.) and calling the finder program that is likely to be able to locate a phrase in the signal that incorporates that word. For example, discovery of an adjective would lead to the invocation of the noun phrase finder program. This program would try to locate a noun phrase surrounding the adjective. The finder would terminate with possibly incomplete results, which are stored for attempted completion or use at some later point.

Isolated words can be located in a speech signal using information about stressed regions in the signal. Stress can be identified from non-phonetic data in the signal in an economic manner (see [LEA 73]). The reliability of phonetic data in stressed regions tends to be higher than in other regions and so words that match well in stressed areas are a good starting point for parse activity. Whether it is a verb or other type of word that is located, the parse can proceed.

In most cases the parser must make a decision about what the root and form of the main verb is. If the main verb is not located through the use of stress information or some other means the identified phrases in the signal can be used to suggest verbs that are likely. The pattern of phrase types and their relative location along with information about gaps can be used as a template which is laid down on top of each assembly in the lexicon. Assemblies that match the template suggest their parent verb as a candidate for the main verb in the utterance. With

small vocabularies this may reduce to a choice of one verb, with large vocabularies the size of the space to be searched is reduced.

Non-phonetic information, apart from stress data, can be used in assemblies as well as syntactic information. For example Lea [LEA 74] has shown that pitch-pause anomalies are frequently inserted at the end of major phrases by speakers in English. These anomalies can be detected (with some chance of error) and added to the acoustic data that the parser can make use of. The anomalies can be located by a finder program which in turn can be referenced in assemblies. In this manner the anomalies can be used to (perhaps) disambiguate the end of complex noun phrases (i.e. "the boy that spoke to the girl..hit a home run.").

The structure of the program that controls the asssemblies in the implementation I have undertaken is somewhat ad hoc and certainly lacks any theoretical basis. I think this may be a problem common to all speech systems. The combination of the ambiguity in the speech signal and the number of rules available to try and clear away the ambiguities makes it difficult to assess the value of one control mechanism over another.

The approach I use involves a number of basic tools whose use is controlled by a small production system. One tool is word matching in stressed regions, as I have mentioned. Another is building phrases from words that have been located in the signal by some program but remain isolated. The third tool involves manipulating a threshold. At the start of the interpretation of

the signal the parser only uses words that have been located in the signal with a particular confidence or better. This threshold of acceptance can be lowered when progress has stopped. The problem, of course, is that as the threshold is lowered the chance of accepting an incorrect word match increases. At the same time some words are likely to be garbled in the signal, and the location of these will be hindered by an acceptance threshold that is too high. So there is a difficult tradeoff.

Additionally the parser can resort to brute force techniques for the location of words. For example, in a question-answering environment the left hand side of the signal can be checked for common openings like "What is", "Give me" etc. Additional information about the domain and context of the situation can be used to do across-the-signal searches for common words, but these searches will be expensive.

The use of these tools must be controlled so that expensive things are done only when needed and cheaper things used conservatively so as to allow filtering of the information in the signal. The production system orders the tools, roughly putting cheapest at the top, most expensive at the bottom. The tools are selected in that order only if certain conditions exist in the parse. The conditions I used, and the ordering of the productions were never satisfactory and were the subject of constant adjustment.

In addition to the difficulties of tool selection I found the evaluation of the 'goodness' of complete or incomplete phrases to

be very difficult. It seems natural to assign numeric values to word matches and phrases so as to indicate their 'distance' (in the information theory sense) from the signal. However, how one favors long words and phrases and how words are combined into phrases and given combined evaluations is not easy to decide. Certainly in my implementation the rules for evaluation were ad hoc. Additionally the signal is error filled and the notion of judging goodness of word matches as a distance from some almost random signal is questionable. There does not seem to be any way to develop a system of evaluation rules that works well and has any sort of consistency. Perhaps no such set of rules exists and the use of numeric evaluation is the root problem.

Implementing this parsing strategy has been a worthwhile experience. I have learned that the overall mechanics are feasible in terms of execution time and performance. At the same time I have come to the realization that the implementation details are difficult to grasp mentally and hence difficult to change and improve upon. By far the greatest need now is to organize and develop a theory for the rules that govern decision making and the evaluation of partial results.

References

[LEA 73] Lea, W., Medress, M., Skinner, T.; Prosodic Aids to Speech Recognition II and III; Technical reports PX10232 and PX10430, UNIVAC Corp.; St. Paul, Minn., April 1973 and September 1973.

[LEA 74] Lea, W.; Prosodic Aids to Speech Recognition IV; Technical Report PX10791, UNIVAC Corp., St. Paul, Minn., March 1974.

[RIESBECK 74] Riesbeck, C.; Computational Understanding: Analysis of Sentences and Context; Stanford Artificial Intelligence Laboritory Memo AIM-238, May 1974.

[WOODS 74] Woods, W., et al.; Natural Communication with Computers, vol 1; BBN report 2976, December 1974.

References not cited in text

[MILLER 73] Miller, P.; A Locally Organized Parser for Spoken Input; Technical report 503, MIT Lincoln Lab, Lexington Mass., May 1973.

[NEWELL 72] Newell, A., Simon, H.; Human Problem Solving; Prentice-Hall, Englewood Cliffs, N.J., 1972.

[REASON 76] Reason, C.;A Bi-Directional Speech Parsing Technique; University of Toronto Department of Computer Science Technical Report 90, March 1976.

[WALKER 75] Walker, W.; Speech Understanding Research; Technical report, Stanford Research Institute, Menlo Park, Cal., June 1975.

PRE-PROCESSING FOR IMAGE PROCESSING

by

Surender Kumar Kenue and Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta

Abstract

Some pre-processing methods for image processing are presented. The first method is designed for normalizing pictures. Normalization is achieved by transforming the original picture to a new picture, with any desired histogram distribution. Various distribution curves experimented include gaussian, triangular, flat, step etcetera. Gaussian curve appears to be the best choice for normalization. Also, a non-iterative picture enhancement method, based on a binary neighborhood relation is proposed. The proposed method obtains better enhancement than Kramer and Bruncker's iterative method. Applications of the method include picture sharpening for visual inspection, performance improvement of an edge detector, data compression, and the obtaining of uniform regions for segmentation.

Section 1: Introduction

This paper presents some pre-processing methods for computer perception of pictures. The first method deals with gray scale normalization. Normalization is essential for pictures taken at different times and under different conditions, since the grey levels will be quite different, even if the overlapping sub-scenes are essentially the same. For example to detect differences between two pictures, it is important either to normalize both pictures to some standard picture or to normalize one picture relative to the other. The method developed is discussed in the next section.

The second part of this paper is concerned with picture enhancement. Picture enhancement is defined as obtaining a good

contrast picture at the expense of losing some detail. Many of the features may not be clearly visible, because of slow gray level transition between objects. The purpose of enhancement is to convert this slow transition to a fast transition so as to achieve good contrast in a picture. The proposed method is discussed in section 3.

Finally, in the last section, test results obtained by applying these methods are given and some conclusions are drawn.

Section 2: Gray Scale Normalization

Normalization of two data vectors when obtained by different techniques is often required in statistics, pattern recognition & in the social science applications. A survey of current techniques in histogram modification is reported by Hummel [3]. Given any desired histogram distribution, the new method generates a picture with the desired histogram distribution, preserving most of the important information in the picture. The method is non-iterative, efficient and expands the original gray level range to the full range (0,2**b), for a b-bit pixel. Various distribution curves like gaussian, flat, step, etcetera have been tried on a limited set of pictures.

Unlike other researchers, who either decrease the number of gray levels or leave it same, this method expands the original range to the full scale (0,2**b). Thus, flattening differs from that of Eberlein et al's [1] in the sense that all 0 to 63 gray levels get the same number of pixels. The method can be summarized as follows:

The histograms of the original picture and the desired distribution picture are computed. A decomposition matrix is defined from these two histograms. The rows correspond to the original gray levels and the columns correspond to the doublet (i,j), where i represents the number of original gray levels which should be assigned to the jth desired gray level. e.g. :

Gray level	# of pixels	(# of pixels, assign to desired g.l)
20	46	(14,1), (27,2), (5,3)

The above line reads : Decompose 46 pixels of gray level 20 into 14 pixels of gray level 1, 27 pixels of gray level 2 and 5 pixels of gray level 3.

The original picture is scanned row wise and at every pixel under consideration, an average of neighbor pixels is obtained. If that average is greater than the pixels gray level, the highest possible gray level from the doublet (i,j) is assigned. Similarly, if the average is less, the lowest possible gray level is assigned. After scanning all the rows, the transformed picture's histogram will be exactly as prescribed.

Section 3: Picture Enhancement

Picture enhancement is concerned with making interesting minute details more visible in a picture. Ideally, one would like to sharpen pictures without losing information; however, most enhancement methods do loose some information. A number of enhancement methods have been discussed in [5]. In addition, a

non-linear transformation for enhancing digitized pictures has been proposed by Kramer and Bruckner [4]. The process is iterative, and according to the authors, it takes about 20 to 50 iterations for a 27*33 matrix. For practical applications like Landsat pictures, where one scene consists of 2286*3600*4 pixels, iterating many times is extremely time consuming. To enhance, a single transformation, which could be applicable in a single pass, would be more useful. The purpose of this section is to propose a non-iterative procedure for picture enhancement.

The Proposed Method

Neighborhood information has been used very often in picture processing [2]. The notion of co-occurances of gray levels and various other forms of statistics have been exploited for texture analysis. The neighborhood matrix of [2] differs from the matrix introduced in this section in the sense that no statistics are computed. Only binary relations are defined between gray levels. The binary matrix is defined formally as given below:

Definition :

Let NEIG(i,j) be a binary neighborhood matrix and N(x) represent the local neighborhood of the point x.

(1) If $y \in N(x)$ then NEIG(y,N(x)) = 1
else NEIG(y,N(x)) = 0

(2) If $y \in N(x)$ then NEIG(N(x),y) = 1
else NEIG(N(x),y) = 0

(3) NEIG(y,y) = 1, where y belongs to all gray levels in the picture set.

The binary neighborhood matrix is symmetric, thus only half of the binary matrix has to be computed. To illustrate the definition, consider the following matrix (Fig. A) of gray levels

18	21	24	24	21		25	25	25	25	25
15	12	25	21	21		12	12	25	25	25
16	16	25	25	25		12	12	25	25	25
16	17	18	25	21		12	16	25	25	25
15	17	17	17	21		12	16	16	16	25

Fig. A Original Fig. B Enhanced

Let the pixel under consideration be (2,2), with the neighborhood function N2, where N2 = ((x,y),(x-1,y),(x,y-1),(x+1,y),(x,y+1)). The following relations are defined for gray level 12 in the binary matrix:

NEIG(12, (21,15,16,25)) = 1

NEIG((21,15,16,25), 12) = 1

and NEIG(12,12) = 1

Similarly after scanning rows 2 to 4, the following binary matrix is obtained.

gray level	12	15	16	17	18	21	24	25	transformed vector
12	1	1	1	0	0	1	0	1	12
15	1	1	0	0	0	0	0	1	12
16	1	0	1	1	0	0	0	1	12
17	0	0	1	1	1	0	0	1	16
18	0	0	0	1	1	0	0	1	17 / 25
21	1	0	0	0	0	1	1	1	25
24	0	0	0	0	0	1	1	1	25
25	1	0	1	1	1	1	1	1	25

The enhancement method is described as follows :

Step 1: Compute the binary neighborhood matrix. Next, each row is scanned for the leftmost and rightmost '1'. The object is to find the minimum & maximum gray levels, which are neighbors to the current gray level(or row). The gray level which is closest to the row number is assigned to the transformed vector.

Step 2: The range of the original gray levels is divided into two parts (low range and high range) at approximately mid range. If any transformed gray level within the low range is greater than the mid range value, then that transformed gray level is replaced by the left most neighbor of the current row. Similarly, for the high range, if any transformed gray level is less than the mid range, the transformed gray level is replaced by the rightmost neighbor The overriding factors will suppress mapping, which take a gray value from a low/high range to high/low range.

Step 3: Since the transformation is many to one, a condition where the gray levels with large frequencies may be mapped into one single gray level may occur. In that case, a check is made to determine the number of pixels which the transformed picture contains. If this number is more than K % of total pixels, these gray levels are not transformed. In this paper, K was set to 40.

Step 4: The enhanced picture is obtained by applying the transformation vector to the original picture.

The enhanced matrix of figure A is given by figure B.

The new method is compared with Kramer and Bruckner's

method and Weszka et al's method [6]. Kramer and Bruckner's method can be summarized as follows :

Given a digital picture P, a new picture is obtained by replacing the gray levels of each pixel by either a local maximum or minimum, whichever is closer to the current pixels gray value. Next the above process is repeated till successive pictures are approximately the same, implying convergence.

Weszka et al simply add a Laplacian to the picture for enhancement. The operator is given by $2f(i,j) - A(i-1,j-1)$, where $f(i,j)$ is the current pixel and $A(i-1,j-1)$ is the standard 3*3 Laplacian.

Section 4: Conclusions and Test results

Two photomicrographs of a Spinach leaf (Figure 1) and a Neuron (Figure 10) were digitized. The pictures are of 256*256 (6 bit pixel) size, and 63 is the most white gray level.

(a) Gray scale normalization

The flat line and gaussian curve pictures are given in figures 5 & 6 for Spinach and figures 14 & 15 for Neuron respectively. Also, some linear transformations on the normalized pictures were applied. For example, one could obtain a step histogram or a U shaped histogram from a flat one. The main conclusions are :

Flattening enhances the pictures, while some detail is lost. Also edge detector output (Figures 9 & 18) was not good, as has been reported in [1]. Figures 9 & 18 were obtained by using a Roberts operator with a threshold of 7. Higher threshold

values did give better edge output, but the enhanced picture's edge output (Figures 9 & 17) was the best.

Gaussian curve picture appears to be the best choice for normalization. Triangular curve picture seems to behave similar to the gaussian curve picture.

(b) Picture Enhancement

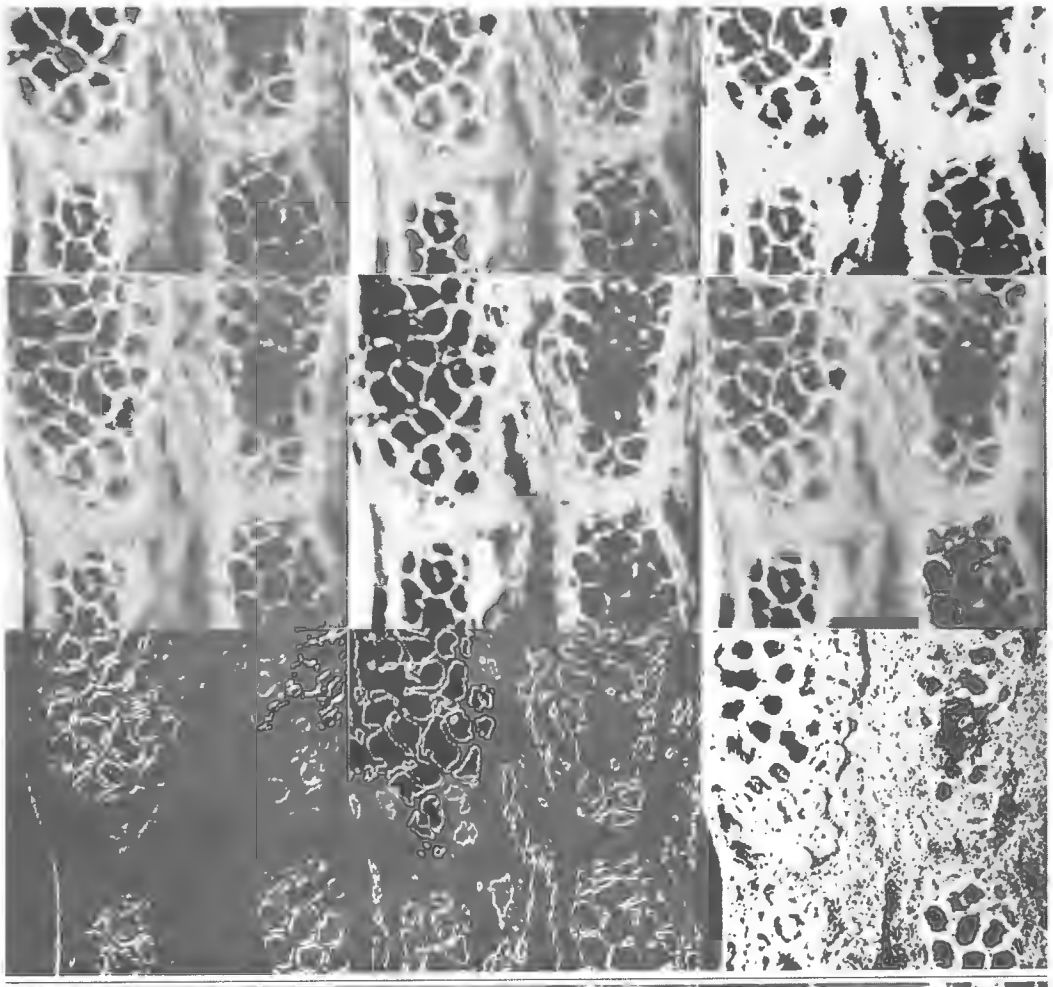
Kramer and Bruckner's iterative method was programmed for a maximum of six iterations. It was found out that the pictures got worse instead of enhanceing them (Figures 4 & 13). Also, the shapes of the objects in both pictures was smeared. Weszka et al's method also did not enhance well (Figures 2 & 11). The enhanced pictures obtained by the new method are given in figures 3 & 12. Thus for all the three methods, limited experiments suggest that the new method is best for visual inspection.

The second application of the method appears to be in the improved detection of edges and contours, using standard edge operators. The edges detected by using Roberts operator with a threshold of 7 on original pictures are shown in figures 7 & 16 for Spinach and Neuron respectively. In the enhanced pictures, edges and contours detected (Figures 8 & 17) were very sharp, noise free and much better than those detected on the original pictures. Various other operators such as Gradient and the Laplacian were tried with the same lack of success. Also, uniform regions were obtained in the enhanced pictures, which is a pre-processing step for segmentation.

It was also noted that a modified enhancement method can be used as a data compressor for converting gray level pictures to black/white pictures.

References

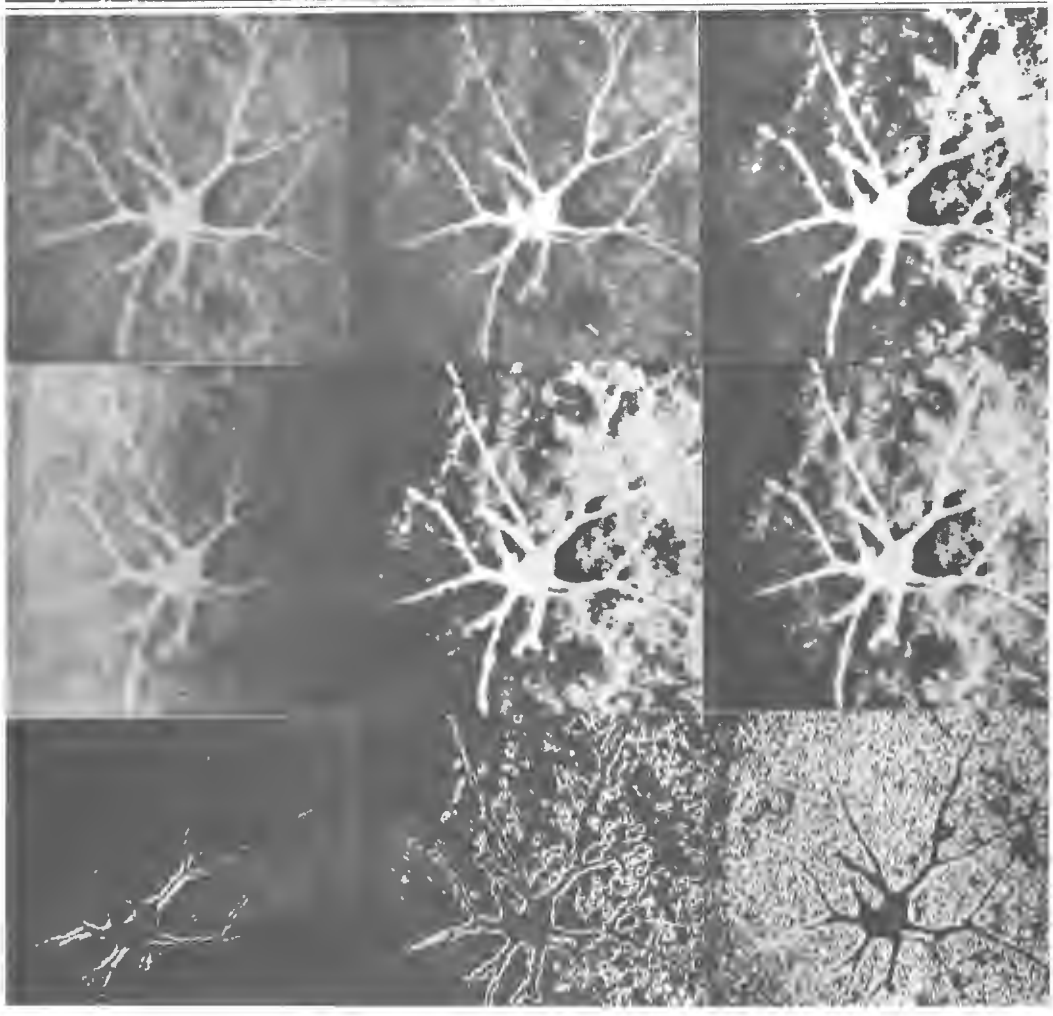
1. Eberlein R., VanderPrug, G.J., Rosenfeld, A., Davis, L.S. Edge and line detection in the ERTS imagery: a comparative study. Rep. TR-312, Univ. of Maryland, Md., June 1974.
2. Haralick, R.M., Shanmugam, K., and Dinstein, I. Texural features for image classification. IEEE Trans. SMC, SMC-3(Nov. 1973), 610-621.
3. Hummel, R.M. Histogram modification techniques, Computer Graphics & Image processing. Vol. 4(3)(Sept. 1975), 209-224.
4. Kramer, H.P., and Bruckner, J.B. Iterations of a non-linear transformation for enhancement of digital images Pattern Recognition, Vol. 7(1975), 53-58.
5. Rosenfeld, A. Picture Processing by Computer. Academic Press, New York, 1969.
6. Weszka, J.S., Carton, E.J., Verson, J.A., Mohr, J.M., and Rosenfeld, A. Some basic edge degradation and enhancement techniques. Rep. TR-278, Univ. of Maryland, Md., Dec. 1973.



SPINACH

FIGURES

1 - 9



NEURON

FIGURES

10 - 18

MAKING MAPS MAKE SENSE

Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada

Abstract

Attacking the interpretation of images designed for person-to-person communication, which have a conventional semantics, will throw more light on the problem of computational vision. A class of such images, sketch maps drawn on a graphical data tablet, has been chosen for study. The design of an initial system is presented and justified with reference to five explicit goals of the project. As with most vision tasks the fundamental problem is resolving ambiguity in context. The system finds instances from a repertoire of primary cues, using them to access a set of cartographic models. The resulting ambiguities are resolved using one of the network consistency algorithms. The uses of multiple representations, levels of detail, model descriptions, intelligent segmentation, cue/model hierarchies and procedural models are also discussed.

1 Introduction

Those who are waiting impatiently for "useful" systems to emerge from computational vision often complain about the narrowness of the domain of pictures that can currently be sensibly interpreted. Although that narrowness was well-justified as an initial research strategy (Mackworth, 1975a), it is time to broaden our scope without losing sight of what has been so arduously won.

Man-made images designed for person-to-person communication have largely been ignored in favour of images of natural scenes. Given that man-made images have their semantics fixed by the laws of convention rather than the laws of optics that dictate the semantics of natural images, this may have been a mistake. Convention is often richer, cleaner and more easily interpreted than nature. This is not to advocate abandoning work on exploiting the nature of textures, shadows, highlights and edges in the interpretation process; by analogy with speech

understanding, it is clear that progress will require a judicious admixture of both approaches.

A common class of image designed for communication is the free-hand sketch diagram. For several years we have had the ability to draw such diagrams directly on graphical data tablets but this ability has not been heavily exploited. Most uses have been very mechanical and ad hoc; only rarely (Anderson, 1968; Negroponte, 1973) can a program be truly said to be interpreting the sketch.

In studying images sketched free-hand on a data tablet, this project has many goals. They include:

A) To explore the relationship between natural and conventional representations.

B) To see if we can broaden the scope of our vision programs by applying the lessons learned in the blocks-world decade to other domains.

C) To determine the extent to which highly domain-specific knowledge can be factored out of the image interpretation program, to be supplied by the user.

D) To make available a useful interpretation program for some restricted but important classes of sketches.

E) To provide an experimental vehicle for studying the control structures required to implement schema-based theories of perception.

2 Sketch Maps

2.1 What?

The initial domain chosen was a set of sketch maps drawn on a data tablet typified by the map shown in Fig. 1. This is so badly sketched that many people have to be told, before they can

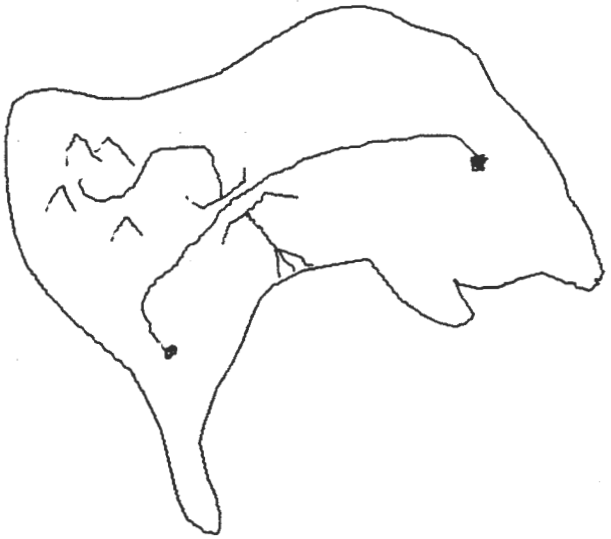


Figure 1. A sketch map of an island

see it, that it represents an island on which there is a road that connects two towns and crosses a bridge over a river that rises in a mountain range and ends in a delta.

2.2 Why?

This domain allows us to explore ways of satisfying all of the goals listed above. In particular, besides having a satisfying mixture of conventional and direct representations, such maps are related to the work we are doing on understanding LANDSAT (ERTS) images (Starr and Mackworth, 1976) which have optical semantics alone. In the long run, such understanding would proceed more successfully if programs were able to accept advice, in the form of sketch maps, about the geography underlying the image.

2.3 How?

The most important fact to realize is that picture elements

are ambiguous. A line element could, in isolation, represent a part of a road, a river, a bridge, a shoreline, a lakeside, or a mountain symbol. An areal element could be land, lake or sea. These ambiguities of interpretation are only resolvable by appeals to models and the interpretations of related picture elements.

A first implementation of an interpretation scheme can start from the fact that recognizing the existence of and controlling similar but simpler ambiguities in the blocks world led to much progress. The implementation cannot, however, proceed until considerable augmentation of those so-called labelling schemes has been achieved.

2.4 When?

An implementation of the system to be described in section 3 is now (July, 1976) about one-third complete.

2.5 Where?

The facilities used include an IBM 370/168 running under MTS with an Adage Graphics Terminal. The system is written in:

1) LISP/MTS (80% of the code so far)

2) MAYA an advanced AI language, corecursive with LISP, designed and implemented by W. S. Havens. Facilities for creating, touring and inferencing with procedural semantic nets (that is, frames) and a spaghetti stack control structure. (10% of the code so far, maybe 30% eventually)

3) GLISP a graphics extension to LISP (Hall et al, 1974) (5% of the code)

4) FORTRAN (5% of the code)

3 How? (Again)

3.1 Multiple Representations

One lesson learned in the blocks world is that pictures must have a variety of representations according to the needs of the various components of the interpretation task. Here we have three different representations of the picture. The first is a GLISP procedure for drawing the picture. Such a procedure is created initially by the stylus-tracking routines in GLISP corresponding to the pen-up and pen-down movements of the stylus on the tablet. Similar procedures can be resynthesized from the network description at any time for display purposes. Secondly there is a network representation of the picture in terms of chains (sets of connected line segments), line segments and segment end points. Each object, in the network built in MAYA, is a node with attributes, relations to other nodes, an indexed database of assertions and a set of local function definitions. Statically, a node may be an instance of a prototype node with corresponding information in it or, dynamically, a series of such nodes may be put on the stack to form an execution environment. Finally, there is an array representation of the picture in which each array element is a list of segment end-points. The array is indexed by the x and the y coordinates of the point.

3.1.1 Levels of Detail

Another lesson is that pictorial representations should encourage the use of a level of detail appropriate to the task at hand - only moving to finer resolution when necessary. Each of the three representations allows this. The code representation can be generated for display purposes at any level of resolution in the network. The levels of resolution in

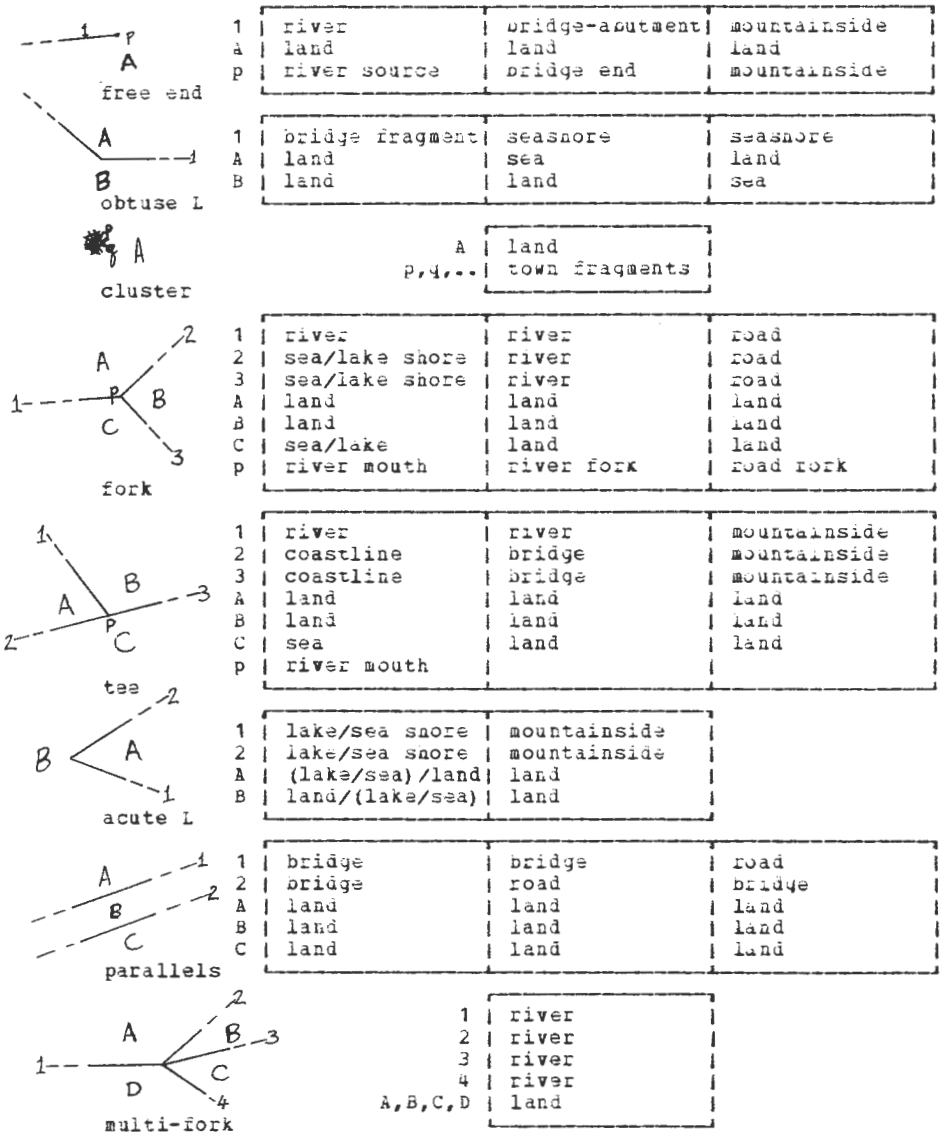
the network correspond to non-semantic, automatic map generalization. At each level the minimum number of line segments necessary to represent the map to a given level of accuracy is used. Finally, the array representation of the end-points is actually a set of arrays: a non-empty array element in array n is subdivided into 4 elements in array n+1.

3.2 Primary Cues

If you were to cut a small hole in a sheet of paper and move it about Fig. 1 before looking at the map as a whole but being familiar with the class of maps represented, you would discover a variety of interesting picture parts. The junctions, most noteworthily, contain much interesting but totally ambiguous information. A partial catalogue of picture parts, known as "primary cues", and their possible interpretations is given in Fig. 2. Note that each interpretation of a picture part places an interpretation on each of the line and region fragments that comprise the part. The primary cues are found by searching the most appropriate picture structure exploiting the levels of detail to make the search as efficient as possible. Each cue found is created as an instance of its archetype with the appropriate bindings for its parts and their relationships.

3.3 Network Consistency

Finding such cues in the picture and then searching for a mutually compatible interpretation would be analogous to the corresponding process in the blocks world; here interpretations are being placed not only on lines but also on regions. This general problem of satisfying a network of binary relations has been treated by a number of authors; Waltz (1972) and Montanari (1974) have proposed two interesting algorithms. Mackworth



	free end		
1	river	bridge-abutment	mountainside
A	land	land	land
p	river source	bridge end	mountainside

	obtuse L		
1	bridge fragment	seashore	seashore
A	land	sea	land
B	land	land	sea

	cluster
A	land
p,q,..	town fragments

	fork		
1	river	river	road
2	sea/lake shore	river	road
3	sea/lake shore	river	road
A	land	land	land
B	land	land	land
C	sea/lake	land	land
p	river mouth	river fork	road fork

	tee		
1	river	river	mountainside
2	coastline	bridge	mountainside
3	coastline	bridge	mountainside
A	land	land	land
B	land	land	land
C	sea	land	land
p	river mouth		

	acute L	
1	lake/sea shore	mountainside
2	lake/sea shore	mountainside
A	(lake/sea)/land	land
B	land/(lake/sea)	land

	parallels		
1	bridge	bridge	road
2	bridge	road	bridge
A	land	land	land
B	land	land	land
C	land	land	land

	multi-fork
1	river
2	river
3	river
4	river
A,B,C,D	land

Figure 3. A partial catalogue of primary cue interpretations

(1975b) presents the problem in general, those algorithms which ensure what is called arc and path consistency respectively, several extensions to them and a variety of applications. The simplest and most efficient arc consistency algorithm, known as AC-3, is described there. We do not have the space to re-present that algorithm or any further discussion of network consistency; suffice it to say that AC-3 will serve as a useful initial control model for this task. Note that in this application of network consistency the nodes (or "variables") are the primary cues found in the picture, their associated finite domains are the corresponding set of interpretations. There is an arc or binary relation between each pair of primary cues that share a line segment or region whose interpretation is constrained by both. The catalogue of interpretations of Fig. 2 is implicitly compiled from a set of models of the possible cartographic objects. But that compilation has lost some information which is present in the models so additional constraints that come directly from the models will influence the interpretation process.

4 Plans and Criteria for Current and Future Work

It seems clear at this stage that the mechanism proposed for the first implementation will work. However, that success will only totally satisfy one of the five long-range goals. AI must go beyond sufficiency.

4.1 Model Descriptions

One of the current short-term goals is to automate the process of generating the primary cue interpretation table. A solution to this problem will contribute to goal C: factoring-out some of the domain-specific knowledge. One could

then transfer to a new domain (for example, making sense of plan views of a house) with much less re-implementation effort. The approach in the second implementation will be to invent a language for describing the structure of the scene objects that can exist (here, the cartographic objects) in terms of a given repertoire of primary cues. The compilation process would then essentially invert these descriptions to construct the primary cue interpretation table. Note that then we would not throw away the model descriptions. The primary cues serve as indices into the set of models: their complete interpretations can then guide the consistency process. In the current scheme, most of the interpretation is guided by the labels placed on lines and regions: other model information is included on an ad hoc basis.

4.2 Consistency and Segmentation

In the blocks world and elsewhere, the slogan "Segmentation is interpretation." turned out to be valid. Here too, the process of making an interpretation consistent must interminqle with and, to a limited extent, drive the segmentation process of discovering the primary cues. It seems quite feasible to incrementally make the partial network consistent and allow the result to suggest which primary features to look for where.

4.3 Cue/Model Hierarchies

Using embedded hierarchies of models (Mackworth, 1976) is another way of interminqling top-down and bottom-up processing. Here we must establish levels of cue of varying complexity rather than the single level of the primary cue.

4.4 Procedural Models

Until now the procedural component of model specification has been suppressed. It has been withheld to explore the

implications of a uniform control strategy: arc consistency. However, experience (Stanton, 1972) has shown that it will eventually be necessary to allow the expresssion of strategic information in the models themselves. Although many of our goals can be satisfied in the network consistency framework (indeed modularity may only be satisfied there), we will have to abandon it eventually, or modify it beyond recognition, to satisfy the fifth goal: achieving an understanding of the control issues raised by schema-based theories of perception.

5 Acknowledgements

I am grateful to Bill Havens for MAYA. This research is supported by grant A9281 from the National Research Council of Canada.

6 Bibliography

Anderson, R. H. (1968) Syntax-directed recognition of hand-printed two-dimensional mathematics. Ph.D Thesis, Div. of Eng. and App. Phys., Harvard Univ., Cambridge, Mass.

Hall, W., Jervis, B. and Jervis, J. (1974) GLISP, a LISP based graphic language. Dept. Comp. Sci., Univ. of B.C., Vancouver.

Mackworth, A. K. (1975a) How to see a simple world. In Machine Representations of Knowledge, D. Michie (ed.) (in press) and TR 75-4, Dept. of Comp. Sci. Univ. of B. C., Vancouver.

Mackworth, A. K. (1975b) Consistency in networks of relations. TR 75-3, Dept. of Comp. Sci., Univ. of B.C., Vancouver.

Mackworth, A. K. (1976) Model-driven interpretation in intelligent vision systems. Perception 5 (in press) and TR 76-2, Dept. of Comp. Sci., Univ. of B.C., Vancouver.

Montanari, U. (1974) Networks of constraints: fundamental properties and applications to picture processing. Information Sciences, 7, 95-132

Negroponte, N. (1973) Recent advances in sketch recognition. AFIPS NCC Proc. 42, pp.663-675.

Stanton, R. B. (1972) The interpretation of graphics and graphic languages. Graphic Languages, F. Nake and A. Rosenfeld (eds.), North-Holland, Amsterdam, pp. 144-159.

Starr, D. W. and Mackworth, A. K. (1976) Interpretation-directed segmentation of ERTS images. Proc. ACM/CIPS Pacific Regional Symp., pp.69-75 also TR 76-3 Dept. of Comp. Sci., Univ. of B.C., Vancouver, 12 pp.

Waltz, D. L. (1972) Generating semantic descriptions from drawings of scenes with shadows. MAC AI-TR-271, M.I.T., Cambridge, Mass.

PICTURE REPRESENTATION WITH ICONIC/SYMBOLIC DATA STRUCTURES

Steven L. Tanimoto

The University of Connecticut

ABSTRACT

A key problem in computer vision is the structuring of pictorial knowledge. Two standard forms for representing pictorial knowledge are the icon or light-intensity matrix, and the relation, expressed either as a set of n-tuples or as a graph. Each form has distinct advantages and disadvantages. The forms may be combined to capture advantages of both using a method described here. The "iconic/symbolic" data structures resulting from this synthesis may be used in efficient pictorial-information retrieval systems by employing an "iconic indexing" technique. Interesting problems regarding minimal representation and easy access arise in the design of iconic/symbolic data structures.

I. INTRODUCTION

A key problem in computer vision is the structuring of pictorial knowledge. Knowledge about how things look, and the contexts things may appear in, is believed essential to sophisticated image analysis processes [8,2]. Two standard forms for the representation of pictorial knowledge are the icon or light-intensity matrix [10], and the relation, expressed either as a set of n-tuples or as a graph [7]. The icon has the advantage that many spatial properties may be stored implicitly (shape, proximities, texture, contrast, etc.) without the necessity of explicitly naming them. The icon has a disadvantage, however, of requiring considerable space when fine resolution is desired. Another disadvantage is that any features or "high-level" properties of the image must be extracted from scratch; this may be computationally costly if features are required frequently.

The relation has the advantage of being compact when the number of properties to be stored is small. Access to that information can be fast when the relationship between query and relation is simple. On the other hand, relations may easily be overcommitted to an insufficient set of image properties.

They also suffer from a loss of the cartesian simplicity of the icon and its simple accessing mechanisms.

A rich class of representation schemes is based on combinations of the iconic and symbolic forms. In one such scheme, relations are defined on icons; icons become the nodes of networks [13]. A dual methodology promotes building icon-like structures out of relations. It is possible to generalize both these notions to get a recursively-defined class of structures that admits the embedding of symbols in icons and icons in relations. Such structures which we call "iconic/symbolic data structures" can replace a number of picture-processing data structures appearing in the literature. A simple iconic/symbolic data structure which we call an extended icon is essential to an information retrieval method called iconic indexing. Iconic indexing allows rapid access to any symbolic information that is "spatially predicated" or positioned in cartesian space.

II. ICON AND SYMBOL

An icon is an image or pattern which bears a "natural resemblance" to that which it represents. The ability to "understand" an icon depends primarily on an ability to extract pictorial features and discover shape, discover structure and recognize objects. It does not depend heavily on an ability to recall definitions (retrieve bindings) of symbols.

A symbol may be considered an object whose value is associated with it without regard to any physical properties of the symbol itself. Thus a symbol may bear an apparently arbitrary relation to the value it is bound with and need not share any visual or other "likeness" with that which it represents.

The distinction we have just drawn is functional. Whether a given object is an icon or an abstract symbol depends contextually on the association mechanism used to determine what the object represents. Hence, there arise situations in which the distinction is easily confused. For example, a representation of (part of) the Vancouver skyline, digitized from a postcard photo-

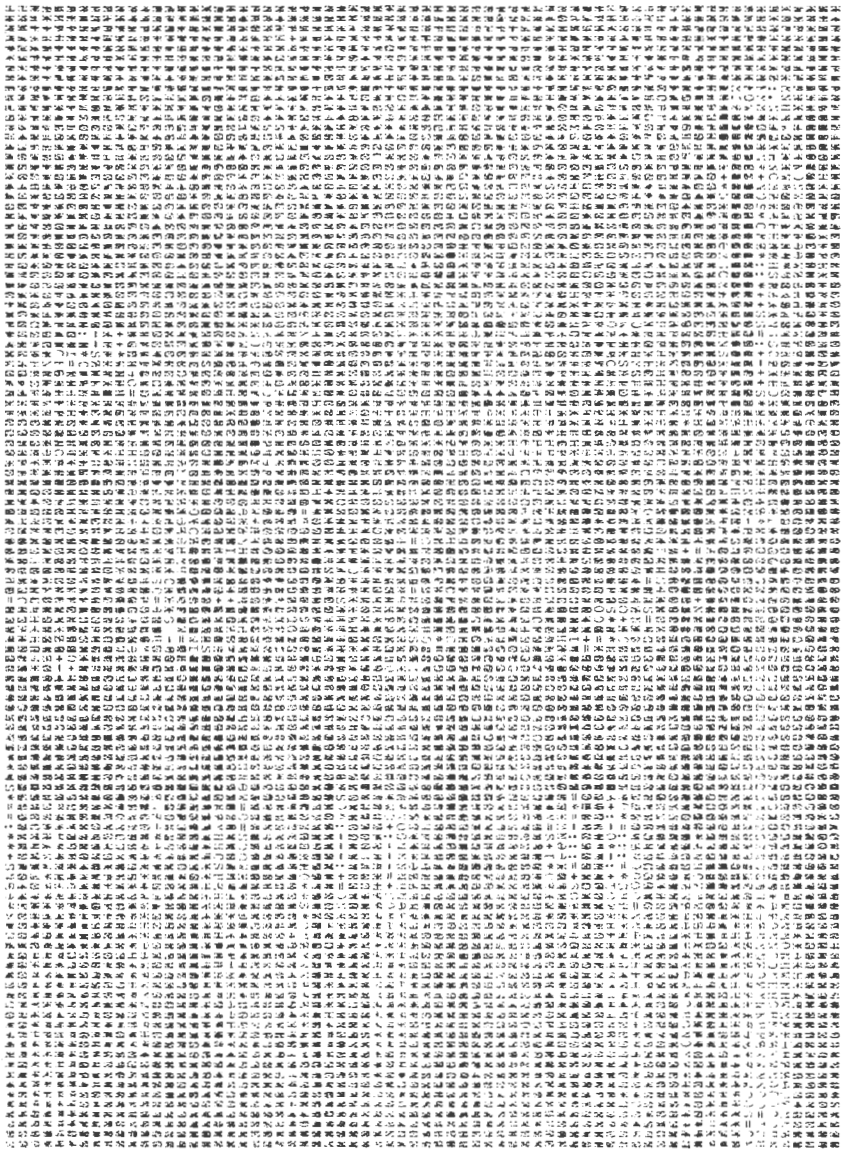


Figure 1
An iconic Vancouver skyline constructed from symbol forms.

graph is shown in Figure 1. Although the pictorial information is presented iconically, local gray values have been expressed by modulation of inked area via selection of alphanumeric symbol forms [4]. A more confusing example shown by Knowlton and Harmon [6] also consisted of an icon made up of symbol forms but the symbols forms themselves spelled out a text that had clear meaning.

Another source of confusion is the use of objects normally regarded as icons as symbols. For example, in Egyptian Hieroglyphics elaborate images have a 1-1 correspondence to phonemes (a finite number of them). Thus these images may be considered members of a finite alphabet, and their occurences in strings makes them symbols functionally.

Stylized icons representing eyes are also used as letter O's in Figure 2, illustrating another means of achieving iconic and symbolic communication simultaneously.

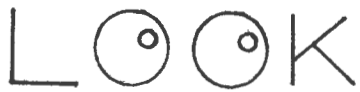


Figure 2
Letter forms may be augmented to communicate both symbolically and iconically.

A source of greater difficulty comes from Art. The intentions of a painter are not always clear to an observer. A canonical question which arises is whether a given object in a painting, say a tree, is to have special symbolic significance (reference through allegory, etc.) or is the iconic designation of a probable object without special significance. A complete knowledge of the context in which the painting was created would be required to resolve the ambiguity [1]. The lesson from all these difficulties in the distinction between the iconic and the symbolic is that the mode of interpretation is all-important.

Let us now consider the purposeful juxtaposition of iconic and symbolic techniques in data structuring.

III. ICONIC/SYMBOLIC DATA STRUCTURES

There are two simple ways to combine iconic and symbolic data structuring methods. The first has been used by Tenenbaum et al [13] for interactive scene analysis. In this scheme, the symbolic or relational portion of the data structure is expressed as a graph and the iconic element is introduced by letting some of the nodes be icons. Thus a semantic net [9] of related objects may have object descriptions that are either iconic or symbolic or both. The iconic descriptions, while sometimes requiring more time to access specific facts, are less committed to a small set of specific facts than are the symbolic descriptions. In addition to the simple icon, an iconic node of their net may be a reference to a subpicture of a simple icon; this permits flexibility and storage efficiency.

The second way of combining the iconic and symbolic may be considered a dual to the first in the following sense. Where in the first method icons may replace nodes of the symbolic structure (expressed above as a graph), in the second method, symbols may replace intensity information in the pixels of an icon. Thus we extend the notion of an icon to be an array of pixels which now may represent either light intensities (iconic information) or any object in a given universe (symbolically). The overhead associated with this option is 1 bit per pixel for a flag that indicates whether the pixel is iconic or symbolic. The primary advantage of this method is that the original icon may be efficiently used as a pictorial index to other information, represented either iconically, symbolically or by a mixed scheme.

We call an icon or picture matrix in which pixels cannot only represent local light qualities (eg., hue, intensity) but also arbitrary data items an extended icon. We now specify this using the PASCAL programming language [14,5].

The use of such methods to describe picture data structures was suggested by Zahn [15], where it was mentioned that definitions of lower level structures (eg., intensity ranges) could easily be changed without necessitating rethinking of the higher level structures (eg., "picture" in his case). In this set of definitions, we assume that a mechanism with a symbol table already exists to associate symbols with other entities. Thus, we are concerned only with the relationship of symbols to the extended icons of which they may be parts.

```
constant    black = Ø;  white = 127;
            symbol_Ø = Ø;  symbol_127 = 127;
            pix_size = 64;

type        indicator = (iconic, symbolic);
            intensity = black .. white;
            symbol = symbol_Ø .. symbol_127;
            pixel = record
                      case ind : indicator of
                        iconic : (intens : intensity);
                        symbolic : (symb : symbol)
                    end;
            pix_range = 1 .. pix_size;
            extended_icon = packed array [pix_range, pix_range]
                            of pixel;
```

The data type extended_icon would in this case be implemented as a 64 x 64 array of 8 bit byte pixels. Each pixel has one of its 8 bits used as an indicator, flagging it as either iconic or symbolic. The remaining 7 bits give either the intensity value or the symbol number. Such a pixel may be called an iconic/symbolic pixel.

Let us now conceive of a more general notion of "symbol" so that it may be considered a pointer to some structure. Allowing this pointer to link to similar extended_icon structures we get a class of structures that can be defined recursively. A more detailed treatment of this was given in an earlier

paper by the author [11]. We refer to such a data structure as an iconic/symbolic data structure (ISDS).

The advantages of ISDS's are their efficiency of space, representational power and the convenience they provide for pictorial information query answering. It is instructive to mention that the use of symbolic pixels can be avoided, keeping much the same general structure, but that a corresponding sacrifice of either accessing efficiency or storage efficiency will result. Figure 2 shows an ISDS and Figure 4 shows a similar data structure in which the symbolic pixel has been avoided.

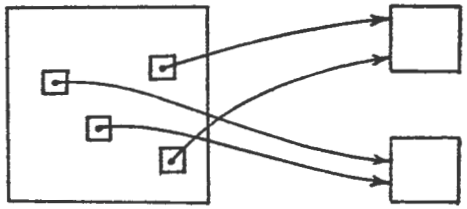


Figure 3
A simple ISDS employing symbolic pixels. Iconic indexing can work efficiently to retrieve an object.

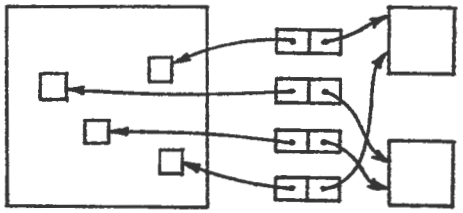


Figure 4
Another data structure representing identical information without the use of symbolic pixels. Iconic indexing requires an extra pass through a table of links.

IV. ICONIC INDEXING

We now summarize a pictorial-information retrieval system that uses an ISDS to speed responses to queries.

A one-level iconic index is an extended icon whose symbolic pixels point to the data items of the pictorial database. The iconic search mechanism is a procedure capable of accepting a query, exploring the iconic index, and in case of success, returning one or more symbolic pixels whose locations satisfy the iconic constraints of the query.

The general form of query answerable by the proposed system is "Retrieve the information associated with X" where X falls into a category such as one of the following:

(a) the locality of (X,Y). A search of the locality of (X,Y) is performed. If a symbol is found, its value is retrieved; otherwise the search fails. This kind of search may be tremendously sped up with the addition of an auxiliary "pyramid of symbol locations" where a small hierarchy of binary pictures is constructed from the projection of the extended icon consisting of the indicator bit of each pixel (see [11,12]).

(b) A point reachable from (X,Y) by moving in direction D = Θ. Pixels along a path of some width W starting at (X,Y) and moving in direction D (i.e. angle Θ measured clockwise from North) are examined until a symbol is found, a picture border is encountered, or a maximum number of pixels have been examined. This search may also be sped up through use of pyramids. An alternative to searching in a path is searching a sector that radiates out from (X,Y).

(c) Others including "A feature point of type F", and "Color C".

The efficiency of iconic indexing lies in its employment of pyramid search. Let "system A" use iconic indexing and "system B" answer the same queries without iconic indexing. Let R be the area of the locality to be searched and let N be the number of symbols in the ISDS. System A searches the locality of the index

quickly, employing pyramid searching to cover an area R in $\log_2 R$ steps.

System B must scan through a list of the coordinates of symbols (say N items) calculating distances to the starting point to find that symbol achieving the minimum distance. Binary search cannot be used effectively to achieve $O(\log_2 N)$ search time because location of an item "close to" (X_o, Y_o) cannot be done simply by searching on X and then searching on Y. Projected distances along X or Y can be used only as lower bounds on the actual distance between a symbol and the starting point. System B uses O(N) time to execute a local search.

In a worst case for system A and moderate case for system B, assume R is the whole icon, and $N = \sqrt{R}$. Clearly as R and N grow, the search time for A grows only very slightly compared with that of B.

In practice, symbols will usually be located near distinctive feature points of the icon such as corners, local brightness extrema, etc. This will tend to keep R small giving a distinct advantage to system A. Difference in performance increases as both the number of symbols in a picture increases and the number of queries to be processed increases.

V. RELATED PROBLEMS

A geographic map is a good example of an everyday iconic/symbolic representation scheme. In order to represent image segmentations, the combination of region boundaries and symbolic labels can also be efficient. Storage and access time tradeoffs result from considerations of the number of iconic and symbolic pixel values, coding schemes for concatenating symbols and the possibility of letting symbols represent small iconic subimages.

Segmentation may be guided by a special ISDS whose symbols identify specific procedures to be used to process the localities of a test image corresponding to the localities of the symbols in the ISDS guide. Such a scheme is a generalization of existing techniques [3].

1. Gombrich, E. H., Art and Illusion: A Study in the Psychology of Pictorial Representation. Princeton, N. Y.: Princeton Univ. Press (1960).

2. Hanson, A. R. and Riseman, E. M., The design of a semantically directed vision processor (revised and updated). COINS Tech. Rept. 75C-1, Univ. of Mass., Amherst, (Feb. 1975).

3. Harlow, C. A. and Eisenbeis, S. A., The analysis of radiographic images. IEEE Trans. on Computers C-22, 7 (July 1973) pp. 678-689.

4. Henderson, P. and Tanimoto, S., Considerations for efficient picture output via lineprinter. Computer Graphics and Image Processing 3 (Dec. 1974) pp. 327-335.

5. Hoare, C. A. R., Notes on data structuring, Structured Programming by O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Academic Press (1972) pp. 83-174.

6. Knowlton, K. and Harmon, L., Computer-produced grey scales. Computer Graphics and Image Processing 1 (1972), pp. 1-20.

7. Kunii, T. L., Weyl, S. and Tenenbaum, J. M., A relational database scheme for describing complex pictures with color and texture. Second Int'l Joint Conf. on Pattern Recognition, Copenhagen (Aug. 1974). IEEE #74CH0885-4C, pp. 310-316.

8. Minsky, M., A framework for representing knowledge. In the Psychology of Computer Vision, P. Winston (Ed.), N. Y.: McGraw-Hill (1975), pp. 211-277.

9. Preparata, R. P. and Ray, S. R., An approach to artificial nonsymbolic recognition. Information Sciences 4 (1972) pp. 65-86.

10. Rosenfild, A., Picture Processing by Computer. N. Y.: Academic Press (1969).

11. Tanimoto, S. L., An iconic/symbolic data structuring scheme. Presented at IEEE Computer Society Joint Workshop on Pattern Recognition and Artificial Intelligence, Hyannis, Mass., June 1-3, 1976.

12. Tanimoto, S. L. and Pavlidis, T., A hierarchical data structure for picture processing, Computer Graphics and Image Processing 4, 2 (June 1975) pp. 104-119.

13. Tenenbaum, J. M., Garvey, T. D., Weyl, S. and Wolf, H. C., An interactive facility for scene analysis research. Stanford Research Institute A. I. Center, Tech. Note 87 (Jan. 1974).

14. Wirth, N., The programming language PASCAL, Acta Informatica 1, 1 (1971) pp. 35-63.

15. Zahn, C. T., Data structures and pattern recognition algorithms: a case study, Proc. Conf. on Computer Graphics, Pattern Recognition and Data Structure, Beverly Hills, CA., (May 1975) pp. 191-195.

THE C2 "SUPER-COMPILER" MODEL OF AUTOMATIC PROGRAMMING

Ted J. Biggerstaff
and
David L. Johnson
University of Washington

ABSTRACT

The C2 synthesizer is constructed of "specialist programs" called strategies. Each strategy synthesizes a single class of target programs (e.g., searches, list rewriting programs, etc.). Each strategy uses three kinds of specification information to synthesize its target programs: syntactic, semantic, and pragmatic. IO examples provide the syntactic information and this information manifests itself as target program actions (e.g., "pushes"). Evaluation of the "class loop invariant" form of the IO Specification upon the IO examples provides the semantic information which ultimately appears as target program branching predicates. The strategies themselves provide the pragmatic or planning information by imposing a specific design upon the target program. The basic synthesis procedure uses these three specification entities to simulate loops within the target program. The source code for the target program loop is inferred from the protocol of this simulation.

1.0 INTRODUCTION

One can conceive of the task of constructing specialist synthesizers for specific classes of programs as the development of a "super-compiler" for a new class of high-level language. Members of this class will be called "C2" languages. Each statement in such a language will correspond to one class of programs (e.g., searches) and will "compile into tens of

statements in some C1 language (i.e., any currently existing "high level" language).

The target program class to be discussed by this paper will be list rewriting programs. These are programs which return a list as their result. This list is constructed by combining and restructuring one or more input lists. Examples of programs which fall into this class are a sort program, all basic set operations (e.g., UNION), a program which copies up to N items which are either even or occur in even numbered positions of the input list, a program which deletes subsequences of equal and adjacent objects, etc.

2.0 PROGRAM SPECIFICATION

The C2 theory of program specification defines three interdependent elements of specification: IO examples, an IO Specification program, and a plan or design for the target program. Table 1 contains example specifications for three target programs. This paper will follow COPYE through the synthesis process.

The first component of specification is a set of target program IO examples. These are given in terms of abstract data and provide some specific examples of target program behavior. (NOTE: Abstract data items are symbolic constants. Their attributes and relationships must be specified symbolically, e.g., (EVEN A), and must be managed by a MICRO-PLANNER like program [4].) For example, COPYE's second IO example describes all possible COPYE behaviors for an input list of length one. That is, if the input list is (A), then the output will be either NIL or (A). For a specific A, which behavior COPYE exhibits

TARGET PROGRAM	COMPONENTS OF SPECIFICATION		
	SYNTACTIC IO Examples describe structures and transformations of structures.	SEMANTIC The IO Specification describes the attributes of and relationships among inputs and outputs.	PRAGMATIC The plan or design to be used to synthesize the target program.
COPYE (L) Copy all top level elements of L which are even integers.	L (Input) / N (Output) NIL / [NIL] (A) / [NIL,(A)] (A B) / [NIL,(A),(B),(A B)] . . . / . . .	DEFPRED (RCOPYE (L N) (FORALL X L (IFF (IN X N) (EVEN X))) "For all X in L, X is in N iff X is even."	Comment: GET-FIND-PUT design; Loop1: X ← GET next element from L; Loop2: FIND where X goes in N, and PUT it there; ENDLOOP2; ENDLOOP1;
UNION (L,M)	L / M / N NIL / NIL / [NIL] (A) / NIL / [(A)] ... / ... / ... (A) / (B) / [(A),(A B)] ... / ... / ...	DEFPRED (RUNION (L M N) (AND (SET N) (FORALL X (L,M,N) (IFF (MEMBER X N) (OR (MEMBER X L) (MEMBER X M)))) (PRECOND) (AND (SET L) (SET M))) "Given that L and M are sets, N will be a set and forall X in L,N, and M, X is a member of N iff X is a member of L or M."	GET-FIND-PUT (GFP) design.
SORT (L)	L / N NIL / [NIL] (A) / [(A)] (A B) / [(A B),(B A)] . . . / . . .	DEFPRED (RSORT (L N) (AND (SETEQUIV L N) (FORADJ X Y N (LE X Y))) "L and N contain the same elements and for all adjacent X and Y's in N, X is less than or equal to Y."	GFP design.

TABLE 1
ELEMENTS OF PROGRAM SPECIFICATION

will, of course, depend upon the attributes of the specific A. IO examples provide a simple and direct method of describing structural information and target program actions.

The second component of specification is the IO Specification program. It describes the attributes of and relationships between the target program inputs and outputs. This program is written in a superset of LISP and is evaluated by a method (abstract evaluation) which is a generalization of LISP evaluation. Abstract evaluation extracts target program predicates from the IO Specification program.

The final component of specification is the design or plan to be imposed upon the target program. This component is a LISP program called a "strategy" and as such, is difficult to characterize in any level of detail. Table 1 contains a highly abstracted formulation of one representative target program control structure which can be produced by this design. In actual fact, an infinite variety of target program control strucures may be produced by this strategy.

The strategy to be discussed in this paper is the GFP or GET-FIND-PUT strategy. Programs synthesized by this strategy all process their input list(s) from front to back. That is, they consist of an outer loop which GETs each element in turn from the input list. The "main step(s)" of this loop FINDs where the element is to go on the output list and PUTs it there. This FIND-PUT operation may be a single step as with COPYE, or an inner loop as with SORT.

3.0 THE SYNTHESIS ALGORITHM

Recall that the C2 loop synthesis algorithm simulates the behavior of target program operating upon abstract data, makes a record (the Function-Strategy or FS tree) of this behavior for the first few iterations of a loop, and then uses induction and generalization to synthesize a loop which would exhibit that behavior. The following discussion describes the synthesis of COPYE through the four basic steps of the algorithm.

Step 1: This step records the behavior (i.e., states) of the target program's internal data structures for the first few iterations of the loop. For any specific iteration, Figure 1 shows a schema segment which could produce the behavior assumed by the GFP design. The GFP strategy will construct a set of such schema segments for specific iterations and then will combine them into a generalized form which not only accounts for all specific behavior described by the IO examples, but accounts for the general behavior implied by the IO examples as a whole.

Figure 2 is the shorthand form for the schema segment in Figure 1. Notice that information which is static (e.g., the GET) or information which is implied by the iteration number (e.g., the state of the input list) is not explicitly shown on the shorthand form. In fact, only the state of the output list need be shown.

Now these implied schemata must be connected together into a behavior tree, such as the one for COPYE shown in Figure 3. This tree can be conceived of as the "unwound" form of COPYE's loop operating upon an input list of (A B C ...). The loop is unwound in the sense that every iteration is represented by separate levels in the behavior tree.

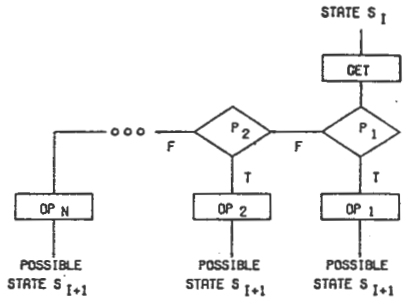


Figure 1
GFP Schema

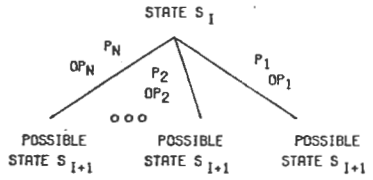
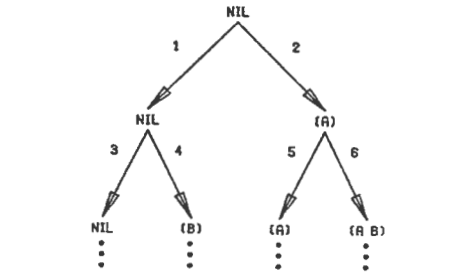


Figure 2
Shorthand for Schema



BRANCH NO.	PREDICATE	ACTION
1	(ODD A)	(NOP«)
2	(EVEN A)	(CREATE« A N)
3	(ODD B)	(NOP«)
4	(EVEN B)	(CREATE« B N)
5	(ODD B)	(NOP«)
6	(EVEN B)	(APPEND« B N)

Figure 3
FS Tree for COPYE

The "class loop invariant (CLI)," for list rewriting programs with one input list, tells us how to construct this tree. The CLI may be expressed by the formula R(L[0]/L[i],N[i]) where L[0]/L[i] is the prefix of the input list which is i elements long, N[i] is the i-th state of the output list, and R is the IO Specification. Using the relationship between a program f and its IO Specification R, the CLI reveals that f(L[0]/L[i]) = N[i]. In other words, the example outputs in Table 1 correspond to the states of the output list when COPYE is applied to the input list (A B C ...). Thus, the example outputs from Table 1 are used to construct COPYE's FS tree shown in Figure 3.

Step 2: Synthesize operators (or actions) which will produce the given transformations of the output list. Two pieces of information are needed to synthesize these operators: 1) The "before" and "after" states of the output list, which are stored in the FS tree, and 2) the value of the i-th element from the input list.

Within the GFP design, there exists a conceptual object called "current item being processed." This object corresponds to a target program variable, X. For an input list of (A B C ...), X is bound on successive iterations to A, then B, then C, etc. Thus, operator synthesis for any specific iteration i is the process of determining what operation would combine X[i] and N[i] to produce N[i+1]. For example, for branch 6 of Figure 3, C2 determines that B can be appended to (A) to produce (A B).

Step 3: Compute the discrimination predicates P1, P2, etc., which determine the conditions under which the target program would have exhibited the respective behaviors A1, A2, etc., and affix these predicate expressions to the FS tree.

Consider branch 6 in Figure 3. Predicate synthesis is realized through two substeps, both of which use the abstract examples and the IO specification shown in Table 1. The first substep constructs the state of the target program's computation when the output variable has the value (A). The second substep computes the predicate which must be true in order for the value of the output variable to be transformed into (A B). Both substeps are applications of the CLI described above.

The first substep is achieved by ASSERTing (in the MICRO-PLANNER [4] sense) the results of abstract evaluation of the expression: (RCOPYE L N), where L's value is "(A)" and N's value is "(A)." The result of this substep is that the expression (EVEN A) is entered into the data base. Thus, we have asserted that the abstract constant "A" is an even integer for this state of the computation even though we do not know the exact value of A.

The second substep of predicate synthesis is achieved by abstract evaluation of the expression (RCOPYE L N) where L's value is (A B) and N's value is (A B). The result is the expression: (EVEN B) which is affixed to branch 6 of the FS tree.

Abstract evaluation (treated in greater detail in [1,2]) is a generalization of LISP evaluation which maps an expression into a more specialized version of that expression. Similar schemes are developed in [3,5]. It is a process of partial execution in which an expression is applied to data, some of which is real and some of which is abstract. The result is a new, simpler expression. Then for a specific answer X in the range of the

expression, the original expression maps to X if and only if the new expression maps to X.

Step 4: Use induction and generalization to map from this record of behavior (the FS tree) to a loop which would exhibit the described behavior. The step consists of three interrelated activities: a) variablization, b) control structure development, and c) the integration of code implied by the GFP design.

Variabilization is the process of mapping from abstract data constants (e.g., A) to target program variable names or expressions (e.g., X or (CONS X)) which would have those abstract data objects as their value at the given state of the computation. There may be several candidates for this mapping, and C2 uses a variety of information to choose the most appropriate.

The control structure for the loop being synthesized is derived by grouping FS tree branches into equivalence classes based upon matching predicate expressions, operator expressions, and subtrees of matching nodes. Finally, C2 adds information such as the GET step, which is implicit in the GFP design. Figure 4 is the completed program.

```
DEFUN (COPYE (L)
       (PROG (X N)
L1      (OR L (RETURN N))
        (SETQ X (CAR L))
        (SETQ L (CDR L))
        (AND (EVEN X)
             (SETQ N (APPEND N (CONS X))))
        (GO L1)]
```

Figure 4
Form of COPYE Synthesized by C2

4.0 CONCLUSIONS

One of the most important aspects of C2 is its ability to synthesize complex LISP programs. An implementation of C2 on the CDC 6400 has synthesized all programs described in the introduction as well as those mentioned throughout the paper.

The concepts of the class loop invariant and abstract evaluation are the fundamental ideas underlying the C2 model. They allow natural, flexible and powerful program specification and generation schemes. It is these ideas which give C2 an open-ended character allowing the basic C2 framework to be applied to a wide number of diverse program synthesis problem domains.

REFERENCES

1. Biggerstaff, Ted J., "C2: a 'super-compiler' approach to automatic programming. Ph.D. Dissertation, University of Washington, Seattle, Washington (January 1976).
2. Biggerstaff, Ted J., Automated consultation. North West 76, Seattle, Washington (June 76).
3. Burstall, R. M. and Darlington, John. Some transformations for developing recursive programs. International Conference on Reliable Software, (1975).
4. Sussman, Gerald J., Winograd, Terry, and Charniak, Eugene, MICRO-PLANNER reference manual. MIT AI Memo 203A (December 1971).
5. Wegbreit, Ben, Goal-directed program transformation. IEEE Transactions on Software Engineering, Volume SE-2, Number 2, (June 1976).

Making Computers "Understand" Programs

Lucio F. Melli
Department of Computer Science
University of Toronto

Abstract

It has been proposed that the "complexity barrier" presented by large programs may be conquered by programming systems which are capable of "understanding" programs and of acting as assistants to programmers. Some systems which employ this idea are examined. Our current and proposed research towards the specification and clarification of the facilities and information requested from a programming advisor system is also discussed.

Introduction

Large programs present large problems: they are difficult to write, debug, test, and modify. This "complexity barrier", as Winograd so aptly describes it in [1], may be conquered by programming systems which are capable of understanding programs and of acting as assistants to programmers. It is the goal of this paper to present some of the work being done towards such a programmer's assistant.

The paper uses Winograd's proposed "A" system as a starting point since it presents the most comprehensive and integrated description of what such an assistant should offer. We then examine some different interpretations of "assistance", particularly [2,3,4,5]. Next, the notion of a programming advisor system is compared and contrasted to that of an idealized assistant such as the "A" system. (The differences extend beyond nomenclature.) Finally the paper discusses our current and

proposed research towards the specification and clarification of such a programming advisor system.

Towards a Programmer's Assistant

This section examines Winograd's proposed "A" system [1] and some of the different interpretations of "programmer's assistant".

The "A" System

In [1], Winograd takes some "in the air" ideas and uses them as a possible solution to the complexity barrier presented by large programs. He claims that this barrier may be conquered by a highly interactive programming system which is capable of "understanding" programs and of acting as the programmer's assistant. The assistant is "moderately stupid" and it is intended to relieve the programmer of the burden of memory work, checking, and drawing more-or-less straightforward conclusions. To be able to do this, the assistant is a reasoning system with models of the programming world and of the programs written in it. The assistant is not intended to be an automatic programmer, nor a program checker which guarantees correct and efficient programs. It is supposed to be a system which helps to magnify the programmer's effort and aid in producing his programs.

To achieve these goals, Winograd proposes the "A" system and identifies four specific ways in which assistance may be provided: error checking, question answering, trivia, and debugging. For error checking and question answering, assistance would go beyond the purely syntactic level since "A" would use its semantic model and deductive capabilities to offer a higher

level of aid to the programmer. By trivia, Winograd means the obvious expansions where the programmer is not interested in how but rather in what is done. This does not imply automatic programming but rather something more sophisticated than the expansions performed by compilers. For debugging, "A" would make use of its deductive capabilities and the model of the program to find possible bugs.

As would any assistant, "A" would need information about the program being considered. This is accomplished by the use of conditions, assertions, purposes, and English. These "comments" provide successively higher abstract descripticns of what the code does. Since "A" can only respond according to the amount of information it is told, the programmer may be encouraged to better "comment" his program to the assistant.

It should be pointed out that "A" itself presents the complexity barrier which it is trying to conquer. However, Winograd is confident that by careful planning "A" can be used to help in its own development by a kind of bootstrapping process.

Other Assistants

The systems of Goldstein [2], Ruth [3], and Sussman [4] all view the assistant as a debugger. Goldstein's MYCROFT [2] understands (i.e. debugs) LOGO turtle programs written by beginners. MYCROFT requires from the user a description of intent (a model) which specifies geometric predicates of the picture to be drawn. MYCROFT also requires a plan, either user supplied or obtained from an analysis of the program. Given the plan, the model, and the program the system interprets the produced picture and notes inconsistencies. Debugging is based

upon correcting the discrepancies by using general debugging knowledge, and imperative gecmetric knowledge (i.e. manipulating the turtle to obtain a desired geometric relationship). A program is debugged if it draws the intended picture.

Ruth [3] is more interested in the application of program analysis to CAI. He feels that a program analyser which can understand how the student is trying to implement his task and what implementation errors (if any) he has made, can be more beneficial as a teaching tool than the standard CAI approaches. In particular, the analyser is concerned with programs that sort an array of numbers using a restricted class of algorithms. A formal generative system (similar to a formal grammar) is used to derive the set of all "reasonably" implemented sorting programs. The system also contains a built-in body of knowledge about how intentions can be realized through code, and the common sources of error in program writing. The generative system uses this knowledge to perform a top-down parse of a program and ccmprehend, verify, and if need be, correct it.

Sussman is concerned with skill acquisition and has developed HACKER [4], a problem solver for Blocks World tasks whose performance improves with practice. HACKER possesses an Answer Library of programs with associated patterns of applicability. When a prcblem is received, the library is searched for a pattern which matches the problem statement. If no program is found, HACKER writes a new program using some general knowledge of programming techniques and knowledge of the Blocks World. If a bug (a failure) is encountered when a program is being run, general debugging knowledge is used to classify and understand the failure. The program is then patched to work for the problem

case. In addition, an attempt is made to generalize and remember the bug so as not to repeat it when constructing a later program.

It should be pointed out that the above three systems are some of the "in the air" ideas alluded to in Winograd's paper. It is interesting to note the similarities of the three systems, particularly their need and use of knowledge of general programming, debugging, and the problem domain.

The system of Mikelsons [5] is concerned with automating the interactions of a naive user and a set of highly parametrized application programs (i.e. business accounting procedures). The system contains a representation of the available programs and options (the program model), and the assumptions made by the programmers about the application domain and the intended relationships between program and application concepts (the application model). The selection of the appropriate parameters for the programs, and therefore the generation of a specific set of application programs, results from a natural language dialogue between the customer and the system. (Note that the customer makes the choice after obtaining answers to his questions.)

A Programming Advisor

This section examines the reasons for choosing a programming advisor system as a step towards a programmer's assistant. It also describes our current and proposed research in this area.

Why an Advisor?

One of the more common fixtures of a computer center is the programming advisor. Using his expertise of programming, he is capable of explaining cryptic error messages, answering questions

about the syntax and semantics of programming languages, detecting obvious errors, pointing out potential bugs, and providing other forms of assistance to the programmer. Since there are usually several persons seeking assistance, direct questions are preferred over vague or general questions.

There exist obvious similarities and interesting differences between a programming advisor and the assistant described by Winograd. They both assist with error checking and debugging, and they both answer questions about programming in general or a specific program. Differences between the two involve the size of programs under consideration, the kinds of expected users, and the types of questions encountered. The advisor usually deals with small programs or small sections of large programs, and caters mainly to beginning programmers whose knowledge of programming is quite limited. Even when a more experienced programmer uses the services of the advisor, the questions tend to be direct and not very sophisticated. On the other hand, the assistant would aid sophisticated users to develop large programs. It therefore would have to handle more complex requests which would require a higher degree of understanding.

It seems that the assistant subsumes the facilities provided by an advisor and therefore can be described as offering a higher level of assistance. However, the advisor has interesting properties which make it attractive as a candidate for research towards a programmer's assistant. As previously mentioned, the advisor handles more direct questions which may possibly simplify the deductive capabilities required. The small programs will prove easier to model. Although the fear exists that what works for small programs may not work for larger programs, one must

take into consideration that to date there is no consensus as how to model programs so that they can be "understood" by a system. Another attractive factor is the great demand for advisors. This simplifies the gathering of information for the specification and the clarification of the tasks of the advisor, and provides a large population to test an implemented program advising system.

Goals and Methodology

Our research towards a programmer's assistant is concerned with two major objectives:

1) Obtaining a clear specification and understanding of the kind of assistant we wish to build. In particular, we are considering a programming advisor system to help novice programmers with their programs.

2) Developing a methodology for obtaining such specifications.

To meet these objectives, we are collecting dialogues between a beginning programmer (i.e. a student enrolled in a first year computing course) and a programming advisor. The dialogues are carried out over two terminals connected to a controlling program which alternatively allows the terminals to send and receive messages. A record of the transmitted messages is kept. The reason for using written rather than spoken communication is to encourage students to be more precise in their formulation of the questions. Written communication also contains fewer "noise" words (i.e. words that do not add to the understanding of an utterance) than spoken communication does.

In order to limit the content of the dialogues, the domain of discourse of the advisor has been restricted to:

1) the problem to be solved;

2) the specific program attempting to solve the problem; and

3) the programming language used. (We are using SP/K, a subset of PL/I.)

In spite of the restrictions, questions can still be very general (e.g. "What is wrong with my program?" or "How do I fix it?"). Since specific information is desired, the advisor frowns upon such questions and demands more specific ones.

The programmer is also warned that although the advisor is an "expert" on various aspects of programming and the programming language, there are things he does not (and cannot) know unless he is told, such as information about the method of solution and assumptions made, or the relationship between parts of the program and the problem. Therefore it is important that the programmer explain to the advisor things that are relevant to understanding the program.

There is no attempt made to fool the students into believing that they are dealing with anyone but a human advisor. They know the reasons for carrying out the dialogues, and they can see the advisor during the dialogues since the terminals are situated in the same room. However, all communication during the dialogues is carried out via the terminals. The same person always plays the role of the advisor to ensure a certain amount of consistency in the types of questions accepted and the the answers given. When giving an answer, care is taken to understand how such an answer is obtained and how it could be given by a computer system (if possible).

It should be noted that our goals and methodology are very similar to those of Malhotra and Sheridan [6]. However, the application areas are different (they were interested in the type

of system described by Mikelsons [5]), and the type of experiment we are conducting is less restrictive than the one they performed.

Future Research

To date, about 15 dialogues have been collected and we expect to have about 20 to 25 dialogues by the end of the summer. After this collection phase, a low level syntactic analysis of the dialogues will be carried out to determine such things as size of the vocabulary, grammatical structure of sentences, etc. In addition, a higher level semantic analysis will be done to identify protocols, kinds of questions asked, types of facilities offered by the advisor, information requested, etc. We are particularly interested in examining our dialogues to see if they can be described in terms of a grammar, as is done for the planning and debugging sessions of [7].

Conclusions

In conclusion, we have tried to make the reader aware of some of the different systems which are steps towards a programmer's assistant. We have also outlined our own research and the belief that an advisor is a logical step towards an assistant. We also believe that it is important to determine the specifications for the advisor before proceeding with an implementation. We feel that the methodology of experimentally collecting and analysing these dialogues will be helpful in the clarification of the kinds of facilities and information that are demanded from the advisor. With such knowledge it will be possible to examine the

feasibility of constructing a computer system to handle some of the aspects of advising beginning programmers.

Acknowledgements

I would like to thank Professor John Mylopoulos for his supervision and assistance.

References

1. Winograd, T.; "Breaking the Complexity Barrier Again"; SIGPLAN Notices, Vol. 10, No. 1, Jan. 1975, pp. 13-22.

2. Goldstein, I. P.; Understanding Simple Picture Programs; AI TR-294, MIT AI Lab, Sept. 1974, 228 pages.

3. Ruth, G. R.; Analysis of Algorithm Implementations; MAC TR-130, MIT, May 1974, 271 pages.

4. Sussman, G. J.; A Computational Model of Skill Acquisition; AI TR-297, MIT AI Lab, August 1973, 199 pages.

5. Mikelsons, M.; Computer Assisted Application Description; RC 5387, IBM Research, Nov. 1974, 32 pages.

6. Malhotra, A. and Sheridan, P.; Experimental Determination of Design Requirements for A Program Explanation System; RC 5831, IBM Research, Jan. 1976, 56 pages.

7. Goldstein, I. and Miller, M.; Intelligent Tutoring Programs: A Proposal for Research; AI Working Paper 122, MIT AI Lab, March 1976, 85 pages.

SOME COMPUTATIONAL STRUCTURES FOR A MODEL OF CONVERSATION

G. McCalla

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada[1]

Abstract:

This paper is concerned with the computational aspects of a representation of knowledge for the modelling of conversation. Frames are proposed as the basic information units of the system. General characteristics of frames, including their structure, procedural abilities, and message passing behaviour, are outlined; then, a specific kind of frame called a pattern expression (|PEXPR) is discussed, with particular attention being devoted to how it encodes its information and how it communicates this information to other frames.

Introduction:

I am currently constructing a computer model of conversation, with the eventual aim that it should take part in three dialogues which might occur at a symphony concert: a conversation to buy a ticket to the concert, one to buy a drink at intermission, and finally a discussion of the first half of the concert with a "friend". The main focus of the research has been on how linguistic abilities interact with the rest of the model's capabilities, rather than on a thorough analysis of language per se. Of primary interest has been the problem of representation, especially in designing the basic computational structures and determining how they connect with one another

---

[1]Most of the research in this paper was carried out while I was a Ph.D student in the Department of Computer Science, University of British Columbia, Vancouver, B.C. I would like to thank the many people at both U.B.C and U. of T. who through their comments and criticisms helped in the development of these ideas.

(both dynamically and statically).

This paper will briefly outline some aspects of the current representational scheme. Earlier versions have been encoded in a LISP program called |LISP, but the current version is not yet implemented.

The Representation:

In designing the representation, a major goal has been to maximize modularity in order to allow a variety of different knowledge structures to co-exist at the same time. To this end, the basic information unit is my version of the ubiquitous frame (Minsky (1974), Kuipers (1975), Winograd (1975), Charniak (1975), and many others), where a frame is essentially a black box containing the model's knowledge about a particular concept. A frame can encode its information in the way most suitable to the concept being represented, rather than in some more general formalism. Thus, a frame to interpret an utterance is mostly procedural; a frame representing knowledge about a ticket seller is largely declarative; the "Queen Elizabeth Theatre" frame might be an implicit binary matrix (similar, perhaps, to Funt's (1976) "direct representations") showing the relative locations of the lobby, seats, ticket-booth, etc.; and the FOO frame may have no structure at all (i.e. be primitive).

Frames communicate by passing messages to one another (much as do actors in Hewitt's formalism (Greif and Hewitt (1975)), except that here some sort of answer is guaranteed and the messages themselves are not frames). Messages are processed by

a special message handler attached to the receiving frame. This message handler must relate each message to the frame's internal structure so that an appropriate response can be generated and returned to the sender. A different message handler is, of course, required for each variety of frame internal representation.

There are many kinds of frames in the system. Most basic are atoms that have names but no structure. Unable to send or receive messages, such frames derive their meaning solely through their use elsewhere. Slightly more complex are frames of type SUBR, EXPR, and |EXPR, the bodies of which are LISP-style functions, and the messages to which are argument lists. More esoteric yet are a small class of frames which represent their knowledge in some more direct way (such as the Q. E. Theatre frame above). Since the internal representation for such frames tends to be very idiosyncratic, most "direct expressions" must have a specially tailored message handler.

All of these frame types are interesting, but they don't really constitute the majority of the system's structures. This status is reserved for something called pattern expressions (|PEXPRs). |PEXPRs can represent everything from declarative knowledge about the agenda of a symphony concert to procedural knowledge about how to take part in a conversation to buy something. Because |PEXPRs are so prevalent, the remainder of the paper will be devoted to describing them.

Pattern Expressions:

A |PEXPR is a frame whose "body" consists of a set of arbitrarily nested patterns, the elements of which are either frame names or pattern variables. A |PEXPR to represent the model's knowledge about a ticket seller could be

TICKET-PERSON

```
(ISA TICKET-PERSON SELLER)
(MESSAGE-HANDLER TICKET-PERSON |PEXPR)
(SELLS TICKET-PERSON TICKETS)
(EXCHANGE TICKET-PERSON BUYER ?GOODS
          !((COND ((EQ GOODS 'TICKETS) 'MONEY)
                  (T 'SERVICES)))
```

Messages to a |PEXPR are also patterns, handled by matching them against patterns in the body of the |PEXPR. Two patterns are said to match if and only if they are EQUAL in the LISP sense. The only exception to this occurs when certain macro characters (such as "?" or "!") indicate to the matcher the existence of a pattern variable rather than an ordinary element (much as do similar macros in languages such as MICRO-PLANNER or CONNIVER (Sussman and McDermott (1972))). A "?" preceding an element in a pattern tells the matcher to automatically match the corresponding element of the other pattern and to assign the matching element as the value of the ?-element in the current frame (name / value pairs are kept on a frame stack). Thus, (COMPOSE ?COMPOSER EROICA) would match (COMPOSE BEETHOVEN EROICA) and COMPOSER would be assigned the

value BEETHOVEN. A "!" preceding an element in a pattern indicates to the matcher that |EVAL, the |LISP interpreter, is to be applied to the element before matching; so that, (COMPOSE !COMPOSER EROICA) would match (COMPOSE BEETHOVEN EROICA) since the value of COMPOSER is BEETHOVEN. Other macros exist, but will not be described since they are not crucial at this point.

The best way of illustrating the somewhat subtle details of |PEXPR message passing is with an example. Assume the interpreter is working through the BUY1 frame (a |PEXPR that is controlling the ticket buying process) and reads the form (TICKET-PERSON (EXCHANGE !CONVERSANT BUYER TICKETS ?FOR-WHAT)). Discovering that the receiving frame, TICKET-PERSON, is a |PEXPR itself, the interpreter passes control to the |PEXPR message handler. The |PEXPR message handler immediately creates a new frame, TICKET-PERSON1, an execution instance of the receiving frame (similar to a Bobrow and Wegbreit (1973) extension except that an execution instance, being a |PEXPR itself, can be treated in the same way as any other |PEXPR rather than existing only as an unanalyzable programming construct). This new instance is endowed with a pointer to the generic receiving frame (in the static "ISA environment" of the instance), a pointer to the sending frame (in the dynamic "execution environment" or "context" of the instance), and an initially

empty stack. It might look like

TICKET-PERSON1

```
(EX-INSTANCE-OF TICKET-PERSON1 TICKET-PERSON)
(EX-ENVIRON TICKET-PERSON1 BUY1)
(STACK TICKET-PERSON1 ())
```

The message pattern (EXCHANGE !CONVERSANT BUYER TICKETS ?FOR-WHAT) is then matched against patterns in TICKET-PERSON1 (conceptually, at least - in reality, efficiency considerations will normally lead immediately to a search of TICKET-PERSON) and no match is found. In such cases, the message handler requests help from the first frame of the message pattern (EXCHANGE), since this frame is usually the most important in a pattern. EXCHANGE suggests going into the ISA environment of TICKET-PERSON1 to find a match. Looking up into TICKET-PERSON, the pattern

```
(EXCHANGE TICKET-PERSON BUYER ?GOODS
          !((COND ((EQ GOODS 'TICKETS) 'MONEY)
                  (T 'SERVICES)))
```

is discovered.

The matcher compares the message pattern (handling "?" and "!" in the context of BUY1) with the TICKET-PERSON pattern (doing "?" and "!" in the context of TICKET-PERSON1). EXCHANGE matches EXCHANGE. !CONVERSANT tells the matcher to find the value of CONVERSANT on the BUY1 stack; and if this is TICKET-PERSON, then the second elements match. BUYER matches BUYER. TICKETS matches ?GOODS with GOODS being assigned the

value TICKETS on the stack of TICKET-PERSON1. Finally, ?FOR-WHAT matches !(|COND ...), and since !(|COND ...) computes to MONEY in the context of TICKET-PERSON1, FOR-WHAT is assigned the value MONEY on the BUY1 stack. A satisfactory match has thus been achieved. Note how the judicious use of "!" and "?" allows |PEXPRs to obtain most of the parameter passing and computational features of ordinary procedure calls; moreover, |PEXPRs extend the usual idea of procedure by allowing the existence in one structure of many different "executable patterns" to handle many different messages.

At any rate, now that the matching pattern (EXCHANGE TICKET-PERSON BUYER TICKETS MONEY) has been discovered, it is asserted in TICKET-PERSON1. This means that should TICKET-PERSON1 be requested to respond to the same message sometime in the future, it need only retrieve the answer, already pre-computed. Once asserted, the pattern is returned as result to BUY1 which then continues its processing. TICKET-PERSON1 stays around so that it can be perused later, perhaps as part of an investigation into previous episodes. In fact, the only way frames such as TICKET-PERSON1 can be removed is at the discretion of an "intelligent" garbage collector.

If in the example, there had been no match in TICKET-PERSON, the search would have continued breadth-first up ISA links from TICKET-PERSON to SELLER, and so on, until a successful match had been found or until there were no more ISAs. In the latter case, FAIL would have been returned to

BUY1.

This ability to carry on in spite of apparent failure greatly enhances the system's robustness and power. The following examples illustrate some of the many different ways of recovering from a failure to match. If, for instance, TICKET-PERSON1 were sent the message (SELL TICKET-PERSON TICKETS-TO-CONCERT), no match would be discovered using the type of ISA search described above, but SELL would know enough to suggest instead that a "relaxed" ISA search be carried out. Here, the requirements for a match are relaxed so that the message pattern need only be <u>covered</u> by a frame pattern rather than matched exactly (a term from MERLIN (Moore and Newell (1973)) - a pattern is said to be "covered" by another pattern if each element x of the first pattern is the same as the corresponding element y of the second, or is a subset of y). By this definition, (SELL TICKET-PERSON TICKETS-TO-CONCERT) is covered by (SELL TICKET-PERSON TICKETS); hence a match is discovered.

If the model failed to match (LOCATION SELF ?WHERE), the search would probably be directed into the execution environment of the receiving frame, since the physical location of the model is more likely to be found in the current context than in the ISA environment. Execution environment searches are commonly entered into when looking for contextually variable things such as time, location, purpose, etc.

A different kind of processing could be undertaken if the message pattern were

(EX-ENVIRON TICKET-PERSON1 ATTEND-CONCERT1). To find a match for this, EX-ENVIRON suggests looking for a sequence of patterns

(EX-ENVIRON TICKET-PERSON1 ?X1) in TICKET-PERSON1,
(EX-ENVIRON X1 ?X2) in X1,
.
.
.
(EX-ENVIRON Xn ATTEND-CONCERT1) in Xn.

That is, ATTEND-CONCERT1 is in the execution environment of TICKET-PERSON1 if there is a connected chain of EX-ENVIRON pointers from TICKET-PERSON1 to ATTEND-CONCERT1. Still other kinds of unmatched information might require even more complex inferences, but I won't go into these here.

|PEXPRs, thus, are structures that combine both declarative and procedural information in a single representation with a uniform access method. Whether there is too much flexibility for easy understandibility is a major question with no ready answer. Suffice it to say that |PEXPRs don't seem any less structured than production systems (Newell (1973)), and these have proven to be useful in many areas of AI. Another major problem is that of efficiency of implementation, something that hasn't overly concerned me as long as the methods aren't inherently explosive. I trust that if |PEXPRs turn out to be useful computational structures, efficient software and/or hardware can be developed to handle them.

Conclusion:

I have not had the time to go into the many non-|PEXPR features of the representation, nor have I really been able to explain in any detail how the representation can be used for

modelling conversation. These aspects will be dealt with in my forthcoming Ph.D. thesis, but for now this brief introduction will have to do. Hopefully, some of the flavour of the model's structures has been communicated.

References:

Bobrow, D. G. and Wegbreit, B. (1973) A Model and Stack Implementation of Multiple Environments. CACM,16,10 (Oct.), pp. 591-603.

Charniak, E. (1975) Organization and Inference in a Frame-like System of Common Sense Knowledge. Proc. Conf. on Theoretical Issues in Natural Language Processing, Cambridge, Mass., pp. 42-51.

Funt, B. V. (1976) The Use of Analogues in Problem Solving. Appearing elsewhere in this volume.

Greif, I. and Hewitt, C. (1975) Actor Semantics of PLANNER-73. Proc. SIGPLAN - SIGART Conference, pp. 67-77.

Kuipers, B. J. (1975) A Frame for Frames: Representing Knowledge for Recognition. Representation and Understanding, Bobrow, D. G. and Collins, A. (eds.), Academic Press, New York, pp. 151-184.

Minsky, M. (1974) A Framework for Representing Knowledge. AI-Memo 306, MIT AI Lab., Cambridge, Mass.

Moore, J. and Newell, A. (1973) How Can MERLIN Understand? Knowledge and Cognition, Gregg, L. (ed.), Lawrence Erlbaum Associates, Baltimore, Md., pp. 201-252.

Newell, A. (1973) Production Systems: Models of Control Structures. Visual Information Processing, Chase, W. G. (ed.), Academic Press, New York, pp. 463-526.

Sussman, G. J. and McDermott, D. (1972) From PLANNER to CONNIVER: A Genetic Approach. Proc. FJCC 42, pp. 1171-1179.

Winograd, T. (1975) Frame Representations and the Declarative-Procedural Controversy. Representation and Understanding, Bobrow, D. G. and Collins, A. (eds.), Academic Press, New York, pp. 185-210.

Sources of Information in Discourse Analysis

Jim Davidson

Department of Computer Science
University of British Columbia
Vancouver, British Columbia

Abstract

A large amount of useful information for natural language analysis exists outside the standard fields of syntax, semantics, and pragmatics. Areas which can be identified include: style, cohesion, staging, text structure, context, and real-world knowledge. This paper illustrates some of the potential of these, by indicating how they might be used for one specific task -- the analysis of narrative discourse. The various types of information are reviewed, and their cooperation and interaction, using a method of prediction, described. Finally, their role in general natural language work is indicated.

1. Introduction

Natural language analysis in recent years has been performed through the 'microworlds' approach: a specific domain, with inherent linguistic formations, is defined, and information outside the boundaries of the domain is skipped. Typically, the areas covered are syntax, semantics, and pragmatics. The approach seems valid; however, this paper represents an effort to call attention to some features of language which are not handled in most microworlds.

To see these features, a larger domain than sentences must be looked at. Narrative discourse (i.e. very short stories) offers itself as an effective test.

Discourse can be typed as narrative, hortatory, procedural, expositional, and dramatic, among others (Longacre, 1970). I have chosen the first of these to work with, as being the least artificial.

The intent here is to present a list of 'information-sources' which can be used in analysis of discourse. Some of these are handled in current systems, some are available but are ignored, and many are only seen at the discourse level.

2. Information Sources

A number of possible linguistic features can be identified, including: cohesion, staging, text structure, context, and real-world knowledge. These are all in some sense 'available', in that they contain gratuitous information which can expedite the analysis process.

Given these possibilities, the question becomes one of finding an effective way to use the information sources. Essentially, three problems arise:

1. each of these must be recognized -- i.e. the surface form they take in a discourse described;
2. the information they carry must be formalized -- converted into a form usable by a computer program;
3. the interaction of the various pieces must be decided: resolution of conflicts, etc., must be specified

About the last, little will be said, except to mention that interaction is a more serious problem than it may seem. Often two sources might be contradictory, or more importantly, might work in unison to provide more power than either one alone. (For example, consider 'syntax' and 'semantics' as possible sources; these two obviously work better together than

separately.)

To attack number (2), we require a uniform representation, in which to encode the linguistic information. An effective tool here would seem to be Riesbeck's (1974) predictions.

Riesbeck's predictions consist of a test (i.e. a demon looking for a specific construct in the input), and an action (specifying what to do when that construct is found). Predictons are spawned by various lexical items, and may in turn spawn other predictions.

There are several for choosing the prediction method; perhaps the strongest is the modularity it provides. When a new information source is identified, it can be incorporated quite easily, to contribute its own predictions to the system.

An interesting question, here as in Riesbeck's thesis, concerns the level of the predictions. It would be nice to tell the system to look for a specific word, but the possibilities of that degree of precision seem slim; rather, a more likely level is that of the concept: 'expect a sentence with John as agent','expect a locative prepositional phrase', etc.).

Two kinds of information are available from the sources mentioned. The first is an indication of what to look for next. This is the most common form in natural language work, and obviously can be represented easily in the prediction system.

The other type is information about how to organize input already seen. An obvious requirement of an analysis program is that is be able to structure its results intelligently; this desideratum does not become evident until the discourse level.

In general, this sort of information can be incorporated into a prediction system, but it depends quite heavily on the underlying representation of language.

The remainder of the paper is devoted to problem (1). I will review the various information sources mentioned, in each case pointing out the most prominent features of the class mentioned (space precludes a more detailed analysis) and indicating how the information could be fitted into the prediction paradigm.

staging:

This is described (Grimes, 1975) as the manner in which the speaker organizes the information for the hearer's benefit. The most important aspect of this is the theme: the 'point of departure' for the speaker, with respect to which he organizes the discourse.

More specifically, theme is the (abstract) concept identified; topic is the surface form used to signal it.

Theme occurs at the sentence, paragraph, and discourse levels. The only level at which it can be easily characterized is the sentence; I shall deal with that one here.

In surface structure, the theme can be indicated by any of the following: fronting (rearrangement of word order to put the desired element at the beginning; e.g. passive voice), extraposition, and embedding (and in some languages, inflection). These can be recognized quite easily; hence, fairly straightforward rules for detecting the theme of a

sentence can be established.

Theme can be useful, in that it indicates what the sentence is, in some sense, about. The fact can aid in a number of areas, from pronoun resolution to summarization. Essentially, it provides a perspective, through which changes in subject can be detected.

cohesion:

Halliday (1976) identifies this as the manner in which incoming information relates to the information previously given. Two aspects of this are relevant: the formation of information blocks, and the degree of specification of anaphora.

Information blocks are chunks of information, which are thrust upon the hearer one at a time. These reflect the given/new distinction, in that they indicate the amount of new information which the hearer is to absorb (this has been described as the rate of 'information injection'). Information blocks are signalled in speaking, through intonation; in writing, punctuation (especially commas) is the usual medium.

The discovery of an information block can be valuable in natural language analysis, since its length is often a cue to its importance: longer blocks generally indicate a lower rate of information injection, hence less importance. Thus, an analysis program can decide what to do with a unit it has just discovered, on the basis of its estimate of the salience of that unit to the story (much as humans do; de villiers (1974) discovered that subjects remembered sentences depending on their centrality to the story being constructed).

The specification of anaphoric reference is a matter often overlooked by programs which merely establish the (extensional) identity of the referent. The point is that the referring phrase often contains clues as to the speaker's estimation of the relevance (to the hearer) of the referent. The various forms of referring (pronouns, demonstratives, inclusive nouns (e.g. thing, one), relative clauses) are specific to a greater or lesser degree. A more specific reference indicates that the speaker felt a greater need to 'point out' the referent -- i.e. felt that it was less than obvious to the hearer. Again, a program might use this to decide the salience of a particular piece of information, and hence what to do with it.

text structure:

A number of writers (Rumelhart (1975), van Dijk (1972), Propp (1968)) have described high-level text grammars, which characterize the 'structure' of a narrative discourse. A sample rule might be:

story --> setting + episode

Unfortunately, there still remains a gap between the high-level description, and the low-level instance. Work is needed to close this, by establishing the surface form of each of the constructs.

One example should help: the prediction for 'setting' might be, at a general level: 'look for some "grounding" information in which to anchor the story'. At a lower level, this might split into 'look for a temporal specification', 'look for a spatial specification', 'look for some new characters', 'look

for a continuing action', where these could be characterized in some reasonable computational manner.

As mentioned, there is much to be done; this remains one of the more promising fields in natural language.

Context:

Context is a somewhat-nebulous term, used to describe the 'surroundings' of the current input. Most of its benefits are manifest in the theme (see cohesion), but two others are relevant here.

First, a primitive notion of foregrounding (Chafe,1970) can be established. That is, if a concept is mentioned once, it is reasonable to expect that it will be referred to again. For the purposes of prediction, this can be encoded as a simple recency test.

The other potential contribution lies in the notion of 'related concepts'. This is very close to scripts, or 'frames' (Minsky,1975), which essentially provide certain default information, and allow unambiguous reference to unmentioned concepts. For example, after seeing the sentence 'John went into a restaurant', the phrase 'the waiter' would be understood without difficulty. This is a currently-popular A.I. approach, and little description is needed.

Context is similar to 'real-world knowledge', discussed next.

Real-world knowledge:

Perhaps the best example of use of this is Riegel's (1974)

inferencing program, which makes (semantic) deductions based on the input. With a little effort, this can be characterized in a manner which permits its use in analysis.

In analyzing narrative discourse, the events or the story are usually connected either by temporal sequencing, or by cause/effect relationships. The former are not easily characterized or predicted, but latter can be handled in a fairly straightforward manner with Rieger's inferences. Essentially, we can use rougnly four of his sixteen inferences (specification, function, cause, and result) to predict from one event to another.

This approach tends to be explosive, but, if restricted, should prove valuable in generating predictions.

others:

A number of other fields might have been mentioned, but weren't.

Collateral (Grimes, 1975) is the use one statement to emphasize another (e.g., rhetorical questions). This could prove valuable for possibilities or summary; the emphasized construct will be seen as more salient.

Diction, or lexical selection, is the theory of the reasons for using specific words. Perhaps the best example of this is the use of synonyms, as opposed to repetition, to avoid monotony in the discourse. This is actually part of what is traditionally called 'rhetoric', and is more common in persuasive than narrative discourse.

Presupposition is another potentially rich area of benefit

in natural language. True, it is hard to characterize, but a bad implementation of presupposition is better than none at all.

Of the five sources discussed in detail, only one (story structure) is specific to narrative discourse. Certainly more domain-dependent information could be extracted. In particular, we might look for a way to effectively characterize the strong time-orientation found in most stories: the temporal ordering provide the central thread of the story, upon which identification and explanation are hung as peripheral elements.

3. Conclusion

This paper has been an effort to provide indications of possible sources of information useful in natural language analysis.

Note that the five mentioned are all in addition to the normal syntax and semantics components. Essentially, they can be viewed as gratuitous information, available in any extended discourse.

My contention is that these, and other sources, are quite basic to the understanding process: an effective program *must* be aware of these and make use of them. The first two, at least, haven't been used in any serious computational linguistics work.

Various methods exist for implementing the features here; one, based on predictions, has been mentioned. What is needed now is more work to flesh out the ideas, and tie everything together.

4. Bibliography

Chafe, W. *Meaning and the Structure of Language*, U. of Chicago Press, Chicago, 1970

van Dijk, T. *Some Aspects of Text Grammars*, Mouton, the Hague, 1972

Grimes, J.E. Kinds of information in discourse, *Alvdbg*, 4 (1971), 64-74

Grimes, J.E. *The Thread of Discourse*, Mouton, the Hague, 1975

Halliday, M.A.K., and Hasan, R. *Cohesion in English*, Longman, London, 1976

Longacre, R. Sentence structure as a statement calculus, *Language*, 46 (1970), 783-815

Propp, V. *Morphology of the Folktale*, U. of Texas Press, Austin, 1968

Rieger, C. Conceptual Memory and Inference, Ph.D. Thesis, Computer Science Dept., Stanford University, 1974

Riesbeck, C. Conceptual Analysis, Ph. D. Thesis, Computer Science Dept., Stanford University, 1974

Rumelart, D. Notes on a Schema for Stories, in *Representation and Understanding*, D. Bobrow and A. Collins (ed.), Academic Press, New York, 1975

Schank, R. C. And Abelson, R. *Scripts, Plans, and Knowledge*, Proc. 4th Intern. Joint Conference on Artificial Intelligence, 1975, 151-157

de Villiers, P.A. Imagery and theme in recall of connected discourse, *J. Exp. Psych.*, 103 (1974), 263-268

Preliminaries for a Computer Model of Conversation

Philip R. Cohen and C. Raymond Perrault
Department of Computer Science
University of Toronto

Abstract

We are building a program which conducts a dialogue with a user, helping him to perform a task. The need for a model of the user's plans and beliefs in such a program is stressed. We outline how speech acts can be represented as operators in a planning system, and how such a user model can be organized. We then discuss the role of plan generation and recognition in language understanding.

Introduction

We are interested in constructing a natural language understanding system whose purpose is to assist its user in performing some task. The system is to engage the user in a conversation as opposed to simply answering questions. Explorations of this domain have already begun in SRI's Computer Consultant program (Nilsson [1975]) and Travel Budget Assistant programs at BBN (Bruce [1975]) and Xerox PARC (Bobrow et al. [1974]). This type of system clearly has potential practical utility, but is of interest to us because it presents some very interesting problems in language analysis and generation, as well as enabling us to work on modelling the conversational process.

Consider the following exchanges with such a system:

(1) User. I want to fly to L.A. tomorrow.
System. The air controllers are on strike.
Why don't you take the train?

(2) User. Where are the bolts?
System. How many do you want?

In (1), the system may have recognized that any attempt to schedule a plane trip will fail. The system replies by indicating this fact rather than by simply responding "OK" to the declarative utterance. Notice that the system then suggests an alternative. In (2), assuming the machine is in charge of dispensing bolts, an answer "I have them" is of little use to the user if it is likely that he wants the bolts themselves rather than just knowledge of their location. The system has noticed an

obstacle that the user would likely encounter following an obvious or literal reply to his question.

Thus the speaker, in order to generate these more helpful replies, makes use of information about the plans of the hearer. Also, speakers assume certain beliefs are held by their conversants. For example, the user in the second example believes that the system knows where the bolts are. If this belief were incorrect, the system could reply "I don't know" which would also not be a literal reply, but rather a denial of the user's belief.

We view conversation as a sequence of actions performed as part of plans by speaker and hearer. These linguistic actions have effects on the beliefs and goals of speaker and hearer, and can only be felicitously performed if certain conditions hold. We claim that these actions can be incorporated into plans with non-linguistic actions, such as dispensing bolts and scheduling trips, and thus we see language analysis and generation as being very closely related to problems of plan generation, execution, and recognition.

Austin [1962] pointed out that the meaning of every utterance should include not just the truth value of the proposition(s) it might contain, but also the act performed by the speaker of the utterance. The same sentence could be used to perform several different acts. For example, a user saying (3)

(3) "The cost of the part is $25"

to a data base management system could be asserting new information, correcting old, or clarifying an earlier part of the dialogue, while the proposition expressed would be the same under all three readings. Other examples of what have been called speech acts are request and promise. We will sketch in this paper how this notion of speech acts can be used in language understanding and generation.

The program we are developing will be operating in a domain where the user is presumed to be executing some plan (or script), runs into difficulty and asks the machine for help. Dialogues in such situations are purposeful and task-oriented. The program will be given a semantic network representation of an utterance. It should then identify the utterance as some speech act and generate a speech act (perhaps more than one) based on its explicit motivation of being helpful. Thus, we will not be dealing, at this stage, with the parsing and generation of surface utterances.

Our possible domains of discourse will include helping someone to perform some task such as baking a cake (cf. Scragg's KITCHENWORLD (Scragg [1975]) and SRI's computer consultant (Nilsson [1975])), and an intelligent assistant program (such as an intelligent data base system handling graduate student files).

Our system will employ an explicit model of its user and use it to be more helpful. The situations our machine would be placed in are such that it would have a default model of its user before even engaging in conversation.

Structure of Speech Acts

Searle [1969] gives examples of necessary (and he hopes sufficient) conditions for the successful performance of a speech act. We have modified and reformulated his list for purposes of a process model of language use (see also Schmidt [1975]). Below we state preconditions for a speaker S uttering a request that a hearer H should do action A. These conditions are formulated from the speaker's point of view, i.e. as the system would use them for generation. Our system will also use them for identification, where it would be playing the role of hearer.

For a request, S believes that:

1. H is able to do A. (I can't reasonably request you to walk through a wall).

2. H believes H is able to do A. (If I believe you believe you can't walk, I would not ordinarily request you to do so.)

3. It is not obvious that H would do A in the normal course of events.

4. S wants H to do A.

5. Requesting H to do A is sufficient reason to cause H to decide to want to do A.

6. Normal input/output conditions obtain (i.e. both S and H speak the same language, are proximate, etc.)

The goal of the request is not that H should want to do A, but rather that H should believe that S has requested him to do A. That is, it is possible for H to refuse to perform A without invalidating S's utterance as a request. We assume that a separate step is necessary for H to decide to do A, which may depend on authority, friendship, etc. S's goal in uttering a request, then, is that S believe

i). H believes S wants H to do A.

There are at least two secondary effects of the request: that H should believe preconditions 1 and 2. These become -- S believes that

ii) H believes S believes H is able to do A.

iii) H believes S believes H believes H is able to do A.

The point of rewriting Searle's conditions is that now they resemble acts that a planning system, such as NOAH (Sacerdoti [1975]) could work on. However, the testing of the preconditions, may require a recursive application of the PLAN algorithm as well as a more sophisticated mechanism for dealing with effects. The discussion of our memory model deals with these problems.

Condition 1 requires that S have a model of H's planning operators, as opposed to a model of H's view of his own operators, as required by 2. Testing 1 or 2 then involves

verifying whether a plan to accomplish A can be constructed using the appropriate operators. To test condition 3, S can see, according to what S knows about H's view of H's plans, whether A would be performed without S taking any action.

Condition 4 would not have to be included as a precondition since it is the goal of the request and the reason the other preconditions are being tested at all. Condition 5 is a statement that once H believes S wants H to do A, then H must decide to do A before H will want to do A. Condition 6, for our purposes, concerns only proximity and will be ignored.

Thus, the definition of REQUEST, which the planner employs, would include 1, 2, and 3 as preconditions and i), ii), and iii) as effects. Effect i) would be flagged as the goal of the act.

Using Speech Acts

We will demonstrate our methodology by examining how we will deal with the problems of the generation and identification of speech acts. First, we outline the structure of the memory model necessary to segment beliefs and wants.

The Memory Model

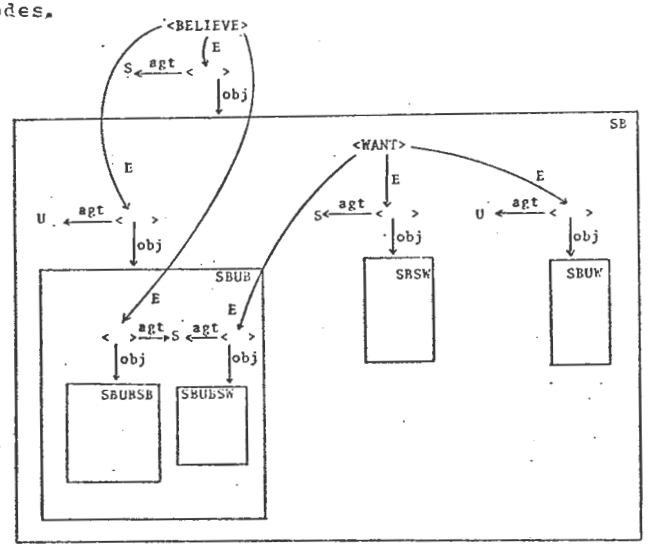
The preconditions and effects of speech acts are formulated in terms of the beliefs and wants of the participants, which may include beliefs about beliefs and wants. Our system maintains these embedded belief models (what it believes its user believes, etc.) by using a partitioned semantic network (Hendrix [1975]). We assume the reader is familiar with semantic nets, though not necessarily of the partitioned variety. As this tool is so important we will explain it briefly. For a more detailed presentation, the interested reader should refer to Hendrix [1975] and to Walker [1975].

Semantic networks have been notoriously homogeneous; so much so that researchers have had difficulty representing scoping of quantifiers, hypothetical worlds, theorems, etc. These

difficulties are primarily due to the inability to group nodes and edges in order to refer to the group as a whole. Hendrix uses a grouping concept called partitioning. He represents quantification by allowing a group (called a space) to be the scope of a quantifier. Spaces can overlap and thus nodes and edges can be in more than one space. In addition, Hendrix also proposes a CONNIVER-like visibility mechanism implemented by adding a partial ordering on the spaces of the network, called the "visibility lattice". A traversal of the net starting in a space on the lattice will be able to see data which are in spaces further up on the lattice.

We identify a space as representing the "belief" or "want", of an agent S by placing it as the value of a case in a proposition stating "S believes" or "S wants". This indicates that the semantics of each space are dependent upon the interconnections of the space with other network entities, rather than simply upon the entities within the space.

The space "System Believes" (SB) would include the spaces "System Believes User Believes" (SBUB), "System Believes User Wants" (SBUW), and "System Believes System Wants" (SBSW), as nodes (see below). WANT spaces can also contain WANT and BELIEVE spaces as nodes.



There is a question as to how many levels of belief-space nesting are required in such a system. So far we have found four or five levels to be sufficient but there are claims (Strawson [1964], Schiffer [1972]) which indicate that theoretically an infinite number may be necessary, but all but a finite number are identical.

Precondition 1 of request is that S believes H believes H is able to do A. Now, how can S check this condition? At least part of this decision must include a determination of whether H can PLAN to do A. Thus S must PLAN for H's doing of A. This can be accomplished by confining the PLAN program to SBUB, which effectively simulates U's planning (to the best of S's knowledge). This use of the nested memory, to check embedded conditions and to provide a searching and processing environment is typical of our system's uses of shared knowledge.

Generation of speech acts:

We argued that conversational systems should consider the effects that literal responses to the user would have on its model of the user. If the system believes that the user is likely to encounter difficulties, a more "helpful" response should be generated to try to overcome these difficulties.

Our system will have reasons for its utterances; they will fit into its plans and achieve effects that enable other acts to take place. The generation procedure might work as follows:

For any goal identified as the user's obstacle: PLAN to achieve the obstacle (e.g. say something in order to satisfy the user's goal of knowing something.) If the plan is successful (and the obstacle can be removed) ensure that the rest of the plan is clear for succeeding goals (check prerequisites of acts). If the plan is unsuccessful, or the path is not clear, find an alternate path from the user's current state to any other higher-level goal in the plan which bypasses the

obstacle and could lead to the result (e.g. find a functional alternative). Attempt to get the user to achieve this goal (i.e. issue a DIRECTIVE -- perhaps SUGGEST).

Notice how this procedure might generate the system's responses in example 1 by checking preconditions of acts in the plan that have been inferred for the user. A statement of the obstacle in the plan and a suggestion of an alternative could thus be generated.

Thus, the system's plans include acts which call for making the user aware of certain beliefs and for getting the user to perform acts.

Identification of Speech Acts

In this section, we will outline how our system would relate an utterance to what it knows of the speaker's plans.

We assume that when the system receives an utterance, preliminary syntactic/semantic processing will suggest alternative speech acts that could describe the utterance, as in example 3. The system then sees if any of the putative speech act identifications of the utterance fit into a plan for the speaker. The first step to be taken is one of ruling out some identifications as impossible based on precondition failure. The preconditions of speech acts are stated from the speaker's point of view and thus the system must search in its model of the speaker for the truth of these conditions.

We treat the problem of identifying a speech act as one of inferring a speaker's plans such that the potential speech act SA is part of the inferred plan. This may be done by planning in the belief space corresponding to the system's model of the speaker's beliefs (SBUB). If we already have a plan for the speaker, we must determine how and where the SA fits into this plan. This is where we are concentrating our efforts in the current research.

This inference process may account for the identification of what Searle [1975] and Ervin-Tripp [1976] have called indirect speech acts.

Progress and Implementation

The thrust of our approach to developing a machine that can engage in purposeful conversation is to integrate linguistic and non-linguistic behavior by viewing speech acts as acts in a plan. In order to do this, the system will maintain an explicit model of its user. (Actually, we will be planning with acts based on Searle's [1975] taxonomy of speech acts, e.g. DIRECTIVE, rather than with the speech acts themselves.)

We are currently in the process of implementing these ideas for a program which could help someone to cook. Our representation of knowledge will be based on a semantic network system similar to the one outlined in Levesque et al. [1976]. We are implementing Sacerdoti's NOAH algorithm to use our nets. We can avoid using his SOUP code by encoding our procedural knowledge directly in networks. Thus, we have implemented a MEMOD (Norman and Rumelhart [1975]) network executor that can execute network programs (script, plans). The act HELP, for instance, will use the primitive act PLAN in its definition. PLAN, however, can operate in any of a number of belief spaces (e.g. SBUE). The partitioned semantic net package has been implemented and is already in use. Its extension to the belief model is straightforward. The system is being developed in the SPITPLUS version of SPITBOL running under TSO on an IBM 370/165.

Acknowledgements

We would like to thank James Allen, Robin Cohen, Mary Morrigan-Tozer, and Corot Reason for their ideas and assistance in the development of the system to date. Prof. John Mylopoulos and Hector Levesque have also been instrumental in developing our philosophy of implementation.

References

- Austin, J. L., How to Do Things with Words, J. O. Urmson (ed.), Oxford University Press, 1962.
- Bobrow, D., Kay, M., Kaplan, R., and T. Winograd, "Steps towards Language Understanding", Xerox PARC, 1974.
- Bruce, B., "Pragmatics in Speech Understanding", IJCAI4, Tbilisi, 1975.
- Ervin-Tripp, S., "Is Sybil there? -- The Structure of American English Directives", Language in Society, vol. 5, no. 1, April, 1976.
- Hendrix, Gary, "Expanding the Utility of Semantic Networks through Partitioning", Proceedings of IJCAI4, Tbilisi, 1975, pp. 115-121.
- Levesque, H., Mylopoulos, J., McCalla, G., Melli, L., and J. Tsotsos, "A Formalism for Modelling", Proceedings of the CSCSI/SCEIO National Conference, Vancouver, Aug., 1976.
- Nilsson, Nils J., "Artificial Intelligence -- Research and Applications", Stanford Research Institute AI Center Progress Report, Menlo Park, Ca., 1975.
- Norman, D., and D. Rumelhart (eds.), Explorations in Cognition W. H. Freeman and Co., 1975.
- Sacerdoti, Earl, "The Nonlinear Nature of Plans", Artificial Intelligence Group, Technical Note 101, Stanford Research Institute, Menlo Park, Calif., January, 1975.
- Schiffer, Stephen, Meaning, Oxford University Press, 1972.
- Scragg, G., "Answering Questions about Processes", in Explorations in Cognition, Norman and Rumelhart (eds.), W. H. Freeman and Co., 1975.
- Searle, J. R., Speech Acts, Cambridge Univ. Press, 1969.
- Searle, J. R., "Indirect Speech Acts", Syntax and Semantics Volume 3: Speech Acts, Cole and Morgan (eds.), Academic Press, 1975.
- Searle, J. R., "A Taxonomy of Illocutionary Acts", Language, Mind and Knowledge, K. Gunderson (ed.), U. of Minnesota Press, 1976.
- Schmidt, C. F. "Understanding Human Action", Proc. of Conference on Theoretical Issues in Natural Language Processing, Cambridge, 1975
- Strawson, P. F., "Intention and Convention in Speech Acts", in Phil. Review 73, pp. 439-460, 1964. reprinted in The Philosophy of Language, J. R. Searle ed., Oxford Univ. Press, 1971.
- Walker D., Paxton, W. H., Robinson, J. J., Hendrix, G. G., Deutsch, B. G., and A. E. Robinson, "Speech Understanding Research", Annual Tech. Rept., SRI-AI, March, 1975.

TOWARDS A SEMANTICS FOR A SCIENTIFIC KNOWLEDGE BASE

Douglas Skuce

Montreal Neurological Institute, McGill University
3801 University Street, Montreal, Quebec

Abstract

The problem of taking scientific knowledge, both from text and from experts, and precisely expressing it to match the abilities of a computer system is discussed. An English-like surface language, LESK, is proposed to aid the computer-naive expert clarify, for herself, others, and the machine, exactly what is to be stated. This language is intended to map relatively easily onto a quantified semantic net machine representation, under development. Clarity of semantic concepts is stressed. Examples of the main problems, particularly involving process descriptions, will be taken from the domain of neurophysiology.

Introduction

It is my feeling that the design of a scientific knowledge base is a worthwhile yet relatively unexplored area of AI research. In particular, medical and biological science could greatly benefit from some increased precision. Unfortunately, the bulk of this knowledge is expressed at present in natural language (NL), a medium poorly suited for the requirements of precision, brevity and machine compatibility. My main objective then is to develop a medium of communication sufficiently like NL to be understandable to biomedical personnel, yet which also possesses a clear semantic foundation suitable for machine "understanding" as well. Mathematically inclined persons lacking experience in an actual biomedical research environment often fail to appreciate the importance of the former requirement. An additional requirement is that all statements be unambiguous.

LESK (Language for Expressing Scientific Knowledge) is my developing approximation to such a medium. The emphasis is placed, at present, on discovering and explicating the necessary underlying semantic constructs. This is being done by considering two real examples, both micro-domains of neurology: the synapse (the communicating contact between nerve cells) and the spinal reflex arc. Interaction with live neuroscientists has been stressed.

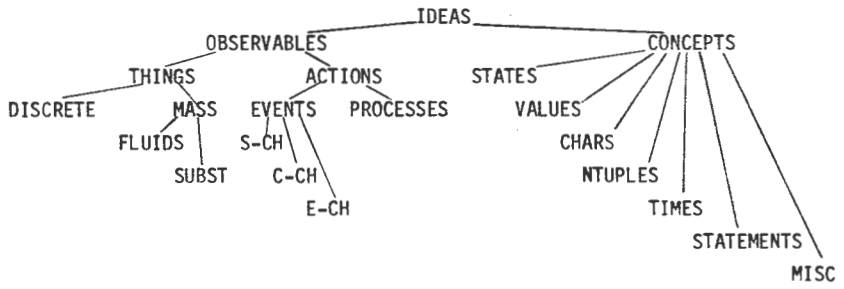
In the first case, a composite description of the essential facts about synapses was solicited from seven scientists. I then am undertaking to express these facts in LESK. For the reflex arc, one scientist was briefly familiarized with the basic concepts of LESK, and she and I have jointly developed a

LESK description, I learning neurology while she learned the concepts of LESK. If these and the synaptic facts had been encoded in some less transparent formalism, verification of them with the expert(s) would be impossible.

This paper will discuss first the underlying principles I have adopted for structuring the semantics of technical knowledge, and then will illustrate these with a number of examples from the two neurological domains. It is a continuation of the work first reported in Skuce (1975).

Clarifying the Basic Semantic Terminology

Underlying any discussion of semantics there ought to be found a collection of basic concepts, which are explicitly denoted as such, and clearly explained. Hence I begin the description of LESK semantics by attempting to make more precise a number of common AI buzz-words which appear frequently, though nearly always without adequate clarification of their meanings. I can only list here some of the main offenders, beginning with the semantic category hierarchy:



This tree, which is not complete, specifies the conceptual terminology that is to be adhered to in stating scientific knowledge. Thus a LESK statement will contain terms chosen by the user to suit the vocabulary of the field, but which are stated to belong to (usually) one of these categories, which have certain very high level rules associated with them called semantic axioms. These axioms have to be inducted from the many examples explored, and constitute the basic "laws of nature" that these terms obey, i.e. that would be known to a machine about these categories. Some semantic axioms are:

ONLY GROUPS, THINGS AND PROCESSES HAVE STATES
EVERY EVENT HAS A TIME INSTANT
EACH CHARACTERISTIC HAS A VALUE
EVERY ACTION HAS A CAUSE

EVERY PROCESS HAS A TERMINATING CONDITION
A THING CAN USE A THING TO CONTROL A PROCESS
ONLY GROUPS, THINGS AND PROCESSES CAN EXIST

Thus each semantic axiom is an abstraction, to the highest level possible, of at least two facts about the world that it has been found necessary to state. These statements are in LESK form; note also the frequent occurrence of the notion of "having". Hundreds of these will be required for even a small domain, though many are common to all domains. The only other system I know of which attempted this to any extent was that of Sandewall (1972).

Among the list of troublesome but common AI semantic terms there are some which may be called "metaconcepts" and which need clarification just as much. On this list I include: class, instance, individual, frame, slot, cause, instantiate, set, and a few more. These also I have attempted to define and use consistently.

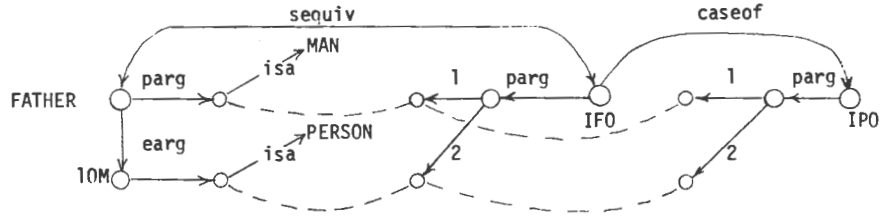
Gathering Knowledge Into Clumps: Frames and Relators

The frame has recently become the common paradigm for discussing the notion of "clumping" knowledge into closely related packets. I have been developing a version of this notion which I call the relator, which is in many ways similar to the unit of Bobrow and Winograd's language KRL (Bobrow and Winograd, 1976). Both units and relators have the SIMULA class as an ancestor. A relator may be thought of as a collection of closely related statements, linked together by some bound variables, which define some conditions on a collection of possible instances of some phrase, which might denote a THING (i.e. a noun), or an ACTION or RELATION (a verb), or some other class of possible instances. When all the variables, called slots, have been suitably replaced by instances of them, we say that we have an instance of the class represented by the relator. This is the basic frame idea, and the KRL unit does the same. LESK stresses the importance of defining (canned) phrases, rather than individual words. Relators, however, distinguish several kinds of slots, of which the most important are the primary and the existential (or Skolem) slots. For example, to define the two related phrases: X IS A FATHER and X IS FATHER OF Y, we would say:

```
SIMDEF    X IS A FATHER                    RELDEF   X IS FATHER OF (IFO) Y
          X IS A MAN                                X IS A FATHER
          E 1OM PERSONS Y, X IFO Y                  X IS PARENT OF Y
END                                        END
```

SIMDEF says we are defining a SIMPLE relator, i.e. with only one primary slot (usually a noun); the phrase following is the one being defined. E means

"there exists" and 1OM means "one or more". Thus this relator has also a set of existential slots (Y1, Y2,...) which must exist for each instance of the primary slot X, and which must have the relation IFO to X. Similarly, IFO is defined as a RELATION, a second type of relator. It implies that two other relators (IS A FATHER and IS PARENT OF) are necessarily involved, and the correspondence between slots is indicated. Since each instance of each of these relators implies the existence of an instance of the other, we say that they are semantically equivalent (sequiv). We diagram these relators thus:



The dashed lines show the correspondence between slots; any conditions imposed on a slot in one relator apply to the corresponding slot in other relators, though not always bidirectionally. For example, parg1 of IFO must be a MAN, but in IPO it would be generalized to PERSON (had we defined IS PARENT OF). This is permissible since IFO is a case of IPO. This (case of) and other inter-relator relations, such as sequiv, converse (a kind of sequiv), implies (half of sequiv) and a few more form a set of high level relations which play a large role in deduction, and other "associative" operations. Briefly, when trying to establish some connection from one relator to another, these links are used to select those which are most likely to be relevant. This process can be performed largely in parallel, which is one of the underlying central motivations behind my view of what a relator should do. The large question of the parallel interaction of relators, during deduction and other actions, is beyond the scope of my present work, in which I am trying only to formulate general principles.

So far we have met two types of relator; there are three more. The TUPLE, like the mathematician's tuple, is a multi-slot relator used as a noun, in which a number of IDEAS are collected together under one name. The slots, one per IDEA, are called components. For example:

```
TUPDEF    A MARRIED COUPLE IS (1 MAN X, 1 WOMAN Y)
          X IS MARRIED TO Y
END
```

TUPLES are also called GROUPs.

It is important though to note a major distinction between the relator and the KRL unit: the latter, in defining a noun, has a slot for every property or attribute that X has, including non-characteristics (e.g. other THINGs that "belong to" X), similar to the existential slots. This makes it dissimilar to the LESK TUPLE, whose slots are only for the IDEAS arbitrarily brought together by definition, i.e. as a mental construct, rather than by necessity of experience. This question of "having", "belonging to", etc. is part of the general issue of quantification. LESK considers this to be as fundamental a representational structural problem as class hierarchies (KRL does not seem to), and quantification has hence been given equal prominence.

Quantification

It would seem as though, with the demise of predicate calculus as a "representational" language, people have forgotten what it was good for: representing quantification. I have found quantification to be an inescapable central aspect of the knowledge I have been considering; indeed about half the static statements (those which do not describe change) are of the form: EACH X VERB QUANTIF Y, with frequent use of second order classes (ones whose instances are themselves classes) and of the notion of unique belonging. I have structured the rules and primitives for denoting quantification under the unifying notion of functional (Skolem) correspondence, which I require the LESK user to understand. Thus LESK primitive phrases like HAS, ITS, or THE X OF Y are defined in terms of "Skolem" pathways which denote an unambiguous referent, which may be thought of as a node in a semantic network. Some examples will help:

EACH MUSL HAS A JOINT = MUSL.X--HAS--JOINT OF MUSL.X
EACH JOINT HAS ITS OWN SET OF FLEX'S = JOINT.X--HASU--(FLEX OF JOINT.X).Y
EVERY EXT OF A JOINT J IS A DIRECT ANTAG OF EACH FLEX OF J =
(EXT OF JOINT.X).Y--ISADIRANTOF--(FLEX OF JOINT.X).Z

These three examples are taken from the spinal reflex statements. In each, the statement preceding the = sign is the LESK statement developed with the neurophysiologist; the underlined words are primitives. Following the = sign is a linear expression denoting the understood quantification. For example, MUSL.X is the class of all muscles; MUSL.3 is one of them. JOINT OF MUSL.3 is its corresponding JOINT, belonging to the class JOINT OF MUSL.X, the class of all JOINTs which have A MUSL as owner. FLEX OF JOINT.X would not be a true expression about neurology, since JOINTs have SETs of FLEXs, e.g. (FLEX OF JOINT.4).X is the SET OF all the FLEXs of JOINT.4, and (FLEX OF JOINT.4).2 is

one of these. Thus each phrase denoting a class or one of its instances may be thought of as a unique internal node which "knows who it belongs to".

The most complicated statement we encountered in the spinal reflex was:
THE A. M. NRN'S OF ANY DIRECT ANTAG OF A MUSL X ARE THE SET OF ALL A. M. NRN'S WHICH INNERVATE ANY DIRECT ANTAG OF X
This illustrates a common need in such knowledge: to give a name to a set of objects which does not normally have a name, so that we can discuss it more easily. The resulting classes are:
(A.M.NRN OF (DIRANT OF MUSL.X).Y).Z--INNERV--(DIRANT OF MUSL.X).Y

Dynamic Relators: EVENTS and PROCESSES

The remaining two relator types describe dynamic knowledge, i.e. how something changes. (Note that the first three describe static knowledge.) Any change is termed an ACTION (no connotation of willful actors here; remember we are interested in describing natural phenomena).

The EVENT is the simplest type; it describes an instantaneous change, in the manner of a STRIPS operator, for example:

EVDEF X RELEASES Y (INTO SPACE Z)
BEF: X HAS Y, OR X KEEPS Y, OR IF X IS A CONTAINER FOR Y, X CONTAINS Y
AFT: SITE OF Y IS INSIDE OF Z

Here the parenthesized part may appear in any occurrence of this phrase, if not it is implied, and Z becomes an existential slot. Any of the BEFore statements become untrue after the EVENT, and vice versa for the AFTer ones. This definition necessarily entails referring to spatial concepts, such as SITE, which is a SET OF POINTs which may be possessed by more than one "owner"; SPACE, a kind of SITE, having three dimensional properties, and IS INSIDE OF, a RELATION between SPACEs. This important subject alone would fill numerous volumes, and I have had to give it only cursory treatment. (Hint for Ph.D. thesis).

The PROCESS is the relator type which deals with any real variable, in particular, TIME. Any phenomenon which requires a real variable conceptually, whether any numerical properties are stated or not, is defined by a PROCESS relator, which is usually "owned" by some THING. This is the biggest difficulty in describing real world phenomena: how to deal with continuous change. One of the few discussions of this problem in the AI literature is Hendrix's (1973), although there is beginning to appear some interest in the synthesis of AI techniques with those found in continuous simulation systems (e.g. Brown's SOPHIE system (Brown, 1974), and Sussman's EL (Sussman, 1975). It is

this area that I am currently (summer 1976) focusing my attention upon, for it is essential to the description of physiological knowledge. (B. Smith is working on the description of physiological knowledge in KRL (Smith, 1976)).

Describing Neurophysiological Processes

Underlying any kind of process description, I assume that ultimately, if we knew enough, we would be able to write equations, as physicists and engineers do, describing the continuous time behaviour of all real variables. But we almost never know enough to even come close to being able to do this; what's more, even if we did, we usually wouldn't want to, because we would become lost in the morass of detail. So why do I mention it? Because I find it provides a unifying perspective on this difficult problem, and because there are some areas where enough is known to reach a "simulation" level, when the need arises. What we must be able to do then is to descend the "detail hierarchy" gracefully, starting with high-level declarative process descriptions, proceeding next to perhaps abstract procedural descriptions (still without any real functions), and descending finally to a "total simulation", where all real variables are specified as a function of time.

In the spinal reflex example, an attempt was made to actually write the "simulation level" equations, since a number of people have attempted to do this already. For example, there are four entities which may be said to possess a "firing frequency": neurons, sets of neurons, muscle fibres, and sets of muscle fibres. To the left of the vertical line below is the high level definition of the term FIRING FREQUENCY; to the right, its amplification for neurons (NRNs) and muscle fibres:

FIRING FREQUENCY (FF) F	FF OF A NRN OR M. FIBR X / CE, CI
$F \geqslant 0$	F ← CE*FEX - CI*FIX, WHERE:
UNIT IS HZ	FEX IS THE FF OF THE EXCSET OF X
	FIX IS THE FF OF THE INHSET OF X

Anything said about FF at a higher level inherits to the lower. On the right, CE and CI are constants, depending on the instance of FF; one FF exists for each instance of X. The ← means "controlled by", i.e. F "knows who gives it its value". This notion, which is easy to make precise, I use as the underlying model of the notion of causality for real variables. Note that NRNs and M. FIBRs have EXCSETs and INHSETs (SETs OF NRNs) which in turn have FFs. (We often use the name of a PROCESS, e.g. FF, if it has one variable, to refer to that variable, e.g. F). TIME does not appear explicitly in FF since FF is a function of other functions of TIME.

Unfortunately, space does not permit discussing the negative feedback loop behaviour of the complete reflex arc. Such negative (and sometimes positive) feedback loops are a major aspect of many important physiological systems. Hierarchical descriptions of their function, particularly of the controlling relations between them, is a central problem in this work.

The single most complex process which must be described in discussing the synapse is the action potential:

THE ACTION POTENTIAL (AP) OF A NRN N IS A TRANSIENT IN THE MEMBRANE POTENTIAL OF N LASTING ABOUT 10 MS.

This is the highest level statement that it is worthwhile to make about an AP. The underlined words are primitives that any LESK system would understand. The term TRANSIENT though, we shall have to define. This definition assumes that elsewhere the PROCESS MEMBRANE POTENTIAL (MP) belonging to EACH NRN has been defined. The MP itself is a complex result of a number of other interacting processes, whose mutual controlling relations produce a stable value of the MP unless perturbed.

Now let's define TRANSIENT. We want to be able to do this in such a way that the machine will understand the term in any similar context. A TRANSIENT is a STATE OF A PROCESS of finite duration (we will not attempt to define "short") during which the normal values of the variables undergo some changes:

SIMDEF A TRANSIENT X IN A PROCESS Y LASTING TIME Z
X IS A STATE OF Y
X'S DURATION IS Z
DURING X: SOME VARIABLES OF Y ARE CHANGING
" " " " " NOT NORMAL
Y IS NOT NORMAL
END

This definition is somewhat lower in the detail hierarchy. It has a much higher proportion of primitives than the higher level ones; in fact the only non-primitive term (i.e. that we cannot expect the system to already know) is CHANGING, which I have included deliberately. First, we have decided to call X a STATE OF Y (recall that PROCESSES CAN HAVE STATES). A STATE OF A PROCESS is a set of CONDITIONS (predicates on the real variables of the PROCESS). Another semantic axiom the machine (and the user) must know is that STATES HAVE DURATIONS, which is a CHARACTERISTIC. The DURING (meaning $\forall t$ in this INTERVAL) part of the definition introduces the most minimal reference to time: these are the statements the machine would look at if in response to

some question, at some detail level, the PROCESS under consideration had been put into a TRANSIENT STATE, so that no assumption would be made that all variables were constant, or that Y was NORMAL (this is probably a useful primitive, since, unless otherwise stated, all THINGs and PROCESSES would by default be in the NORMAL STATE). I feel this captures the essential meaning of the phrase IS A TRANSIENT IN.

Now what about the adjective CHANGING? This is getting harder, for we are being unavoidably drawn "downward", toward those devilish real variables that lurk under everything. But besides VARIABLES, one may speak of other IDEAS as changing too. Which ones? This is a semantic axiom question again:

ONLY STATES AND VALUES OF CHARS CAN CHANGE

Though we haven't space to discuss CHARACTERISTICS (CHARs), they always link some IDEA to a VALUE, such as a NUMBER, a TRUTH VALUE, an ADJECTIVE, or a few more debatable IDEAs. We have decreed then that THINGs for example never CHANGE, only their STATEs and the VALUEs of THEIR CHARs do. For example, part of the NORMAL STATE of a THING X is X EXISTS. Every GROUP, THING OR PROCESS X has a special STATE: X DOES NOT EXIST, in which no other statements appear. Entering this "dead" STATE is the ultimate "CHANGE" for such an X. But back to our problem. Let's say this :

```
EVDEF    CHAR X IS CHANGING AT TIME T        (I have assumed simple
         BEF:  LET X1 = VALUE OF X            understanding of the
         AFT:  VALUE OF X ≠ X1                equivalence of IS and
END                                           ARE.)
```

The important thing to note here is that we have descended low enough so that our definitions are beginning to take on a procedural aspect. The semantics of DURING in the TRANSIENT definition would combine with the simulation level definition of a TRANSIENT, which would advance time in small discrete steps, and with the CHANGING definition to check that after each time step, at least one variable had a new VALUE, lest the definition of CHANGING, and hence TRANSIENT, be violated.

Let us go on now to ask a question: "What triggers an AP?"

THE AP OF A NRN N OCCURS WHENEVER THE MP OF N BECOMES GREATER THAN THE THRESHOLD OF N

The axiom from which this statement derives is:

EACH EVENT OR PROCESS IS CAUSED BY AN EVENT OR AN (EVENT, THING)

The latter TUPLE accounts for our frequent way of saying that "a thing did something, which caused...." To map the form of the question onto the form of the fact, all that is needed is a kind of sequiv:

EVENT X TRIGGERS A P-STATE Y (OF PROCESS Z) IFF Y OCCURS WHENEVER X

I view OCCURS WHENEVER as one of a small number of causal primitives, linking EVENTS, PROCESSES and STATE CHANGES, which serve to answer questions about causality at high level, and to act as an "executive" to the searching and running of lower level descriptions. Suppose we asked:

WHAT HAPPENS WHEN THE MP OF A NRN N BECOMES GREATER THAN THE THRESHOLD OF N?

This is a very vague question, in terms of how "deep" an answer is expected. First of all, the high level causal links would be traced to determine the immediate consequences: THE AP OF N OCCURS. But now there are two directions to consider: further causal consequences, and more detail, e.g. developing the TRANSIENT description of an AP. The interrogator would have to specify to what level she wished the description. I envision three or four levels, in different "perspectives" (the KRL term), between the top-level AP description and the simulation level, and shall briefly discuss the kinds of knowledge which inhabit these regions.

For example, one way of amplifying our AP description would be:

DURING THE AP OF A NRN N: 1. THE MP RISES FROM THE THRESHOLD OF N TO ABOUT 30 MV IN ABOUT .5 MS; THEN 2. IT DROPS TO ABOUT -90 MV IN ABOUT 1 MS; etc...

This is a serial sub-process description: its main utility is that it matches the visual appearance of an AP on an oscilloscope; it also provides a rough first approximation to a simulation. But the important underlying processes (the answers to "why" questions) are several parallel interacting ionic conductance changes. The whole trick will be to be able to define constraints on these processes first individually, and then to combine them by specifying only their interaction constraints, as one could do in an analog simulation, or in mathematics, e.g.:

$i_{Na} = E_{Na}g_{Na}(V,t)$; $i_K = E_K g_K(V,t)$; E's are constant

$V = (E_{Na}g_{Na} + E_K g_K)/(g_{Na} + g_K)$; $i_{Na} + i_K = 0$

We see that V (the MP) is defined by a set of simultaneous equations, a luxury we could make great use of in AI (I mean the simultaneity, not numerical equations). A machine could easily derive automatically most of the descriptive conclusions about V that a person would from these equations.

But there is a large grey area of description of interacting processes that is not contained in the equations, certainly so when we don't have any equations. For example, during phase 1, MP is being controlled by a regenerative interaction between it and the sodium conductance (g_{Na}). The threshold voltage is that at which the regeneration has a "loop gain" (the engineers have

some good ways of describing such things) exceeding one. This rapidly accelerates the increase in MP. Phase 1, however, gives way to phase 2 evidently because this regenerative relation can only be transient: g_{Na} seems to "shut itself off", to use the words of one well-known text. Such mutual transient causalities are the kind of description which inhabit this grey area, which I am now investigating.

Some Related Projects

The project whose raison d'etre is most similar to mine, in terms of the intended application at least, is the REL project (Thompson and Thompson, 1975). This is not an "AI oriented" project (though it seems to be heading into AI); rather its goal has been to develop a running system rather than a theoretical system. Unlike LESK but like most "data base" systems, REL's orientation is toward the querying of a large data base of instances of classes, rather than toward organizing the categorical (class level) knowledge, as is done in LESK. But REL is certainly an advanced system amongst those which are now appearing in this area of "intelligent natural language front ends to data bases".

The MEMOD/SOL system of Norman and Rumelhart (1975) is a semantic net oriented English language question answering system which, like REL, has the distinction of being a long-standing project which is relatively advanced in implementation. It is not oriented to scientific data, nor to a large instantial data base, but puts more emphasis on the underlying theoretical structures, though hierarchical descriptions and quantification are given little consideration. Both these systems, however, give the user the ability to define a considerable variety of terminology, an essential ability for any practical system.

Moving toward more theoretical systems, I have already noted Bobrow and Winograd's KRL, presently in the process of definition and implementation (via INTERLISP). Though having no one application in mind, these authors share my concern with the problem of discovering underlying semantic structures. Some synthesis of ideas developed both in KRL and in LESK will probably be appropriate within a year.

Sandewall has reported (1972) on an ambitious system, PCF-2, which attempted to axiomatize the use of many words and phrases, together with a delineation of the semantic categories, with similar motivation to that of LESK, though with quite a different approach, there being no discussion of the machine representational structures. This project, however, does not seem to have been reported on further.

Concluding Remarks

In searching the AI literature for ideas on how to approach this problem, I have been struck by two things: first, that the bulk of this work is devoted to representing mainly pyschological and sociological aspects of human situations; and second, that there is not enough cognizance of work in other areas of computer science. Since I feel the representation of physical knowledge to be a more tractable problem than the representation of psychological phenomena, and since any "understander" for human affairs will certainly need to know about the physical world, the semantics of physical phenomena should be given more attention, even if one's goal is not specifically that. On the second point, I have obtained many ideas about what relators should look like by considering notions like abstract data structures (e. g. as in the language CLU (Liskov and Zilles, 1975)), modern parallel process simulation languages (e. g. DELTA, Kyng and Pedersen, 1974), and analog computers. Such broad considerations characterize the approach I have been taking as "top down", in that I am attempting to appreciate the fullest variety of problems involved before writing code.

The LESK approach may finally be described as an attempt to provide a common precise set of semantic structures, to be shared by humans who wish to express some technical knowledge clearly and by machines intended to digest this knowledge, such that the enormous gap in subtlety and complexity between the human's and the machine's representational system may be more comfortably bridged. Unlike most "natural language" systems, LESK puts the onus for clear communication back on the human user, who is most able to accept it.

A final question: could a body of knowledge, developed and residing in a LESK-understanding machine, enabling a scientist to predict the outcome of experiments and to gain insight into the structure of nature, be called a theory? We may eventually be forced to reconsider exactly what is meant by the term theory.

Acknowledgements

The author gratefully acknowledges the support of the I. W. Killam Memorial Fund of the Montreal Neurological Institute, and the generous and valuable assistance of John Mylopoulos.

References

Bobrow, D. and Winograd, T. An overview of KRL, a Knowledge Representation Language. Available from authors. (May, 1976).

Brown, J. S., Burton, R. R., and Bell, A. G. SOPHIE. A sophisticated instructional environment for teaching electronic troubleshooting. BBN Report No. 2790. (Mar. 1974) Bolt Beranek and Newman Inc.

Hendrix, G. Modeling simultaneous actions and continuous processes. Artificial Intelligence. (1973).

Kyng, M. and Pedersen, B. M. Description of a model of a single helix pomatia brain neuron and an associated neurophysiological experiment. DELTA Report No. 3. Institute of Mathematics, University of Aarhus. (1974).

Liskov, B. and Zilles, S. Specification techniques for data abstraction. IEEE Trans. on Software. vol SE-1 no. 1 (Mar. 1975), 7-19.

Norman, D. A. and Rumelhart, D. E. Explorations in Cognition. W. H. Freeman and Company. San Francisco. 1975.

Sandewall, E. PCF-2, a first-order calculus for expressing conceptual information. Dept. of Computer Science. Uppsala University. 1972.

Skuce, D. An English-like language for qualitative scientific knowledge. Advance Papers of the Fourth International Joint Conference on Artificial Intelligence. Tbilisi. (Sept. 1975), 593-600.

Smith, B. A computational model of anatomy and physiology. MIT AI Lab memo. (Feb. 1976).

Sussman, G. J. and Stallman, R. M. Heuristic Techniques in computer-aided circuit analysis. IEEE Trans. on Circuits and Systems. vol CAS-22, no. 11 (Nov. 1975), 857-865.

Thompson, F. B. and Thompson, B. H. Practical natural language processing: The REL system as prototype. in: Advances in Computers, vol 13. Rubinoff, M., and Yovitts, M. (eds.) Academic Press. New York. 1975.

Of Crayfish and Production Nets

Peter F. Rowat

Computer Science Department, University of British Columbia, Vancouver, B.C., V6T 1W5.

Abstract. In the first half it is argued on methodological grounds that animal simulation is an appropriate task to tackle in artificial intelligence, and the crayfish is proposed as a candidate. Some relevant aspects of the crayfish are presented. In the second half the basic system requirements for any organism controlling program are discussed in psychological terms. Production nets, derived from the production system approach but having a high degree of parallelism, are then introduced as an implementation system.

1. Why simulate a crayfish?

Whatever one's approach to artificial intelligence, there is no doubt that a theory of the organization of intelligent organisms is needed, whether man, machine, or animal. A sound methodological approach to developing such a theory would be to satisfactorily explain the most elementary behaviours before progressing to more advanced ones. In fact there has recently been a shift of emphasis in artificial intelligence from advanced behaviour such as game playing and formal reasoning to more mundane behaviours such as stacking blocks and common-sense

reasoning. But, while most effort in artificial intelligence is focussed on human behaviour at its various levels, no-one has any idea how to write programs to exhibit behaviour strictly comparable with that of creatures lower down the evolutionary scale. For instance, no-one can specify all the information processing mechanisms required to produce behaviour as complex as that of a cat. From this there follows the suggestion that perhaps one should start by simulating the sensory input and motor output of some simple organism, write control programs for it and study the intelligence or appropriateness of the resultant behaviour. A successful control program, if written in a uniform and generalizable way might, then, contain the germ of a theory of intelligence. The computer language or system used for such an implementation would, in a sense, be the theory.

Other arguments can be adduced for this radical suggestion. For instance, there is an established psychological view[1,11] that thinking is closely related to skilled motor behaviour. If one also admits that skilled behaviour means roughly the same as intelligent behaviour, then it follows that an understanding of the information processing mechanisms required to produce intelligent behaviour in a simulated organism will probably be basic to understanding the higher mental processes. Other pointers in the same direction are these. On purely introspective grounds, the thinking and imagery involved in mathematics and problem solving is qualitatively little

different from that involved in more mundane activities such as moving a piece of furniture around a house. And lastly, since even the simplest creature in the world has to have good ways of handling space and objects to survive, the importance of understanding this task before aiming for higher goals is obvious.

But what organism should be simulated and by what criteria should one judge whether "intelligent" behaviour has been produced by the organism controlling program? When the organism is human then the criteria are usually clear. If the organism is some arbitrary robot then "interesting" behaviour may be produced, but it is impossible to provide any substantial basis for this judgement, for the only extant models of intelligent behaviour are the creatures around us. One is then forced to the conclusion that one should simulate some simple creature for which the lowest level sensory input and motor output are known and whose highest level behaviour is also known in detail.

It was once suggested that this is a bad direction to take because animals don't solve any difficult problems[9]. But a cursory glance at the animate kingdom reveals many features of behaviour which are almost universal. These include:

(1) Multiple processes running in parallel;

(2) Continual perception-action responses ranging from the simplest knee-jerk reflex, through higher level predator-flight responses, up to the most sophisticated

human ones;

(3) Aspects of behaviour which go under the general terms set, attention, orienting reflex;

(4) Adaptive behaviour which occurs in some form in the very simplest of creatures, ranging from response habituation to insightful learning (remember Kohler's monkey and bananas?).

Any system displaying these behavioural features would be of considerable interest to artificial intelligence.

Back to the question of simulation, what creature should be simulated? The crayfish is a good candidate. This small, common, lobster-like creature is easily observed and its sensory input and motor output have been extensively documented[17]. Some of the basic facts follow.

It is a few inches long and dwells under rocks and in natural crevices in freshwater ditches. Its central nervous system contains less than 10^5 neurons, 7-8 orders of magnitude fewer than in man. Its sensory equipment includes two compound eyes on moveable stalks, two long antennae and two short antennules on its head, two statocysts(gravity detectors), and many tactile hairs. The antennae and antennules have many vibration, displacement, and chemo-receptors. Its motor equipment includes three pairs of appendages to pass food to the jaws, a pair of stout pincers, four pairs of legs for walking, five pairs of small swimmerets for swimming, and a strong

fan-shaped tail which it uses for swimming and, with a flip under its body, to escape backwards from danger in a hurry. Its compound eyes, each containing several thousand component ommatidia, send seven known types of fibers or messages to the brain. The most interesting of these are the jittery movement or "bug-detector" fibers as in the frog, and the space-constancy fibers with visual fields which change in size and location on the retina with the position of the animal's body in space. Three other classes of visual fibers are concerned with movement detection. The statocysts, antennules and antennae between them send messages concerning the frequency, amplitude, and direction of water-vibration to the brain[18]. There are several simple reflexes such as claw opening and closing, escape and defense. These last two involve the whole body and quickly habituate to repeated stimuli. More complex reflexes are those of feeding, copulating, and righting, and at a higher level the crayfish can learn to run a simple maze[18]. It has a simple visual memory which lasts up to eight minutes. In a brainless crayfish the reflexes of feeding and copulating, once started, continue without stopping, which suggests that these reflexes are independent processes, controlled by the brain through inhibition.

2. Production nets as a means of implementation

What language or system should be used to implement an organism controlling program? It will be easier to answer this question if the above behavioural features are rephrased in more system-oriented language.

The first and third features obviously require parallelism. However, the multiple parallel processes should not only include the peripheral processes, as required by Neisser's pre-attentive processes[11] and Marr's primal sketch[8], but also the more central processes, in view of Posner's conclusion[12] that an item entering the nervous system activates multiple parallel codes or representations. This is also indicated by the everyday observation that one's level of conversation while driving decreases during the execution of some tricky manoeuver. The interactions between the parallel processes must obey Bobrow and Norman's[2] "principle of graceful degradation", which in operational terms is the "principle of continuously available output". The notions of effort and attention[7] become, respectively, the availability of a variable but limited computing capacity, and a global process which monitors incoming stimuli and distributes computing capacity to the multiple parallel processes. This should probably be identified with Hebb's autonomous central proceess[5]. The orienting reflex becomes the manifestation of this global process when a rapid redistribution of computing capacity is required by the occurrence of a novel event. The second feature requires a uniform way of expressing perception-action responses at all

levels of abstraction, with the implication that hierarchies of description and of action can be naturally formed; and the fourth requires a meaningful self-modification capability. The final requirement to be mentioned is for an inhibitory mechanism: one process must be able to inhibit another. This is a well-known phenomenon in the literature on animal behaviour, and is suggested by the behaviour of a brainless crayfish mentioned earlier.

The most basic feature of any organism is surely the presence of continual perception-action or S-R responses, which immediately suggests a production system implementation for at least the lowest levels of response. Other evidence in their favour is the diverse range of tasks[4] to which they have been applied. Note that this includes several adaptive applications: learning to play poker respectably well[15], generating new rules for a scientific theory[3], modelling the development of a child's seriation ability[6]. Waterman applied them to several psychological learning tasks[16], including a production system version of EPAM. So the requirement of a self-modification ability is certainly attainable. However the parallelism requirement is hard to satisfy, and most production systems have very little structure.

The production net concept is a modification of the production system design to allow a high degree of parallelism and structure. Instead of searching through the whole list of

rules at every step as in a production system, each rule in a production net is connected to a small number of other rules, so a certain amount of search is removed and some structure is added. To allow for parallelism, several rules may fire at once. A production net works in cycles. There is a limited source of computing capacity available and every rule requires a certain amount to fire. This quantity varies from rule to rule but is constant for each rule over time. The left hand side of a rule consists of the conjunction of a number of conditions, some of which may be inhibitory. A condition is represented by a pattern and becomes true when it matches an incoming message. Some rules have conditions that match the occurrence of external events. When a rule fires it sends a message to each of the rules it is connected to, and may initiate one or more external actions. An attention mechanism controls the distribution of computing capacity to rules. If a rule is allocated less than the required amount of computing capacity then even if all its conditions are true it cannot fire. It cannot fire at the next cycle unless all its conditions become true again. The winning rules are those that fire at the end of a cycle. The target rules of a cycle are those to which the winning rules from the previous cycle are connected. The candidate rules are those whose conditions are satisfied. Thus one cycle of a production net consists of five steps.

(1) Collect all messages from the previous winning rules.

(2) Match messages to patterns in the target rules.

(3) Compute the candidate rules.

(4) Use attention mechanism to decide which candidates win.

(5) Fire the winning rules and discard the losers.

An outline scheme for the attention mechanism is this. Every rule has associated with it two numbers, an importance and a probability. Consider the eating rule. One condition is "food being directed towards mouth", and its message is "start chewing". The importance of this varies with the state of hunger, and the probability depends on whether the organism has just tried to scoop a juicy morsel into its mouth. The probability is propagated through the net by the Bayesian technique of Duda, Hart, and Nilsson[19]. The importance usually varies on a much longer time scale. The attention mechanism uses a function of these two numbers to decide which candidate rules win or lose. I acknowledge the encouragement of Alan Mackworth in writing this paper.

References

1. Bartlett,Sir Frederic(1958). Thinking.

2. Norman,D.A., and Bobrow,D.G.(1975) On data-limited and resource-limited processes. Cognitive Psychology 7:44-64.

3. Buchanan,B.G., et al.(1972) Heuristic theory formation: data interpretation and rule formation. Machine Intelligence 7, eds. B. Meltzer and D. Michie, Edinburgh University Press.

4. Davis,R., and King,J.(1975). An overview of production systems. AIM-271, Stanford AI lab., Computer Science Dept., Stanford University.

5. Hebb,D.O.(1949). The organization of behaviour: a neuropsychological theory.

6. Howe,J.A.M., & Young,R.M.(1976). Progress in cognitive development. DAI res. Report no. 17, University of Edinburgh.

7. Kahneman,D.(1973). Attention and effort. Prentice-hall.

8. Marr,D.(1975). Early processing of visual information. AI Memo 340,MIT AI Lab., Cambridge,Mass.

9. Minsky,M.L.(1963). A selected descriptor-indexed bibliography to the literature on artificial intelligence. In: Computers & Thought.

10. Minsky,Marvin(1975). A framework for representing knowledge.

11. Neisser,Ulric(1967). Cognitive psychology.

12. Posner,M.I.(1972). Multiple codes. in: Visual information processing, eds. W.G.Chase & H.A.Simon.

13. Sokolov,E.N.(1963). Perception and the conditioned reflex.

14. Taylor,R.C.(1968). Water-vibration reception: a neurophysiological study in unrestrained crayfish. Comp. Biochem. Physiol.27.

15. Waterman,D.A.(1970). Generalization learning techniques for automating the learning of heuristics. Art. Intell. 1:121-170.

16. Waterman,D.A.(1974). Adaptive production systems. Psych. Dept., Carnegie Mellon University.

17. Wiersma,c.a.g.(1970). Reactivity changes in crustacean nervous systems. in: Short-term changes in neural activity and behaviour, eds. G.Horn & R.A.Hinde, Cambridge University Press.

18. Yerkes,R.M. and Huggins,G.E.(1903). Habit formation in the crawfish Cambarus affinis. Harvard Psych. Studies I:565-577.

19. Duda,Richard O.,Hart,Peter E.,Nilsson,N.J.(1976). Subjective bayesian methods for rule-based inference systems. Stanford Research Institute, artificial intelligence center, technical note 124.

THE USE OF ANALOGUES IN PROBLEM SOLVING

Brian V. Funt
Stanford Artificial Intelligence Laboratory*

Abstract

This paper concerns the use of an analogue as an aid to a problem solving program. Difficulties, including the frame problem, which arise in dealing with even simple physical environments are discussed. To overcome these, a diagram, together with procedures for modifying it, is used as an analogue of the external problem situation. Features are extracted from the diagrams by algorithms running on a simulated parallel processing moveable 'retina'.

1. Introduction

A computer program, WHISPER, has been written which, following the suggestion of Sloman[1], uses analogues in its reasoning. Not only are some difficult problems overcome by using an analogue, but they help the program to obtain solutions in a more straightforward manner. A more detailed description of both the kinds of hurdles that a system such as WHISPER faces, and of the program itself is given in the author's thesis[2].

2. The Domain

We will consider the problem of determining the stability or instability of a stack of objects. If there is an instability, then the sequence of events occuring as objects fall is to be predicted. The objects are of arbitrary shape, have frictionless surfaces, and are of uniform density and thickness. A typical example from this

* This research was carried out at the Univ. of British Columbia

class of problem which WHISPER can handle is shown in figure 1. It is called the 'chain-reaction' problem because object B rotates and collides with object D causing it to topple as well. Problems may also involve sliding objects.

3.1 Finding Motion Discontinuity Points

Once an object is found to be unstable, it is simple to describe its ensuing rotational or sliding behaviour with an equation of motion. However, it is still necessary to determine the point at which the motion described by the equation will be interrupted. The object's behaviour after a motion discontinuity will of course be described by a new equation. Collisions are one of the most likely events to cause a change in an object's motion. They are also one of the most difficult to detect because there is generally little to suggest that one object in the universe is a more likely collision candidate than any other. Although heuristics can be devised to predict some collisions, situations such as that depicted in figure 2 in which there is a surprise collision frequently arise. Any strategy relying on the computation of collisions of only point P on object B, or other strategies which partition the class of possible candidate collision objects on the basis of being members of the same structure or on the basis of being below the falling object would more than likely overlook such situations.

The reason current AI systems have had difficulty with the collision problem is their lack of a good representation for empty space.

Systems such as Fahlman's[3] BUILD describe the position of each object by the coordinates of some arbitrary point on it. No mention is made of where objects are not located. When this approach is used empty space must be found through 'proof', either computational or deductive, of the statement 'there does not exist an object at location P'. Generally, this is accomplished by using the equivalent statement 'for each object O, O is not at P', and individually testing all the objects in the universe. The result is unmanageable growth of computational requirements.

3.2 The Amalgamation Problem

The problem of combining two separate descriptions into a new description is the amalgamation problem. Frequently WHISPER must imagine two objects to be glued together; the amalgamation problem arises in forming the description of the combined object. Some of the properties the combined object must inherit are: its shape, center of gravity, and position relative to other objects.

3.3 The Frame Problem

The frame problem[4] concerns the updating of a system's representation of the world to reflect the effects of an action in the world. Hayes[5] has pointed out that this actually is not a single problem, but involves several issues; issues which apply in WHISPER's domain. Although the direct effects of an action may be quite trivial, (for example, when an object moves, the direct result is that its position has changed) there are many other more complex

context-dependent effects which can also result. Long causality chains arise in the physical environment which must be properly inferred in order to propagate the effects of the action into the system's representation of the state of the world. An example of this is the way the type of surface contacts made between arbitrarily shaped objects change when an object is moved.

In addition to knowing what is affected by an action it is also necessary to know what remains unaffected. If it is not known that a property is unaffected by an action, then in general we can say nothing about that property still holding after the action occurs. One aspect of the frame problem is finding a good way to express the multitude of things unaffected by each action without having to state each separately. Some things which remain unchanged when an object moves in WHISPER's domain are: the positions of all other objects, the shape of objects, and the contacts between objects other than those involving the moved object.

4. WHISPER

The overall structure and organization of the WHISPER system is shown in figure 3. Its essential components are: the qualitative physical knowledge, the retina, the redrawing transformation procedures, and the diagram. The qualitative physical knowledge is the domain dependent part of the system, consisting of 'specialist' procedures expressing elements of the behaviour of rigid bodies when acted upon by gravity.

4.1 The Qualitative Physics

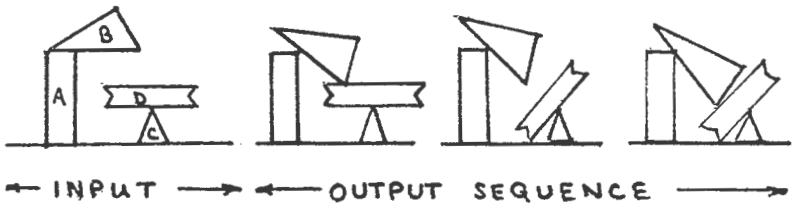


FIGURE 1

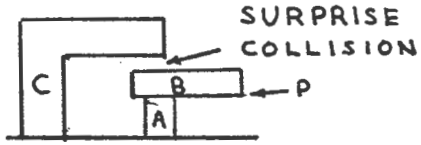


FIGURE 2

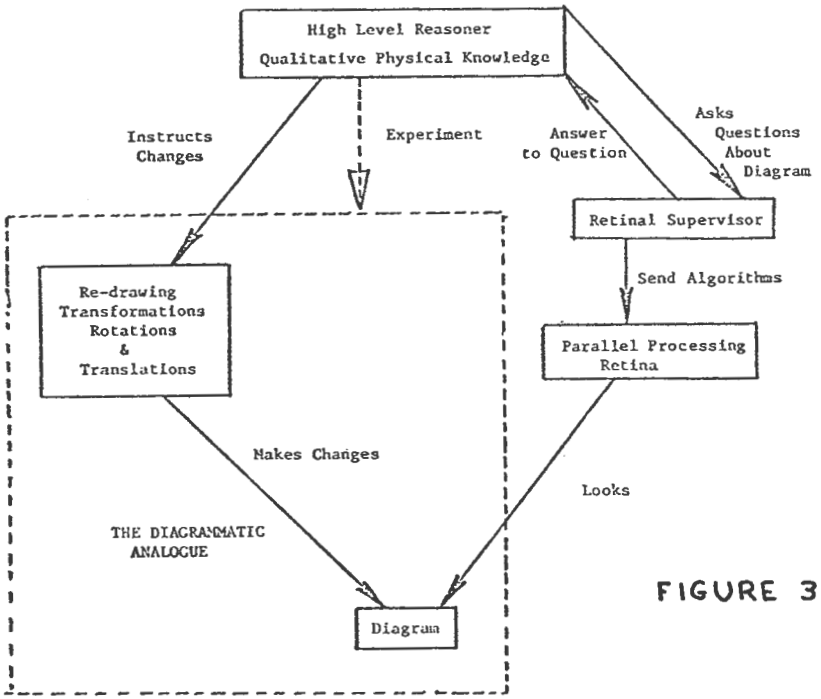


FIGURE 3

Knowledge of physics is represented procedurally. Each specialist encapsulates a qualitative piece of knowledge such as: 'if the center of gravity of an object does not have supports to both its left and right, then it hangs over too far and will topple'. In contrast to Fahlman's BUILD system, WHISPER's understanding of Physics is closer to a child's than an engineer's.

When a specialist requires information about the state of the world in deciding the applicability of its knowledge to the current situation, it sends a request to the retina to examine the diagram for the presence of a specific feature. The specialist interprets that feature relative to the current domain. For example, a specialist which needs to know if object X supports object Y, asks the retina to see if Y is above X and Y thouches X in the diagram. If the qualitative knowledge discovers that a change of state, an action, will occur in the world, then it calls the redrawing transformation to modify the diagram to reflect the effects of this action.

4.2 The Retina and Its Perceptual Primitives

The purpose of the retina is to extract information from the diagram in response to queries from the qualitative knowledge specialists. Its role parallels the human eye and its early perceptual processing stages. The retina is basically a parallel processor, and algorithms, called perceptual primitives, have been designed to execute on it. Due to parallelism, their execution times are of the

same order of magnitude as more conventional operations. Each perceptual primitive determines whether a particular feature exists in the diagram as seen from the current location of the retina. The current set of percepts includes: similarity, center of area, symmetry, contact points, visualization of rotations, nearest and farthest locations satisfying an arbitrary predicate, and curve tangents, convexities and concavities.

The retina consists of a large number of processors operating in parallel (simulated parallelism) with communication links between neighbouring processors only; that is with the exception of a shared link to a supervisory sequential processor. The diagram-to-retina mapping defines the geometry of the retina which is as shown in figure 4. Each 'circle' represents the area of the diagram from which the corresponding processor receives input. Thus the resolution at which the diagram is seen decreases towards the periphery of the retina. This is similar to the varying acuity of the human eye as is the ability to move the retina over the diagram and 'fixate' it at new locations. A useful property of the chosen retinal geometry is that the image of an object can be rotated about the retinal center by simple message passing between circumferentially neighbouring processors.

4.3 The Diagram

The diagram the retina 'looks' at is the pattern formed by values in a two-dimensional array. The combination of WHISPER's retina and

array diagrams parallels human use of diagrams represented on paper, not human visual imagery. The diagrams are constructed so that objects' shapes and positions are represented by corresponding shapes and positions in the diagram. The diagram allows WHISPER to work with both convex and concave irregularly shaped objects without added difficulty. For easy recognition, each object is shaded a different colour, and contours of objects are shaded a colour related to the colour of their interiors.

The Analogue

The combination of the diagram and transformations applied to it is an analogue of a situation involving a stack of physical objects. An analogy exists both between the static states of the diagram and the static states of the physical situation, and between the dynamic behaviour of objects in the diagram and the behaviour of objects in the world. Of course, the behaviour in the diagram and behaviour in the world are not identical; objects in the diagram do not automatically begin to move as do objects in the real world. However, many aspects of an object's dynamic behaviour are properly portrayed when it moves in the diagram. If an object moving in the diagram collides with another object, then a collision will also occur in the world. Similarily, if a path is clear in the diagram, then it is clear in the world. Moving an object also causes its support and contact relationships to change.

5. How Using an Analogue Overcomes the Described Problems

Having outlined a set of difficulties common to simple physics situations, and a system for manipulating and processing diagrams as analogues of these situations, the question is: does the use of the analogue help overcome these problems? Let us briefly consider the how the analogue provides a solution to each of them.

5.1 Amalgamation

The shape of an object is represented by the shape of a shaded area in the diagram. Different objects are shaded different 'colours' so the amalgamation of two shapes into a single new object is only a matter of ignoring their colour difference.

5.2 Motion Discontinuities

The diagram alleviates the empty space problem which is important to collision detection. Physical space in the problem domain is represented by space in the diagram. There is no need to prove that a particular point in space is unoccupied, it is only necessary to look in the diagram for areas of empty space. To detect a collision the 'image' of an object an object is incrementally rotated on the retina, and after each increment a quick parallel check is made to determine that the space now occupied by the object was previously unoccupied. If it was occupied then a collision has occured. The structure of the retina and the diagram-to-retina mapping is such that although a collision will never be missed, the angle of increment is large and therefore the retinal rotation is fast.

5.3 The Frame Problem

Because WHISPER relies on a diagrammatic analogue as a representation of the state of the world instead of a description, it is not troubled by the frame problem. The state of the world is represented by the state of the analogue, and action in the world is represented by corresponding action in the analogue. The corresponding action is the application of the appropriate transformation, and the effects of the action are correctly represented by the resulting state of the analogue.

In the chain-reaction problem the qualitative knowledge procedures know that the action of B's rotation is represented by calling the rotation transformation procedure to redraw B at its new location in the diagram. It can proceed just as if the resulting diagram were its original input and it were starting a brand new problem. The most important information which has changed in the transition between the states as a result of the rotation is: the position and orientation of object B; the position of its center of area; the contacts it makes with other objects; and the shape of the areas of empty space. There are also a multitude of things which have not changed and are correctly left unchanged by the rotational transformation, such as the position of all the other objects, the shape of all objects, and the contact relationships of other objects not involving B. All of these things work out correctly without the need of any deduction or inference on WHISPER's part. All that it need do is to use its retina to look at the diagram and extract whatever information it needs.

ACKNOWLEGEMENT

Raymond Reiter and Alan Mackworth in addition to many other people provided invaluable assistance in the development of these ideas. The financial support of NRC grant A7642 is gratefully acknowledged.

Bibliography

1. Sloman, A. Interactions Between Philosophy and Artificial Intelligence: The Role of Intuition and Non-Logical Reasoning in Intelligence. Artificial Intelligence, 2(1971), 209-225.

2. Funt, B. WHISPER: A Computer Implementation Using Analogues In Reasoning. Ph.D. Thesis, Department of Computer Science, Univ. of British Columbia., (1976).

3. Fahlman, S. A Planning System For Robot Construction Tasks. AI TR 283, M.I.T. (May 1973).

4. Raphael, B. The Frame Proplem in Problem-Solving Systems. Artificial Intelligence and Heuristic Programming. (Ed.) Findler, N. and Meltzer, B. Edinburgh University Press(1971), 159-169.

5. Hayes, P. Some Problems and Non-Problems in Representation Theory. AISB Summer Conference Procedings. (July 1974), 63-79.

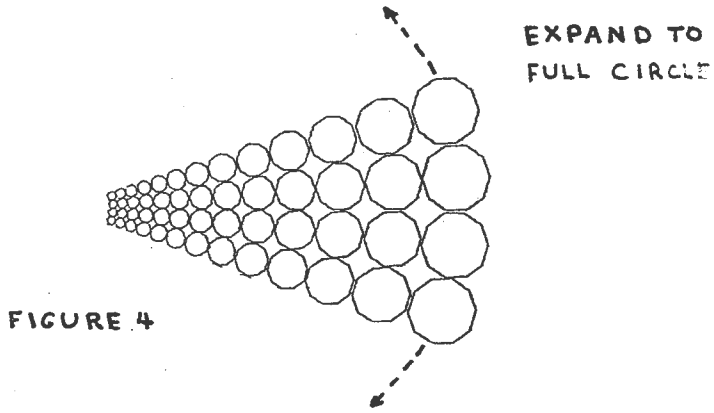


FIGURE 4

Adversary Arguments for the Analysis of Heuristic Search in General Graphs*

by

Jay Munyer and Ira Pohl
University of California, Santa Cruz
Information Sciences

Abstract

The classic heuristic search algorithms have been analyzed to-date only in the case of tree domains. A worst-case analysis is attempted over general graphs using adversary arguments.

a) Previous work in the field of heuristic search is reviewed.

b) The use of adversary arguments is common in complexity analysis; and the essential themes from this work are related to the heuristic search problem.

c) A proof is given for the following new result:

A heuristic search routine (the familiar path finding algorithm) using a heuristic of bounded error e, conducting a search in a graph whose maximum outdegree is b, will find a solution path by expanding more than

$$b^{w(2e-k) + (1-w) \cdot \alpha} \text{ nodes}$$

* Work supported by National Science Foundation Grant 443150.

$$\text{where} \quad \begin{cases} w \le \frac{1}{2}, \text{ the weight on the heuristic term} \\ k = \text{length of shortest solution path} \\ \alpha = \lfloor (k-1)/2 \rfloor \cdot ((w/(1-w)) \cdot ((2e+1) + 2) \end{cases}$$

The Cook [1971] results on NP-complete problems and the extension of these results through reduction arguments [Karp 1972] led to a realization that only heuristics can achieve reasonable solutions for otherwise intractable problems (e.g., [Garey et al. 1972]). These results and advances in understanding of a formal model for heuristic search have together reignited a quest for analytic results in the theory of heuristic search.

The basic algorithms for heuristic search as described in the A. I. literature are outlined in Pohl [1970a] or Nilsson [1971]. These models viewed deductive searches as path problems in state spaces. Originally, they were primarily inspired by the empirical work of the University of Edinburgh researchers ([Doran and Michie 1966], [Ross 1973]). Two groups analyzed the efficiency of these methods, the SRI robotics team [Hart, Nilsson and Raphael 1968] and the University of California, Heuristic Theory Project ([Pohl 1969],[Pohl 1973]).

The Pohl model, based on a generalization of the Graph Traverser and algorithm A*, used an algorithm HPA.[1] The different algorithms yielded a variety of results. The GT work focused on the empirically observed efficiency of heuristic search as it related to the quality of the heuristic function. The A* work

1. See appendix for a formulation of this model.

proved a dominance result in the restricted case that heuristics estimated distances, and that these heuristics were used in a branch-and-bound fashion to achieve a least costly or shortest solution path. The HPA work abandoned any restrictions on how the heuristic term would be combined with the cost-to-date term and analyzed worst-case performance on the domains. While not explicitly stating the analysis in terms of oracles [Knuth 1973] or adversary strategies as they are now called, this was one of the first uses of this important technique in the analysis of algorithms [Aho, Hopcroft, and Ullman 1974].

The analytic results in heuristic search theory had two important offsprings. First, there was a reinterpretation and application of these ideas to the theorem proving domain [Kowalski 1972], [Waidelich 1973]. Second, there was the understanding and extension of certain dynamic programming methods and allied combinatorial search routines in these terms [Martelli and Montanari 1975], [Camerini, Fratta and Maffioli 1973].

The current extent of all these research efforts would require an article by itself too lengthy to fit in these proceedings. Instead, we will mention some recent highlights and continue onto the main results of this paper — an adversary argument extending the analytical results on heuristic search into general graph domains.

Recent Results

Dynamic Weighting: [Pohl 1975]

In dynamic weighting, the relative weighting of h(x) and g(x) are affected by their depth in the search tree. The deeper into the search the less weight is placed on the heuristic term.

$$w(x) = 0.5 + B(x), \quad 0 \le B(x) < \tfrac{1}{2}$$
$$\text{where } 1/B(x) \propto \text{search depth}$$

In intuitive terms, this is done to avoid being excessively misled by an overly optimistic heuristic. However at the top levels of the search there is stronger reliance on the heuristic term in order to promote a depth first search. Dynamic weighting retains some of the advantages of A^* where a least costly solution is desired. These advantages were demonstrated by applying these techniques to the Traveling Salesman Problem using the Held-Karp scheme to obtain a heuristic estimator [Held, Karp 1971]. Similar ideas are applicable to game tree searches such as the Harris Bandwidth search [Harris 1973].

Pruning and Partial Development: [Ross 1975], [Michie and Ross 1970]

Ross has considered an algorithm GT4 which only partially develops nodes and is used in conjunction with search tree pruning. He has extended many of the analytic results of the HPA scheme to this more general method.

Theorem [Ross 1975]: Given that the heuristic function satisfies the monotone condition, then $GT4^*$ over tree domains will in the worst-case look at no more nodes by using $f = g + h$ in comparison to $f = h$.

Furthermore Ross gave conditions under which staged searches invoking pruning led to solution paths identical in length to those obtained by full search without pruning.

Complexity of Path Searches: [Johnson 1973], [Martelli 1975]

Johnson showed how Dijkstra's algorithm (HPA^+ with w = 0) could run in exponential time on graphs with non-negative cycles but with negative edge lengths. This comes from nodes being placed back in the candidate (open) set exponentially often. Martelli extended this result to A^* using admissible heuristics and showed how A^* could be improved to avoid this catastrophe.

"Algorithm B (Martelli) is thus a simple variant of A^* and can be obtained from it by substituting for steps (1) and (3) the following: (chapter 3, Nilsson 1971)

(1') Put the start node s on a list called OPEN.
Set

$\bar{g}(s) \leftarrow 0$, $\bar{f}(s) \leftarrow \bar{h}(s)$, $F \leftarrow 0$

(3') If there are some nodes in OPEN with $\bar{f} < F$, select among them the node n whose $\bar{g}$ value is smallest; otherwise, select the node n in OPEN whose $\bar{f}$ value is smallest and set $F \leftarrow \bar{f}(n)$. (Resolve ties arbitrarily, but always in favor of any goal node.) Remove n from OPEN and put it on a list called CLOSED."

Note that this algorithm is breadth first within a radius defined by consecutive values of F.

Adversary Arguments

Algorithms performances in problem domains can be compared with respect to their effectiveness on bench mark problems. Adversary arguments are attempts to manufacture very difficult conditions that conform to the problem constraints. The best adversaries lead to the worst-case performance for algorithms in a given problem domain [Knuth 1973].

e.g. adversary for HPA in tree domains:

given: $h^*(x)$ is the perfect estimator

$h(x)$ is of bounded error ϵ

Let $h(x) = h^*(x) + \epsilon$ along shortest path otherwise $h(x) = h^*(x) - \epsilon$; it can be proved that this adversary leads to worst-case performance for HPA acting on tree domains [Pohl 1970b]. (A recent analysis of alpha-beta game tree search was published using this form of argument [Knuth, D. E. and R. W. Moore 1975].)

We now turn to a summary of results extending our analysis from tree domains to general undirected graph domains. An interesting aspect of these results is a comparison between HPA, the original algorithm of Pohl which did not allow nodes in S to be placed back into $\bar{S}$ (i.e., did not allow closed nodes to be reopened), and HPA+, which does allow a closed node to be reopened when a shorter path to it is found. It is necessary to be able to do this when a shortest path to a goal node is desired, but it had not been thought that this would be efficient when any solution path found is satisfactory. Our results however show that, for the worst-case at least, allowing closed nodes to be

reopened significantly reduces the effort required to find a solution.

These results are restricted to the case of $w \leq \frac{1}{2}$, which includes algorithm A* of Nilsson ($w = \frac{1}{2}$), breadth-first algorithms such as Dijkstra's shortest path algorithm ($w = 0$), but not "pure heuristic search" which has $w = 1$. Also, we consider the "path problem" in which all edges are of unit cost, so $g(x) = g(\text{parent}(x))+1$. The results can easily be extended to the case or arbitrary positive edge costs, but not to negative edge costs.

LEMMA 1. If HPA+ is used with a heuristic function of bounded error e in a graph with shortest solution path of length k, then for all nodes N which are expanded in the discovery of a solution (not necessarily the shortest path), we have

$$f(N) \leq \begin{cases} w \cdot e + (1-w)k & \text{if } w \leq \frac{1}{2} \\ w \cdot e + w \cdot k & \text{if } w \geq \frac{1}{2} \end{cases}$$

Pf: (sketch)

Let $u_o = s, u_1, u_2, \ldots, u_k = t$ be a shortest path, then there is always some $u_i \in \mathcal{S}$, the candidate set. Furthermore, it can be shown that there will be such a u_i, with $g(u_i) = i$. For this candidate we have

$$f(u_i) = w \cdot h(u_i) + (1-w)i$$

with $h(u_i) \leq h_p(u_i) + e$, taking this maximum value on the solution path as our oracle gives for $w \leq \frac{1}{2}$ a maximum value of $f(x) \leq w \cdot e + (1-w) \cdot k$. Since at any time there is at least one node

in the candidate set which does not exceed this value — no node can be placed in S of larger f-value. q.e.d.

A similar argument works to give the result for $w > \frac{1}{2}$.

Definition for any node n in a graph,

$h_p(n)$, the "perfect heuristic", is the length of the shortest path from n to the goal node t .

$g_p(n)$ is the length of the shortest path from the start node s to n which does not include the goal node (i.e., the shortest path from s to n which would be found by HPA).

$f_p(n) = w \cdot h_p(n) + (1-w) \cdot g_p(n)$.

Theorem 2. If HPA+ uses $w \leq \frac{1}{2}$ and a heuristic function of bounded error e, then in a graph of shortest solution path k, the set S of expanded nodes will always be a subset of a set of nodes T where for $n \in T$, $f_p(n) \leq 2 \cdot w \cdot e + (1-w) \cdot k$.

Pf: By Lemma 1 we know that for any node n which is expanded,

$$f(n) = w \cdot h(n) + (1-w) \cdot g(n) \leq w \cdot e + (1-w) \cdot k$$

Since $h(n) \geq h_p(n)-e$ and $g(n) \geq g_p(n)$, this becomes

$$w \cdot h_p(n) - w \cdot e + (1-w) \cdot g_p(n) \leq w \cdot e + (1-w) \cdot k$$

or

$$w \cdot h_p(n) + (1-w) \cdot g_p(n) = f_p(n) \leq 2 \cdot w \cdot e + (1-w) \cdot k.$$

Definition: The branching rate b of a graph G is the maximum of (degree - 1) of any node in G.

Corollary 3 If HPA+ used $w \leq \frac{1}{2}$ and a heuristic function of bounded error e, then in a graph with branching rate b and shortest solution path of length k, we will always have

$$|S| \leq b^{2(w/(1-w))e+k}$$

Pf: By Theorem 2,

$$S \subseteq \{N \mid w \cdot h_p(n) + (1-w) \cdot g_p(n) \leq 2we + (1-w)k\}$$
$$\subseteq \{N \mid (1-w) \cdot g_p(n) \leq 2we + (1-w)k\} \quad \text{since } h_p(n) \geq 0$$
$$\therefore |S| \leq b^{2(w/(1-w))e+k}$$

Thus we have a bound on the number of nodes visited by HPA in the course of a search. In order to determine the actual computational complexity, however, we must know how many times a given node may be reopened. It is proved in [Munyer 1976] that when $w \leq \frac{1}{2}$ this number is bounded above by a constant depending only on w and e; from this follows the following:

Corollary 4 The computational complexity of HPA+ using $w \leq \frac{1}{2}$ and a heuristic function of bounded error e in a graph with branching rate b and shortest solution path of length k is at most $O(b^{2(w/1-w))e+k})$.

We now turn to HPA, which, unlike HPA+, cannot reopen closed nodes.

Theorem 5. For all b and k, there exists a graph with branching rate b and shortest solution path of length k in which HPA using $w \leq \frac{1}{2}$ and a heuristic function of bounded error e will expand more than $b^{w(2e-k)+(1-w)\ell}$ nodes, where $\ell = \lfloor (k-1)/2 \rfloor$. $((w/(1-w))(2e+1)+2)$ is the length of the path which is found.

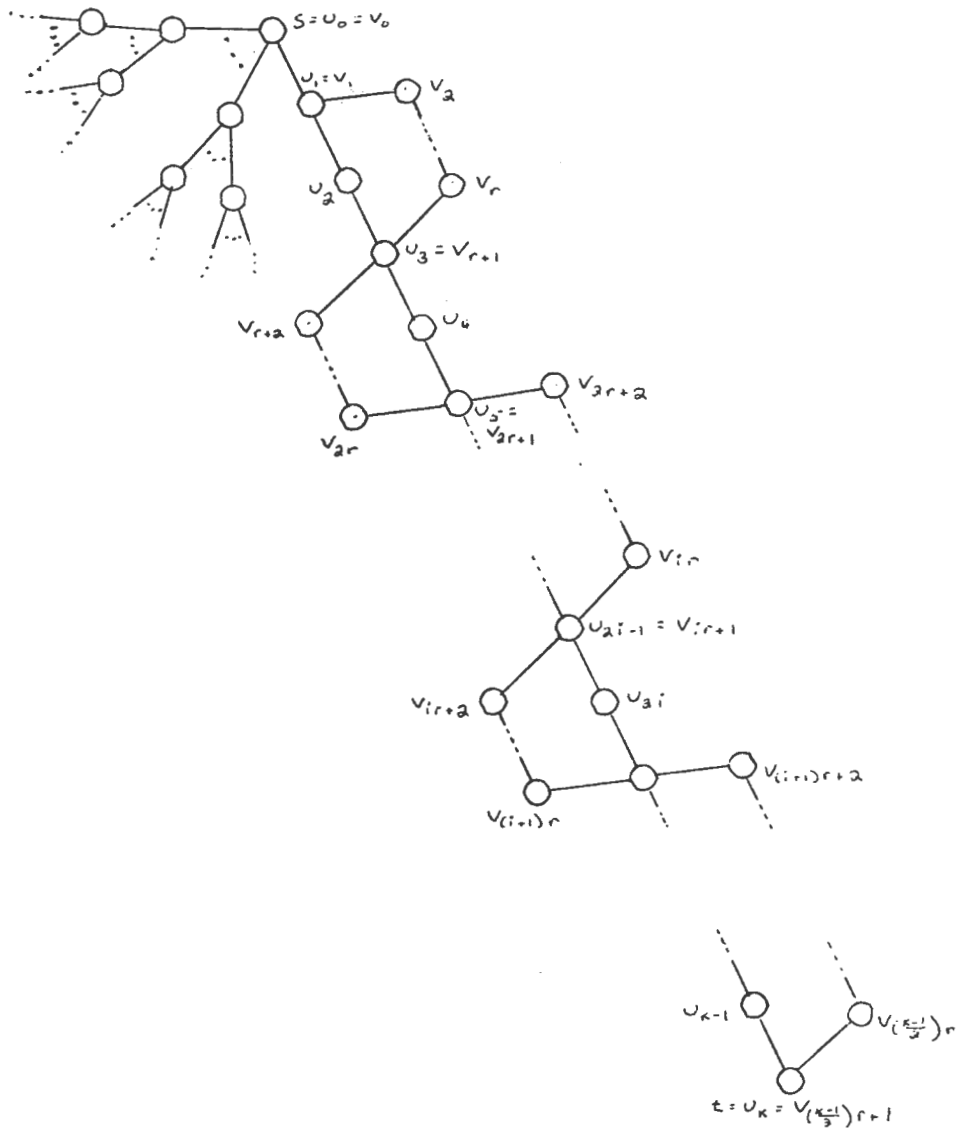


Figure 1

Proof A detailed proof is given in [Munyer 1976]; space permits only an outline here. The graph is shown in figure 1. The shortest solution path is $u_0 = s, u_1, \ldots, u_k = t$, and the longest solution path $v_0 = s, v_1, \ldots, v_\ell = t$ intersects the shortest path at every other node, $u_{2i+1} = v_{ir+1}$. In addition there is a full infinite tree of nodes rooted at s. We show first that it is possible for the longest path v to be found rather than the shorter path u when $r < (w/(1-w))2e + 1$, from which it will follow that the goal node t will have $f(t) = we+(1-w)\ell$, and finally that $b^{f(t)+w\cdot(e-k)}$ of the tree of nodes at s will be expanded.

The adversary uses the following heuristic function:

$h(u_{2i}) = h_p(u_{2i}) + e$ for $1 \le i \le k/2$

$h(t) = e$ for t the goal node

$h(N) = h(N) - e$ for all other nodes N.

With this heuristic function, the nodes $v_2, \ldots, v_r$ will be expanded before u_2, and u_3 will be expanded as a descendent of v_r with $g(u_3) = r$ instead of being expanded as a descendent of u_2 with $g(u_3) = 3$. Similarly, u_{2i+1} will be expanded as a descendent of $v_{(i+1)r}$ with $g(u_{2i+1}) = (i+1)r$. Thus the goal node t will be expanded with $g(t) = \ell = \frac{1}{2}(k-1)r$ if k is odd and $g(t) = \ell = \frac{1}{2}(k-4)r+1$ if k is even.

In the infinite tree of nodes rooted at s, all nodes N with $f(N) < f(t) = we+(1-w)\ell$ will be expanded before the goal node is expanded. There are $b^{w(2e-k)+(1-w)\ell}$ such nodes, and the theorem is proved.

Appendix: HPA model of heuristic search [Pohl 1970].

A problem space is a locally finite directed graph G.

$G:X = \{x_1, x_2, \ldots\}$, X is the set of nodes and can be infinite

$E = \{(x_i, x_j) | x_i, x_j \in X, x_j \in \Gamma(x_i)\}$, E is the set of edges and can be infinite if $|X|$ is infinite.

Γ is the successor map

$\Gamma:X \to 2^X$ where for all x, $|\Gamma(x)| \in N$

Heuristic Path Algorithm (HPA)

s = start node, t = terminal node, x = any node

$g:X \to N$, the number of edges from s to x enumerated by HPA—distance-to-date term

$h:X \to R^+$ (the nonnegative reals), an estimate of the number of edges on a shortest path from x to t—heuristic function

$f(x) = (1-w)g(x) + wh(x)$, $0 \le w \le 1$—evalutation function

S = set of nodes already visited and expanded

$\tilde{S}$ = set of nodes one edge removed from those in S, but not in S—candidate set

1. Place s in S and calculate $\Gamma(s)$, placing them in $\tilde{S}$. If $x \in \Gamma(s)$, then $g(x) = 1$ and $f(x) = (1-w) + wh(x)$.

2. Select $n \in \tilde{S}$ such that f(n) is a minimum.

3. Place n in S and $\Gamma(n)$ in $\tilde{S}$, discarding any nodes already in $S \cup \tilde{S}$. Calculate f for these new successors of n. If $x \in \Gamma(n)$, then $g(x) = 1 + g(n)$ and $f(x) = (1-w)g(x) + wh(x)$.

4. If n is the goal state, then halt, otherwise go to step 2.

HPA+ is a modification of HPA which like A^* may return nodes to $\tilde{S}$ the candidate set. This is important when desiring least costly solution paths. In effect, the shortest path back to s is kept. In general graph domains this is non-unique and may repeatedly force the recalculation of f(x).

HPA+ has step 3 of HPA replaced by:

3': Place n in S and check all $x \in \Gamma(n)$ for the following possibilities. If $x \in \Gamma(n) \cap \tilde{S}$ and the new value of f(x) is smaller than its old value, replace the old value by the new value. If $x \in \Gamma(n) \cap S$ and the new value of f(x) is smaller than its old value, remove x from S and place it in $\tilde{S}$. Otherwise place x in $\tilde{S}$.

Notes: In dynamic weighting w(x) replaces the constant w.

A heuristic function satisfies the monotone criterion if for all $x \in X$ and $y \in \Gamma(x)$,

$0 \leq h(x) - h(y) \leq C(x,y)$ and $h(t) = 0$.

$C(x,y)$ = cost of the single edge (x,y).

The monotone criterion is provably equivalent to the seemingly more stringest consistency condition of [Hart et al. 1968].

References

Aho, A. V., J. E. Hopcroft and J. D. Ullman [1974] The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass.

Camerini, P., Fratta, L., and Maffioli, F. [1973] A heuristically guided algorithm for the Traveling Salesman Problem, Politechnico di Milano, Memo 73-1.

Cook, S. A. [1971] The complexity of theorem proving procedures, Proc. 3rd Annual ACM Symposium on the Theory of Computing, pp 151-158.

Doran, J. and D. Michie [1966] Experiments with the Graph Traverser Program, Proc. Roy. Soc. A v 294, pp. 235-259.

Garey, M. R., R. L. Graham, and J. D. Ullman [1972], Worst-case analysis of memory allocation algorithms, Proceedings of the 4th Annual ACM Symposium on the Theory of Computing, pp. 143-150.

Harris, L. [1973] The Bandwidth heuristic search, Proc. IJCAI 3, pp. 23-29.

Hart, P., N. Nilsson and B. Raphael [1968], A formal basis for the heuristic determination of minimum cost paths, IEEE Transaction Sys. Sci. and Cyber. v 4, pp. 100-07.

Held, M. and R. Karp [1971] The Traveling Salesman Problem and Minimum Spanning Trees-Part II, Math. Prog. v. 1, p. 6-25.

Johnson, D. B. [1973] A note on Dijkstra's shortest path algorithm. JACM 20(3), pp. 385-388.

Karp, R. [1972] Reducibility among combinatorial problems, Complexity of Computer Computations, pp. 85-104 (eds. Miller, R. and Thatcher, J.), Plenum Press, N. Y.

Knuth, D. E. [1973] Sorting and Searching, Addison-Wesley, Reading, Mass.

Knuth, D. E. and R. W. Moore [1975] An analysis of alpha-beta pruning, Artificial Intelligence 4, pp. 293-326.

Kowalski, R. [1972] AND-OR graphs, theorem-proving graphs and bi-directional search, Machine Intelligence 7, pp. 167-94 (eds. Meltzer, B. and Michie, D.).

Martelli, A. and Montanari, U. [1975] From dynamic programming to search algorithms with functional costs, Proc. 4th Int. J. C. A. I., pp. 345-350, Tbilisi, USSR.

Martelli, A. [1975] On the complexity of admissable search algoriths, Report Institute di Elaborazione della Informazione del Consiglio Nazionale delle Richerche, Pisa, Italy.

Michie, D. and Ross, R. [1970] Experiments with the adaptive Graph Traverser, Machine Intelligence 5, pp. 281-300 (eds. Meltzer, B. and Michie, D.), Edinburgh University Press.

Munyer, J. [1976] Some results on the complexity of Heuristic search in graphs, UCSC Heuristic Theory Project Memo HP-76-2.

Nilsson, N. [1971] Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York.

Pohl, I. [1969] Bi-directional and Heuristic Search in Path Problems, SLAC Report 104, Stanford.

Pohl, I. [1970a] Heuristic search viewed as path finding in a graph, Artificial Intelligence v 1, pp. 193-204.

Pohl, I. [1970b] First results on the effect of error in heuristic search, Machine Intelligence 5, pp. 219-36 (eds. Meltzer, B. and Michie, D), Edinburgh University Press.

Pohl, I. [1973] New Results in the Theory of Heuristic Search, Proc. VII Int. Cybernetics Congress, Namur, Belgium, pp. 303-309.

Pohl, I. [1975], Practical and theoretical consideration in heuristic search algorithms, UCSC Heuristic Theory Project HP-75, NATO-ASI Conference on the Machine Representation of Knowledge, Santa Cruz.

Ross, R. [1973] Adaptive Aspects of Heuristic Search, Ph.D. Thesis, Edinburgh University.

Ross, R. [1975] Recent results on the efficiency of heuristic search, draft report, Machine Intelligence Research Unit, Edinburgh University.

Waidelich, M. [1973] Heuristic Search in Theorem Proving Systems, UCSC Heuristic Theory Project Memo H-4.

A Methodology for the Evaluation of Chess Playing Heuristics

Laszlo Sugar
University of Toronto

This paper presents a method whereby chess heuristics can be evaluated. It is based on statistical analysis of scored master games. A description of game selection and an absolute scoring strategy is given.

This paper outlines experiences we have had with the evaluation of chess-playing heuristics. The motivation for heuristics evaluation is familiar to anyone who has worked with game playing programs. It is often very difficult to judge the effectiveness of individual heuristics or to determine how a set of heuristics interact. Most often, trial and error assessment methods are used and changes to heuristics are based on intuition rather than quantitative analysis. This situation is mainly due to the lack of clear and measurable criteria for correctness of chess moves, and hence correctness of heuristic scoring.

We present a methodology that confronts these problems by experimentally determining chess heuristic performance. Briefly, the measures are based on the collection and analysis of heuristic scores over a set of selected master-level games. These scores are compared to an absolute scoring of the same games. Statistical analysis then quantifies the individual and collective effectiveness of the heuristics.

The main issues of such an evaluation scheme are concerned with the choice of the 'model' games and the development and application of an absolute scoring scale. We used a probabilistic measure of success for such a scoring scale, based on the Closeness to Win (CTW) [Horning72] chess-playing strategy.

In the sequel, we begin with a discussion of basic concepts concerning the use of heuristics in move selection and introduce the subject of chess program evaluation. Next, we present a description of an evaluation experiment we conducted using a chess program, CHUTE, developed at the University of Toronto [Valenti74]. Finally, we conclude with some general remarks about the results of the experiment and the viability of our methodology.

Heuristic Evaluation

The Scoring Polynomial

The heart of most chess programs is a move evaluation function which controls the generation of the game tree and determines the selection of moves. The usual form of the evaluation function is a scoring polynomial consisting of a linear combination of heuristic scores. Each heuristic assesses some aspect of the move (or position) that is under scrutiny.

The function of the scoring polynomial is predictive in nature. It tries to determine which moves will result in the best subsequent positions. The ideal situation would be if the ordering of moves determined by the polynomial were identical to the ordering determined by complete lookahead.

To be effective, the polynomial must contain a sufficient number of heuristics to measure all relevant aspects of a position. It is equally imperative that proper weightings be assigned to the heuristics in order to reflect the relative importance of the aspects.

One important quality of heuristics is what we call "completeness". In order to be complete, a heuristic must reflect through its score the changes in the particular aspect it is supposed to measure. As an example, one such aspect is the material balance of a position, defined as the difference in the number of pieces.

For a comprehensive description of chess program design methods, the reader is directed to [Newborn75]. The problems inherent in using a scoring polynomial for game tree search are discussed in [Marsland74].

Chess Program Evaluation

In order to evaluate a chess program, one of the things we need to measure is "correctness" of move selection. For a given position, we would like to compare the program's move to the ideal move. We can pick out the ideal move in two ways: examine a complete game tree, or have a master pick it out for us. Complete game tree examination is, of course, impractical.

The effect of master selection is achieved if we use positions that have occurred in master play. A master game is,

in effect, a nearly ideal path through a game tree. The problem of ensuring that this path leads to the optimal win is solved by our selection of games; we ensure that the play is close to optimal because masters are making the moves. Thus our assumption in using these games is that the winner tries to win, the loser tries to stall the loss and the quality of play ensures that "good" moves were selected to attain these respective goals. At the least, we are assuming that the quality of these games is better than that attained by any chess program to date.

To evaluate heuristic effectiveness, we also need an absolute move or position scoring method. This is necessary in order to compare the scores assigned by the scoring polynomial to the actual values of the master-selected moves.

The Game Library

The first step of our research involved the gathering of games for the game library. Ninety-eight games were selected as suitable and coded into machine readable form. The games were then further processed by CHUTE to produce a file of 4353 scored position/move records.

The games that were included in the library had to meet certain criteria. The calibre of the players had to be at the master or grandmaster level; hence most of the games were taken from the records of international tournament competition. One important criterion was that the game had to have a winner. In other words, drawn games were not used in the game file. This feature was necessary for the purposes of scoring the games. In short, the qualities the games had to meet were that the players should be very competent and the outcome of the game should be determined by superior play rather than by obvious blunders.

One of the common features of master games is that they often end in resignation. Since our scoring method assumes that the ending is a checkmate, the following convention was adopted to deal with the anomaly. If the game was resigned, then the winner was 'awarded' a checkmate 4 ply from the point of resignation if the number of moves so far in the game was greater than 30, and 6 ply if the number of moves was less than 30. If the game ended in checkmate, then no modifications were needed.

Our rationale behind this measure was that a resignation in the early part of the game would be due to a hopeless position, whereas near the end the loser could see an impending checkmate. This is a gross generalization, but some simple mechanism had to be developed that would differentiate resignations from checkmates without extensive analysis.

Some further pruning of the games had to be made in order to derive the final set of positions for the library. This selection was based on the presence of a '?' move in a game. A '?' move means that the master move was clearly not the optimal move for the position. The validity of our scoring method depended on the moves after a move being optimal. Thus, only the portion of a game that had no '?' moves after any given move was usable.

The CTW Scoring

As a basis for our ideal scoring method, we used the CTW game tree search strategy. The philosophy of this strategy differs from minimax in that search is directed by uncertainty rather than purely by absolute scoring. This means that the probability of fanout selection and terminal evaluation errors can be included in the considerations for pruning. A feature of this strategy is the introduction of a uniform scoring scale based on the closeness to the end of game, the 1/N score. (The reader is directed to [Horning72] for a complete description of this strategy.)

We applied the CTW scoring method to the positions in the library in the following way: an ideal CTW scoring polynomial would assign a "closeness to win" to a move in the range of -1(lose) to 0(draw) to +1(win). This same type of measure was used in the tree lookahead. From a given position, if a move resulted in a win in N ply, then the value of the move was 1/N, if a loss in N ply then -1/N. Naturally, the game tree may contain a terminal win/loss at more than one level, but the ideal score was obtained by noting at which level a win/loss is inevitable given that the winner will make moves to achieve this win and the loser will make moves to avoid a loss as long as possible.

The CTW scoring scheme is "ply sensitive". This means that the backed up scores will reflect the depth of back-up as well as

the value of back-up. Since both heuristic and back-up scoring is on the same scale, we can expect a much more homogeneous scoring system.

Analysis of Data

The scoring of the game library provided us with the necessary raw data for further analysis. We now had, for each position, the ideal move and its ideal score. We also had information on how CHUTE ranked the ideal move in relation to its own choices. The following analyses that we performed helped to condense and summarize this data.

Ranking

A ranking measure similar to the one employed by Samuel [Samuel63] and [Samuel67] was used to measure both the original effectiveness of CHUTE and the effectiveness of the derived results. A measure of the predictive power of the scoring polynomial is how it ranks moves as compared to lookahead ranking. If we assume that the master's move is the best move in any position, then we can measure how the polynomial ranks the best move. The ranking can give us the values of the best move omission error probability as a function of fanout.

The actual measure was calculated as follows: The seven best moves, as picked by CHUTE for each position, were examined. The relative rank of the master move was then noted and the results tallied to produce the following set of measures:

P1 P2 P3 P4 P5 P6 P7 P8

where Pi, for $1 \le i \ge 7$, represents the percentage of cases examined where the master's move was ranked within the first i moves, and P8 represents the percentage of cases where the master's move was not in the top seven moves. We could interpret 1-Pi as the probability of omitting the master move from the tree for a fanout of i.

We applied this measure to the game library four different times. Each time we varied the portion of the file that we used. This was done in order to see how well CHUTE did in different parts of a game.

Heuristic Weightings

The purpose of the heuristic weighting experiment was to provide us with an evaluation of the scope of the scoring polynomial. In particular, we wanted to measure the polynomial in terms of an absolute scoring scale. We also wanted to determine if the set of heuristics in the polynomial was complete and to derive a better set of weightings for the heuristics. We performed multiple regression analysis of the heuristic scores to the ideal 1/N score in order to suggest the new weightings.

The master's move was scored using the 1/N method. This score should have been derivable as the result of the scoring polynomial. Hence the ideal 1/N score was regarded as the dependent variable and the heuristic scores were regarded as the independent variables. Using a large number of such variable n-tuples, namely the data on the position file, multiple regression analysis produced a linear combination of the independent variables. This estimated the dependent variable with minimum mean square error. This linear combination, or polynomial, was then used to predict values of the dependent variable. The predictive equation we derived was:

score= w0*h0 + w1*h1 + ... + w21*h21 + c + r

where wi are the weightings, hi are the heuristic scores (there were 22 heuristics in CHUTE), c is a constant, and r is the residual error. The residual has mean zero and the smallest standard deviation possible for any linear combination of the heuristic scores. Hence for this set of heuristics, over this collection of data, the above polynomial provided the optimal linear prediction function. (For the actual values we derived for wi, see [Sugar76].)

The value of regression is in its ability to assess the overall effectiveness of the original weightings. The weightings suggested by the regression can be compared to the original weightings and used to determine how well the original polynomial predicts the linear relationship of the heuristics to the CTW score. The new weightings can also be interpreted as a measure of heuristic effectiveness. A high score means that the

heuristic is a good predictor of CTW, while a low or negative score means that it is a poor predictor.

The residual error (r^2) measure gives us an upper bound on the expected reliability of the heuristic set used in the polynomial. This points out any possible weaknesses in the heuristics themselves as well as the possibility of non-linear relationships of the heuristics to the CTW score.

The weightings that are derived and the effectiveness of the new polynomial are highly dependent on the success of the original regression fit. This is measured in terms of the percentage of the variance of the dependent variable that the polynomial accounts. If this measure is low to begin with, then the new polynomial can't be expected to be very effective.

Results

Our results have shown that the methods used in polynomial and heuristic evaluation were effective tools in confirming and isolating problems. In general, this means that such measures can be valuable tools in the process of writing and improving chess programs. In the specific case, the results have served the purpose of evaluating CHUTE and shedding light on the feasibility of the CTW strategy.

One result significant for CHUTE was the discovery that we could exclude certain heuristics from the scoring polynomial without lowering the polynomial's ranking effectiveness. This suggests deep problems in the reliability of the heuristics that were excluded. More importantly, however, it was the regression analysis that predicted which heuristics could be excluded. This is a confirmation of the accuracy of the correlation to CTW score as a measure of individual heuristic effectiveness.

The rankings derived over the whole file allowed us to calculate some reasonable estimations of the probability of fanout selection and terminal evaluation errors in CHUTE. The probability of not including the master move in the tree at the first level for a fanout of 7 (the value used by CHUTE in the 1974 ACM tournament) was .25. The probability of the scoring polynomial not ranking the master move as the best move was found to be .69. (See Figure 1 for more detail).

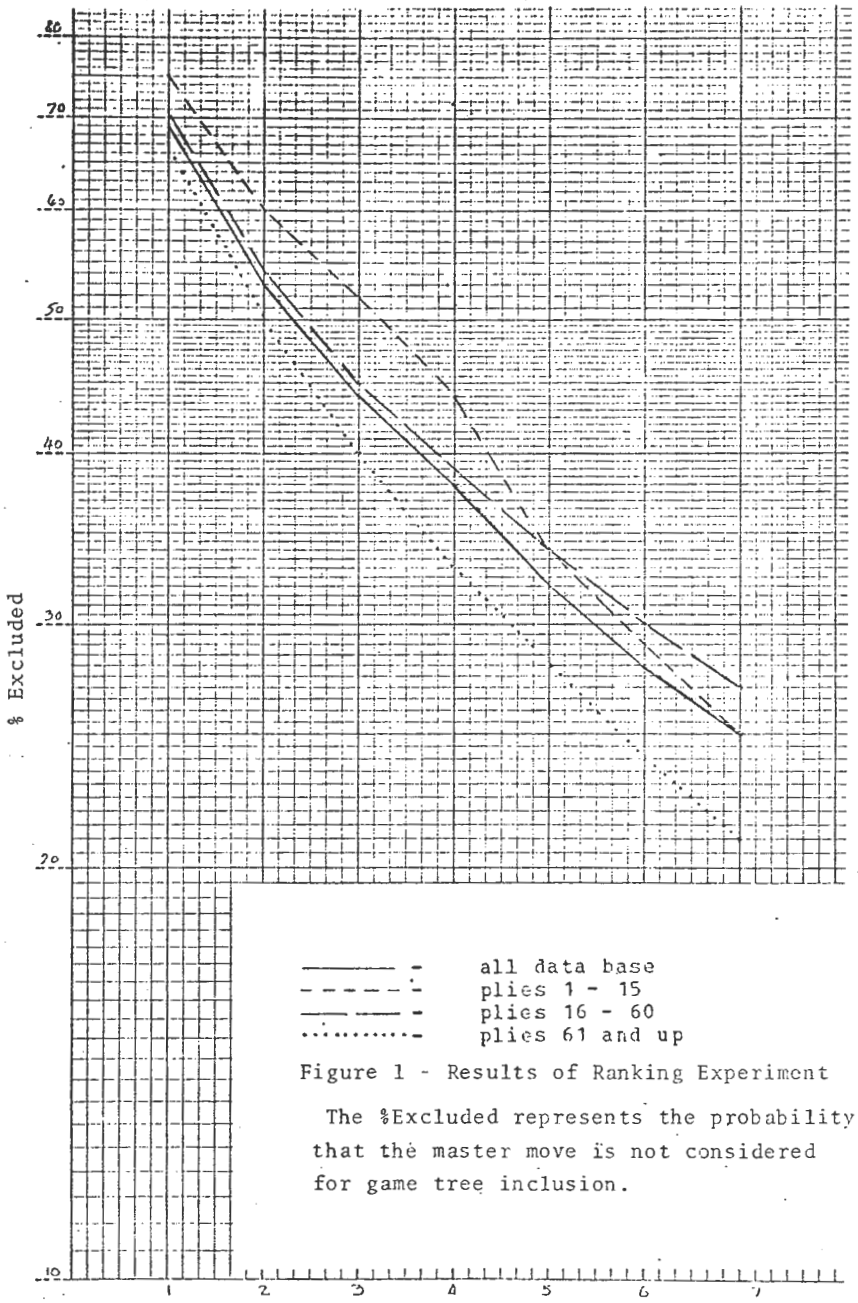


Figure 1 - Results of Ranking Experiment

The %Excluded represents the probability that the master move is not considered for game tree inclusion.

Weaknesses and Limitations of the Method

The methods we have outlined have certain shortcomings. The most obvious of these is our inability to assign an all-encompassing score for program effectiveness. While we have evaluated the components, we have not established an all-encompassing scale on which chess programs could be compared.

There is the problem of finding a uniform move/position scoring scale. While we believe that the 1/N scale is appropriate, we have no conclusive evidence to support this belief. It is quite possible that other scales (e.g., $1/N^2$, $1/\log(N)$) would have been better.

Finally, there is the question of the game library. One possible criticism that we can see is that the type of positions we used was not indicative of computer chess play. We have not included a lot of possible positions that chess masters never encounter due to the high quality of master play. Thus the reliability of the regression results may not be as good as we would want. The inclusion of non-master positions may produce better results.

There is further the possibility that the regression over master moves is simply a measure of the heuristics's ability to detect winning positions "after the fact", rather than their ability to predict winning moves. Thus regression may only be useful for determining heuristic effectiveness as a terminal node evaluator, rather than as a pruning tool.

Conclusions

Our aim in this research was to develop and test heuristic evaluation measures. These measures proved to be helpful in evaluating the effectiveness of CHUTE. We also explored the problem of implementing the CTW strategy, but without any conclusive results. The viability of the CTW strategy is still an open question. (A strategy similar to CTW, based on depth in the look-ahead tree, has been successfully employed by Nievergelt [Nievergelt74] for the Shannon switching games)

Acknowledgements

I would like to thank Professor J. J. Horning for his expert guidance during the research and for his valuable comments on this paper. I am also grateful to Professor Z. G. Vranesic for our fruitful discussions about the research and to Mike Valenti for allowing me the use of CHUTE. James Allen and Dave Thompson provided appreciated comments during the preparation of this paper. Financial support was gratefully received from the National Research Council of Canada.

References

[Horning72]

Horning, James J.; 'Outline of a Strategy For a Chess Playing Program, 72sep09'; unpublished, University of Toronto, Dept of Computer Science, 1972. (see Appendix 1 of [Sugar75])

[Marsland74]

Marsland, T.A., Rushton, F.G.; 'A Study of Techniques for Game-Playing Programs', in Advances in Cybernetics, 1974.

[Newborn75]

Newborn, M.; Computer Chess; Academic Press, New York, 1975.

[Nievergelt74]

Nievergelt, J., Farrar, J.C., Reingold, E.M.; Computer Approaches to Mathematical Problems; Prentice Hall, New Jersey, 1974.

[Samuel63]

Samuel, A.I.; 'Some Studies in Machine Learning Using the Game of Checkers'; in Computers and Thought ; McGraw-Hill, 1963, pp. 71-105.

[Samuel67]

Samuel, A.I.; 'Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress '; IBM Journal of Research and Development 6, 1967, pp. 601-617.

[Sugar76]

Sugar, L.; 'An Experimental Evaluation of Chess Playing Heuristics'; Tech. Report CSRG-63, University of Toronto, 1976.

[Valenti74]

Valenti, F.M.; 'CHUTE1, An Easily Modifiable Chess Playing Program '; M.A.Sc. Thesis, University of Toronto, October 1974.

THE 'THINKING MACHINE' ARRIVES BUT ONLY AS A CHILD

Richard S. Rosenberg
Department of Computer Science
University of British Columbia
Vancouver, B. C., Canada

Abstract

As part of an ongoing study of A.I. and the popular media, we wish to explore the popularity of the child analogy theme in the reporting of research on intelligent machines. This theme argues for the supposed and obvious analogy between the computer as a learning machine on the road to intelligence and the child learning to be an intelligent adult. Three examples of such machines, the Perceptron, the Cybertron and Cynthia appeared on the public scene between the years 1958 and 1963.

1. Introduction

If the early 1950's were witness to the arrival of an electro-mechanical species of so-called turtles, tortoises, rats, etc., subsequent years have been noteworthy for a series of claims and counterclaims on the issue of 'thinking machines'. The modern electronic digital computer had been developed during the second World War for military purposes and when the war ended, its immediate application to business, science, and industry became apparent. What was also apparent was that the computer itself provided a unique mechanism for the study of a wide range of heretofore theoretical questions, characterized by a version of the mind/body problem; namely, "Can machines think?".

The very arrival of the computer, before it had exhibited the ability to do anything but perform arithmetical operations at very high speed, brought this question to the fore. For the first time, or so it seemed, there existed a machine which could carry out activities previously restricted to humans. Therefore, it is not altogether surprising that it was seized upon as technology's current model for the human brain.

The general purpose of this paper is to serve as one component of an analysis of how and to what effect ongoing research in A.I. has been presented to the general public via the popular media. More specifically, we wish to focus on

the supposed analogy between the computer as a learning machine on the road to intelligence and the child learning to be an intelligent adult. This theme is reflected in many of the claims made for intelligent machines as these are reported in the popular media.

As we examine a representative sampling of magazine and newspaper articles, a number of points will become clear. Among these are a general uncritical reporting of accomplishments and claims, a frequent drift into anthropomorphism, and a devaluing of human skills and genius in the face of computer achievements. We will be concerned mainly with that most important of human skills, learning. Two learning machines, the achievements of which were widely hailed, are presented. Also included are two other examples of the 'child syndrome'.

2. Learning

The most frequently enunciated argument against the possibility of developing intelligent machines is that computers do only what they have been programmed to do, whereas humans if nothing else are characterized by their ability to adapt to novel situations. With this in mind, a considerable effort has been mounted by researchers to investigate the nature of machine learning. In some cases, the learning is done within a fairly narrow domain, such as checkers, the purpose being to develop the learning techniques as far as possible with hopes that they can be generalized. The alternative approach is to study learning in a broad environment in order to develop a general learning machine able to cope with a wide variety of situations.

This latter approach is sometimes characterized as cybernetics or referred to as the study of self-organizing systems. During the 1940's there were four scientific developments which were mutually reinforcing. In neurophysiology, the nature of nerve conduction was being uncovered. Part of the model of the brain being developed, pictured the cerebral cortex as composed of a vast network

of neurons with a multitude of interconnections.

In 1943, a very important paper by Warren S. McCulloch and Walter H. Pitts of M.I.T.[1] appeared. Based on developments in neurophysiology, McCulloch and Pitts were able to analyze mathematically some characteristics of networks composed of such idealized elements: "Because of the "all-or-none" character of nervous activity, neural events and relations among them can be treated by means of propositional logic."[2]

In Norbert Wiener's book, Cybernetics,[3] published in 1948, he outlined experiments and mathematical studies carried out over several years which convinced him and others that similar techniques could be used to analyze the behavior of both living and artificial systems. The central notion here is feedback; in attempting to achieve some goal, a system utilizes information from the environment to indicate how close it is and how forces under its control should be activated.

The last strand in this story is the development of the electronic computer during World War II, and its increasing availability for scientific research in the 1950's. Researchers in self-organizing or adaptive systems were motivated to use the computer to model networks of neuron-like elements. The basic belief was that these networks could learn to recognize patterns by a program of trial and error governed by cybernetic principles.

Through such a training process it was hoped (and believed) that the network would eventually distinguish among a large class of patterns with a low error rate. If such a system worked, it would be a general learning system applicable to arbitrary patterns. Such patterns could be visual, aural, tactile, etc. Since the network is in some sense a model of the human brain, its behavior might reveal some features of the human brain itself.

So researchers could claim that the study of nerve networks was a legitimate branch of neurophysiology as well as the initiation of a new field of in-

quiry, that of self-organizing or cybernetic systems whose aim was the development of intelligent machines.

Two learning machines, Perceptron and the Cybertron, will now be discussed. The Perceptron is the prime example of a research project which owes its existence to the self-organizing principle. The New York Times of July 8, 1958[4] reported this work. The second caption for the story is of some interest:

> Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser.[5]

Of interest also is the fact that it is a branch of the military which has a direct concern in the matter, as the opening paragraph reveals.

> The Navy revealed the embryo of an electronic computer today that that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.[6]

How is the public supposed to interpret the phrase "embryo of an electronic computer"? Is it some kind of baby computer related to a full blown computer as a human infant is to an adult? The lay reader is left to his own devices but the implication is clear.

The designer of the Perceptron, Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory is not very reticent about his creation,

> ... the machine would be the first device to think as the human brain. As do human brains, the Perceptron will make mistakes at first, but will grow wiser as it gains experience[7]

but then neither is the Navy:

> ... the Perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control." ... Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language.[8]

They expect the first Perceptron to be finished in about a year, cost $100,000 and be able to read and write. Although no information is provided as to how Perceptrons work, the reading and writing one "will have about 1,000

electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells."[9] This compares with the human brain's estimated ten billion neurons and 100 million connections with the eyes. A skeptic might wonder about a device which could read and write using only about 1000 neuron-like elements. If he were somewhat more informed he might also raise questions about the numbers game.

In a subsequent article on Perceptrons in the Sunday New York Times,[10] additional information is provided.

> Navy officers demonstrating a preliminary form of the device in Washington said they hesitated to call it a machine because it is so much like a "human being without life."[11]

Any comment would be superfluous. In a demonstration of Perceptron-principles, an IBM 704 computer was used to simulate a Perceptron in order to provide an example of its learning behavior. Apparently the computer was able to learn to distinguish right from left for a number of squares situated at random either on the left or right side of a field. To this, Dr. Rosenblatt commented, "... after having seen only thirty to forty squares the device had learned to recognize the difference between right and left, almost the way a child learns."[12] This latter remark was totally irresponsible. Neither then nor now is there any adequate theory of how a child learns this difference. One additional outrageous claim before we turn to Cybertrons.

> Only one more step of development, a difficult step ... is needed for the device to hear speech in one language and instantly translate it to speech or writing in another language.[13]

Without much evidence, Dr. Rosenblatt would have us believe that in the very near future (as of 1958) all human skills will be reproducible by Perceptrons. Either a gross underestimation of these human skills or an equivalent overestimation of Perceptrons (or both) was operative, to say nothing of the ever present need for overoptimism to maintain research support. The usefulness of comparing a learning machine to a child, whose growth and maturation are obvious, cannot be under-

estimated as a public relations ploy.

The arrival of the Cybertron from the Raytheon Company was hailed almost simultaneously in 1961 by the New York Times, the Wall Street Journal, and the Christian Science Monitor with the following headlines:

> Robot Machine Learns by Error
> New Device Solves Problems That Have No Formulas[14]

> Raytheon Device Learns From Experience, Can Find Subs, Tell Bad Berries From Good[15]

> Cybertron 'Learns' As It Hums[16]

Well, what has been achieved and what is promised for the Cybertron? Of two models under development, the smaller "... is working on military problems of an undisclosed nature under a contract with the Department of Defense"[17]while the larger, "... is being developed to recognize speech sounds".[18]

The Christian Science Monitor provides some of the most amusing images.

> If there is a "happy ending" to the Frankenstein legend, the Raytheon Company's new "learning machine" may provide it. ... The new Cybertron ... exhibits many of the precocious characteristics of a small boy. ... Somewhat like the small boy's mentality which it resembles, the Cybertron performs "chores" and problems but does not draw conclusions.[19]

If I were a small boy I would be quite insulted. The reference to the Frankenstein myth is interesting because of the suggestion that an intelligent machine may turn out to be beneficial and not destructive, especially of its creator. The projected uses for Cybertron machines range from the mundane to the exotic:

> The machine learned in a few hours the technique of separating real from false target echoes picked up by an electronic sounding device. ... The machine can read and interpret cardiograms ... and can be taught to do such things as sort bad strawberries from good ones.[20]

> ... would be used in work such as the analysis of radar data, the sorting of industrial or agricultural products and as a help in weather forecasting.[21]

Pure fantasy. No intelligent machines are at present used for any of these tasks. Furthermore, claims that the machine could be taught to identify radar images in a few hours whereas human operators required several months

sound more like an advertising campaign for the defense dollar.

3. A Machine Which Listens and Learns

In October 1963, the Christian Science Monitor carried an Associated Press story on Mr. Arnold Lesti's Invention, "'Think Machine' Obeys".[22] We might have expected the word, thinking, but certain important considerations such as space probably prevailed. Mr. Lesti's invention is called Cynthia:

> "Now open, Sesame," he instructed.
> And 4 year-old Cynthia obligingly opened the door to a conference room.
> "Open now, Sesame."
> The door remained closed.
> "She's learned her lesson well," Mr. Lesti grinned. "She's been taught to open the door when I give the correct command."[23]

Cynthia, a synthetic intelligence machine, is the third generation of a 1959 prototype of "a machine that seems able to think for itself".[24] Cynthia is about six feet tall, two feet wide and was developed at a cost of $400,000. What are Cynthia's abilities?

> Cynthia is able to receive voice instructions effectively.
>
> "The machine has the internal capacity to correct its own errors."
>
> "Cynthia has the ability to recognize and understand concepts, so that when the machine sees something it never saw before or hears something it never heard before, it is able to tell into what concept or category the new instance fits and come up with the correct answer."[25]
>
> These are presumably its present skills but the future is limitless:
>
> ... fourth or fifth generation Cynthias will have the ability to read typed, printed, or written matter and be able to understand the meaning of sentences, paragraphs, pages, and books, either written or spoken.
>
> Only slightly more sophistication is needed before we will have a machine that can carry on an intelligent conversation or solve the most complicated problem. The day is not too far off when we'll be able to mass produce scientists and engineers.[26]

Mr. Lesti is particularly interesting because he formed his own company, Andromeda, Inc., of Kensington, Maryland, in 1957 in order to study the nature of intel-

ligence. Is Mr. Lesti a lonely genius? As he says, "People thought we were crazy when we began working on this".[27] And as the article notes, both the National Aeronautics and Space Administration and the Department of Defense "have expressed considerable interest in Cynthia". Whether or not this means financial support we are not told. I am also not aware of any recent generations of the Cynthia dynasty.

4. Graphing and Touching

Much of the experimentation on mechanical hands or arms has emphasized either master-slave systems used to handle dangerous substances such as radioactive isotopes or arm-like equipment for performing rather simple tasks as spot-welding on an industrial assembly line. Our main interest here is to examine mechanical arms in the context of their relation to intelligent machines. Clearly, for a robot to be human-like it must have means for interacting with its environment in a non-trivial fashion. The Wall Street Journal reported on one such development in 1962 in a short article with the caption, "Groping Mechanical Arm Is Like An Infant's, Learns Through Touch".[28]

> In somewhat more detail, we are told that MIT,
>
> ... says that it has developed a mechanical hand and arm, linked to a computer that can explore the world about itself like a small child in a dark room.
>
> The hand gropes slowly a half-inch off the floor in an area about five feet square. It can locate a box, explore it by touch to determine its size, then find blocks and put them in the box.
>
> The arm and hand apparatus is similar to that used by researchers to manipulate radio-active materials by remote control. But in the MIT device, the hand has been fitted with 30 "sense organs" to orient position and detect pressure, such as from encountering a block.[29]

What is it that makes this computer-controlled mechanical arm distinctive?

> ... it adapts to the unexpected much like an infant.
> ... the system is capable of understanding its environment
> ... because it is capable of correlating its program - in

> the computer - with the problem it faces. [It] forms possibly the first artificial creature that can deal with the outside world and have a limited understanding of it.[30]

The one example given of the arm encountering a problem is that if a board is placed in the way of the exploring arm it will feel its way around the board. The reader will recall similar claims for dealing with environments made over 10 years earlier by the creators of the mechanical tortoises and mice. However, this is certainly a step along the path of equipping computers with manipulative devices with which they can alter their environment. Thus, our interest in this report is not about the quality of the research, but rather about the unwarrented comparison to the behavior of a child.

5. Final Comments

This brief sampling of newspaper articles from the years 1958 to 1963 indicates the attractiveness of the child analogy theme, especially for the popular press. Somehow the image of the computer as a maturing infant is a subject with irresistable appeal. Since the newspapers are quite willing to print fairly outrageous claims without counter-balancing opinion, such stories appear quite frequently. More than any other kind of reporting, the cataloguing of either unsupported results of the promising of early, major successes, conditions the public to a willingness to believe in or at least a general acceptance of the imminent arrival of walking, talking, thinking machines. If such machines are so relatively easy to construct what does that imply about the value of human abilities?

Acknowledgement

The support of the National Research Council under research grant NRC A-552 is gratefully acknowledged.

Footnotes

1. McCulloch, Warren S., and Pitts, Walter H., "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathem. Biophys., Vol. 5 (Chicago; University of Chicago Press, (1943), pp. 115-133.
2. Ibid. p. 115
3. Wiener, Norbert, Cybernetics, New York, J. Wiley, 1948.
4. "New Navy Device Learns By Doing," The New York Times, Tuesday, July 8, 1958, p. 25.
5. Ibid.
6. Ibid.
7. Ibid.
8. Ibid.
9. Ibid.
10. "Electron 'Brain' Teaches Itself," The New York Times, Sunday, July 13, 1958, Section IV, p. 9.
11. Ibid.
12. Ibid.
13. Ibid.
14. Fenton, John H., "A Robot Machine Learns By Error," The New York Times, Tuesday, Aug. 15, 1961, p. 61.
15. "Raytheon Device Learns From Experience, Can Find Subs, Tell Bad Berries From Good," The Wall Street Journal, Tuesday, Aug. 15, 1961, p. 8.
16. Hughes, Albert D., "Cybertron 'Learns' As It Hums," The Christian Science Monitor, Thursday, Aug. 17, 1961, p. 4.
17. Fenton, op. cit.
18. Ibid.
19. Hughes, op. cit.
20. The Wall Street Journal, Tues., Aug. 15, 1961, p. 8.
21. Fenton, op. cit.
22. "'Think Machine' Obeys," The Christian Science Monitor, Oct. 24, 1963, p. 12.
23. Ibid.
24. Ibid.
25. Ibid.
26. Ibid.
27. Ibid.
28. "Groping Mechanical Arm Is Like an Infant's, Learns Through Touch," The Wall Street Journal, Jan. 4, 1962, p. 7.
29. Ibid.
30. Ibid.

Descriptive and Command Schemata: Knowledge Representations for Answering a Questionnaire

Rainer von Konigslow, Queen's University

Abstract

In this paper, I argue for an integration of computational and empirical approaches to the study of human intelligence. I propose that empiricist principles and criteria be applied when evaluating what a computational model tells us about human cognitive processes. The issues are discussed in reference to ALGERNON, a language-comprehension programme which simulates how subjects deal with impression formation and person evaluation tasks.

The present paper is not about specific implementation techniques for representing knowledge. Rather, it addresses the problem of deciding what features of an implementation we wish to assert as analogous to human cognitive processes. Computational models utilize tree structures and propositional representations, and we may be persuaded by computational success to assert that such information structures describe humans, i.e., we would assert an analogy between the computational representation and cognitive structures in individuals. Besides the computational approach, however, there is a second tradition which bears on what we can assert about knowledge representation. It is empirically oriented, and is concerned with making testable predictions about actual human performance and with contriving laboratory situations which reveal why subjects behave as they

do, i.e. which isolate particular cognitive processes.

Empiricists assert that a theory cannot be true if it does not allow us to predict actual performance. Most computational models do not simulate either the actual behaviour of specific individuals or the average behaviour of a population. Nor do they display full competence, i.e., they do not simulate or allow discrimination of the full range of acceptable behaviour. Rather, they produce behaviour which is minimally competent, behaviour which falls within the confidence limits of observable behaviour from the population.

Without tests through prediction, however, empiricists would not give credence to any claims of analogy between computational models and human cognition. I suggest that the empiricist criteria can be satisfied if we build computational models which simulate psychological laboratory experiments. The laboratory situation, with its carefully contrived and controlled tasks, constitutes a good mini-world for a model. Further, I believe that current natural-language computational techniques permit us to simulate how subjects behave in laboratory situations and thus permit us to make and to test assertions about how people "really work".

I have been working on a process-oriented model for a well known laboratory task, impression formation. The model is incorporated in a LISP programme, ALGERNON, and is based on the approach suggested here (von Konigslow, 1974).

Answering a questionnaire

A typical experiment in person evaluation involves a questionnaire which asks subjects to evaluate a fictitious person. When entering the laboratory, the subject receives

instructions and then the questionnaire. A scale alternative is marked to record the evaluation, the only overt response on the questionnaire. A questionnaire item might be, for example:

"Bill is industrious and intelligent.

like 1 2 3 4 5 6 7 dislike."

Past accounts have proposed mathematical models to "explain" how the information from more than one trait adjective is integrated into an overall evaluation, (Rosenberg, 1968) As an account of cognitive processes, mathematical models are not sufficient. They do not explain how the descriptive sentences are interpreted to form an impression, how the evaluation is translated into a scale choice, or how the instructions "select" the relevant cognitive processes. In short, such models are not sufficiently complete.

In impression formation and person evaluation tasks, almost all of the information is conveyed in language. The experimenter tells the subject what to do, and how to do it, and provides descriptions of persons. The subject deals with the information and generates responses according to the instructions. ALGERNON, therefore, is based on language comprehension techniques for: (a) understanding and acting on the instructions, (b) interpreting the descriptions, and (c) generating the appropriate responses.

Before proceeding, let me demonstrate ALGERNON in a simple evaluation task. The output of ALGERNON is indicated by a period in the left margin; lines prefixed with an asterisk contain output followed by input (underlined in this paper). ALGERNON prints a question mark prefix if it expects a sentence, and a dollar sign for scales. It also prints a dollar sign underneath the chosen scale alternative.

?Take instructions.

. Please enter the instructions for task1:

?Imagine students.

?Read an assertion.

?Evaluate the individual.

?Repeat.

?Quit.

* Do you want me to execute this immediately? yes

. This completes the instructions for task1.

ALGERNON interprets input according to expectancies. At the beginning, it expects sentence input (indicated by the question mark). It acts under the implicit command "Cooperate" (with the experimenter), its normal question-answering mode. It also expects commands to tell it what to do. The command changes the expectancy, it invokes a schema which interprets subsequent commands as descriptions of a task, e.g., task1. It assumes that such commands are not to be executed at present but are to be assembled into a knowledge representation of the task which can both be executed and paraphrased. In other words, ALGERNON's representation of the task is both descriptive and imperative. (The parser produces a predicate calculus representation for sentences, which can be integrated into a data structure and which can be executed as a LISP-like function.) The predicates in the commands must point to function or task definitions as part of their lexical entries. The interpretation of the noun phrases as either modifiers or references depend on the function definition for the predicates. References are not resolved until the task is executed. "Quit!" is ambiguous in this context; it may be part of the task being defined.

```
?Execute task1.
?Bill is intelligent.
$like  1 2 3 4 5 6 7  dislike
.       $
?Bill is lazy.
$like  1 2 3 4 5 6 7  dislike
.              $
 Bill is intelligent and lazy.
$like  1 2 3 4 5 6 7  dislike
.           $
?Quit.
.   This statements is not an ASSERTION,
.      it is a IMPERATIVE.
* Should I proceed with it? yes
.   End of the task.
```

In executing the task, ALGERNON enters another context. Both what to do and what to expect is determined by its knowledge about the task. The first command (Imagine students) indicates that, while doing the task, references to persons should not be interpreted as references to individuals it knows about. ALGERNON must make up new descriptive nodes containing only the knowledge that they are students. It then expects to read an assertion (indicated by the question mark). Descriptive sentences about persons lead to changes in knowledge, i.e., impression formation. A schema is invoked which integrates the information in the description with the previous knowledge about that individual. Attitudes and prejudices may lead to further inferences (cf, Asch, 1948). ALGERNON can detect inconsistencies, deal with redundancy, and make inferences.

The evaluation has several components; ALGERNON expects to read a scale, with the end-terms indicating an evaluative dimension. Second, it will use the impression to evaluate the descriptive elements separately, the trait terms included in the impression. Third, it integrates the information into an overall evaluation. Fourth, it chooses one of the scale alternatives. Such an evaluative procedure is implicit in previous models, though only the third step was formalized. Other procedures are under investigation (von Konigslow, 1976).

The example above illustrates how ALGERNON can make predictions about actual human performance. Except for the simplified instructions, it answers the same questionnaire given to human subjects. ALGERNON can be used either to predict average performance or to match a specific individual. In either case, it needs "tuning", i.e., it has to be provided with the attitudinal and evaluative information the subject is assumed to bring to the laboratory. For predicting average responses, ALGERNON can be tuned with population norms. For matching a particular individual, a split session or within-subject design is used. The subject is given a set of items containing single trait term descriptions, and his responses are used to tune the model. The model predicts the responses to further items contain more than one descriptive term. The two approaches are also used to test the mathematical models referred to above.

Representing Knowledge

Given that ALGERNON can make good predictions about human performance, what kind of claim can we make about the correspondence between the information structures or processes of the model and those of human subjects. The empirical tradition

is relatively silent on this issue; it is primarily concerned with the behavioral correspondence. If a model does not predict, it is modified or rejected. Models which do predict are not shown to be true, but they do receive support from the prediction. Note that only those elements of the model which are functionally involved in the prediction may receive support from the success of the prediction.

None of these principles are particularly helpful when considering what aspects of ALGERNON's construction or operation to assert as analogous to human cognition. All of the parsing and language comprehension machinery of ALGERNON is used with every questionnaire item and, therefore, is functionally involved in the prediction. To disregard the details and the implementation specifics, we could claim support for the class of functionally equivalent models. However, there is, at present, no computational equivalence theory, no "reduced machine" for AI programmes. Also, specification of an equivalent reduced finite state machine would not be a satisfactory "explanation" of the cognitive processes of humans. We not only have to decide what claims to make but also how to express these claims.

On the assumption that human cognitive processes must be determined by neurologically "hard-wired" functions and by stored cognitive structures, analogous to stored instruction computers, we can restrict ourselves to claims about structures, i.e., information representations which contain knowledge. A claim about the relation between subject's knowledge about the task and ALGERNON's relevant function definitions is one example. A claim about the correspondence between a subject's impression of the individuals to be represented and the data structures in ALGERNON

represents a second. The first example illustrates claims about command schemata while the second considers descriptive schemata.

The most general kind of claim concerns partitioning the information structures by function; there must be an information structure which represents the knowledge the subject has gained from the instructions. In the demonstration of ALGERNON, we made similar claims for the impression. As other examples from ALGERNON, there are information structures for: (a) evaluative information associated with personality descriptions, (b) lexical information for parsing, (c) attitudes and prejudices (implemented as antecedent theorems), and (d) conditional tasks, represented by conditional imperatives such as "If the lecturer is warm and friendly, talk to him." (simulating the results of Kelley, 1950).

Some of the preceeding claims may be so obvious that they are accepted by default. In general, however, such claims need support through evidence and argument. Let us look at function decomposition, a further example of a partitioning claim. At one level of analysis, function decomposition is represented through the commands of our simplified instructions, "Imagine students", "Read an assertion", etc. Partitioning is explicit, and can receive support either from a demonstration that actual instructions given to subjects are organized in a similar manner, i.e., with equivalent content, or by giving such instructions to subjects and by getting equivalent results. More direct. i.e., behavioral, evidence can be obtained if we compare the reports when we ask both the model and the subjects to paraphrase their instructions and compare the reports. Consider the paraphrase of instructions produced by ALGERNON for Task1.

?Describe task1!

. TASK1 IS A TASK.

. Label: START-OF-TASK

. Execute the IMPERATIVE: IMAGINE ALL STUDENTS.

. Execute the IMPERATIVE: READ SOME ASSERTIONS.

. Execute the IMPERATIVE: EVALUATE ALL INDIVIDUALS.

. Start again from label: START-OF-TASK

. Label: END-OF-TASK

. Execute the IMPERATIVE: UNIMAGINE ALL STUDENTS.

No subject produces a paraphrase of the form above. However, we want to make a claim only about the general type or purpose of a process and about its relative ordering; we do not need most of the information in reports from subjects. We can reduce the information by devising scoring procedures to be applied to both the output of the model and the reports of the subjects.

Another, somewhat similar, approach to function decomposition can be based on protocol statements. For instance, we claimed that reading a description led to forming an impression, and that the evaluation involved identifying the value dimension, evaluating the items in the impression, integrating the values into an overall evaluation, and choosing a response alternative from the scale. Consider a sample protocol from ALGERON.

?Verbalize evalutions!

. Yes sir.

?Execute task1!

?Bill is intelligent and lazy

. The impression on which the evaluation is based:

. BILL IS A PERSON AND A STUDENT.

. BILL IS INTELLIGENT AND LAZY.

$pleasant 1 2 3 4 5 6 7 unpleasant

. The terms "PLEASANT" and "UNPLEASANT" suggest that

. the evaluation should be based on the dimension

. "AFFECTIVE".

. Based on the predicate(s): INTELLIGENT, LAZY

. and on the variable(s): STUDENT

. the evaluation on the dimension "AFFECTIVE" is: 0.50

. I shall assume that the SEVEN response alternatives

. divide the dimension into categories of equal width.

. I therefore choose the scale alternative: "4",

. the FOURTH from the left, the high-valued end.

Note that ALGERNON does not separate the evaluation of single items from the integration of their values. While we have not yet formally collected and scored protocols, none of the exploratory protocols made such a distinction. But, the traditional information integration approach is based on the distinction. Also, the equal interval assumption was not mentioned in the exploratory protocols.

So far we have focussed on partitioning claims. A second type of claim focuses on the content of information structures. Because content represents differentiable elements in a structure, it is similar to a partitioning claim, except that the focus is on identifying and naming the constituents. Again, we can get direct data by asking subjects to describe the relevant information. The task description is illustrated above, and the protocols include the impression, a description of the information about the person.

In conclusion, I have argued that empirical criteria should be used in evaluating claims about human cognitive structures and

processes and have illustrated both the kind of claims which might be made and the type of support which might be adduced for them. Most of the claims are meta-theoretical, i.e., they are not claims about the details of a particular model but rather are about general features of information structures and processes. Space does not permit discussion of several interesting issues, e.g., process control, event structures, or the implications of the present approach for traditional psychological theory. It is hoped that the present approach will assist the uneasy marriage of the "odd couple", the computational and the empirical traditions and thus encourage development of a genuine cognitive science.

References

Asch, S.E. Forming impressions of personality. J.Abn.Soc.Psychol. 41,(1946),258-290

Kelley, H.H. The warm-cold variable in first impressions of persons. J.Pers. 18,(1950),431-439

Rosenberg, S. Mathematical models of social behavior. In E. Lindzey & E. Aronson (Eds.) The Handbook of Social Psychology (Vol.1) Addison-Wesley, Reading, 1968.

von Konigslow, R. A Bayesian inference model for evaluative judgements in impression formation. Read at Can. Psychol. Assoc., Toronto, 1976

von Konigslow, R. A cognitive process model of person evaluation and impression formation based on a computer simulation of natural language processing. Natural Language Studies No.19, U. of Michigan, Phonetics Laboratory, 1974

RECENT PROGRESS IN THE ESSEX FORTRAN CODING SHEETS PROJECT.

R. Bornat, J.M. Brady and B.J. Wielinga,

University of Essex.

This paper describes recent progress made in building a program to interpret Fortran coding sheets using several sources of knowledge. Currently the program consists of three parts: a sheetfinding program, a segmentation and character reading program, and a program which reasons about Fortran from the "blob structure" of the coding sheet. To date, all these programs are more or less operational and some results are presented. The emphasis of the project lies currently on the interaction between various parts of the program, in particular the interaction between the Fortran reasoner and the character program. Some preliminary ideas on this interaction will be discussed.

Introduction

The FORTRAN coding-sheet project at the University of Essex is an attempt to show the effectiveness of the use of knowledge in a visual perception task, rather than the mere possibility of employing knowledge. It is for this reason that we have chosen such a well-trodden topic - that of reading a casually hand-printed FORTRAN program like that in figure 1. The topic has already been studied from the AI point of view, notably by Munson, Duda and Hart (Munson 1968) (Duda and Hart, 1968) and of course much work has been expended on upper-case hand-printed character recognition in the absence of knowledge of the text being read. Further justification of our approach may be found in Bornat and Brady (1976a) and work up to the beginning of this year is described in Bornat (1976), Bornat and Brady (1976b), Brady and Wielinga (1976a) - this paper reports on progress in our work since then.

The effectiveness of knowledge in visual perception has to derive from redundancy in the visual scene. Perceiving one part of the scene and knowing something about what the scene contains enables us to predict something about another part or at least gives us constraints on its future interpretation. FORTRAN in particular is enormously redundant syntactically. Upper-case handprinting, on a sheet with ruled lines and 'blips' which form a sort of clock-track, is highly constrained. Writers try to distinguish similar characters but don't follow a template - there are variations in size, regularity of spacing and so on.

The program was originally conceived as falling into two sections - one using knowledge about FORTRAN, the other about characters and writing. It is intended to be a collection of intercommunicating processes, with the output being plausible interpretation of the program on the sheet. Due to our terror when first faced with the sheer size of our input (one sheet is digitised to 12M bits or about 300K PDP-10 words) we added a preprocessor. Figure 2 shows the organisation of our program. All the separate parts exist

University of Essex
Computing Centre

TITLE SAMPLE — FORTRAN CODING SHEET — SHEET 1

PROGRAMMER Mike Brady — DATE / /

C STATEMENT NUMBER	CONT	FORTRAN STATEMENT	IDENTIFICATION AND SEQUENCE No
C		PLEASE FILL UP ONE LINE WITH LOTS OF CHARACTERS..BUT BE TOO NEAT!	
		WRITE(5,18)	
18		FORMAT(' ATHIS PROGRAM SUMS, RECIPROCALS AND THEIR SQUARES)	
20		WRITE(5,15)	
15		FORMAT(' ?TYPE N:')	
		SUM = 0	
		SUMSQ = 0	
		READ(5,30)N	
30		FORMAT(I2)	
		IF(N.GE.0) STOP	
		DO 40 I = 1,N	
		SUM = SUM + 1./I	
40		SUMSQ = SUMSQ + 1./(I*I)	
		WRITE(5,50)N,SUM,SUMSQ	
50		FORMAT(' SUMS ',I2,2E15.4)	
		GOTO 20	
		END	

Form No. C.C.10

Figure 1

(some more developed than others, of course) and work is just starting on the real meat of the project - developing a dialogue between the 'character experts' and the "FORTRAN reasoner'.

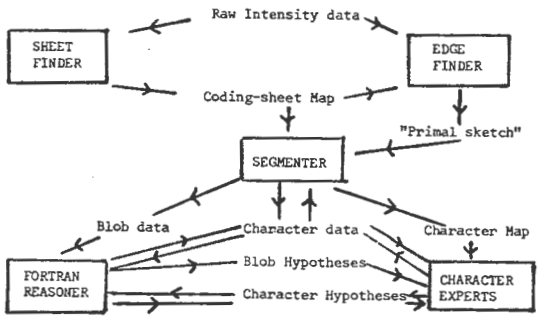


Figure 2

Data Collection is via a 35mm negative, photographically enlarged into a 175*125mm positive, digitised to 256 light-levels on a photodensitometer. As part of the project we've had to build our own 'vision system' - an interpreter (Bornat and Wielinga 1976) picture I-O routines an indexed database and a 'frame' system.

The Coding-sheet finder

One of the most obvious tasks in our project was to find where to look. We had the idea of taking a 'long-distance view of the sheet, with sufficient resolution to see blobs of writing and perhaps the ruled lines but insufficient to see details of the individual characters. The program is reported in Bornat and Brady (1976b). It works on reduced-resolution data - a 4*4 reduction gives us a manageable 20K of PDP-10 words.

The original motivation for the program was to produce a 'blob map' which would be the first input to the FORTRAN reasoner. Now that we are more experienced in low-level 'vision hacking', we find that we can get better information, collected in a more satisfactory fashion, from the Segmenter (see below), so this part of the program has been relegated to the task of telling the Segmenter where the lines are, what parts of each line seem to be completely blank, and give an estimate of the inter-blip gap. We utilise the fact that the lines are long, straight, parallel and periodic to indicate where we may have missed a line or interpreted some writing as part of the line.

Edge detection and segmentation

Given the outline of the coding sheet as produced by the sheet finder, an area of the sheet corresponding to one line is selected and read into memory. The grey-level data are transformed to gradient space using a 3 x 3 gradient operator (Roberts, 1963) and thresholded. Feature points with a similar gradient direction (quantised to 8 different values) are grouped together into edges. This process results in a representation of the writing on the line in terms of a set of edge elements, similar to a "Primal Sketch" as proposed by Marr (1976).

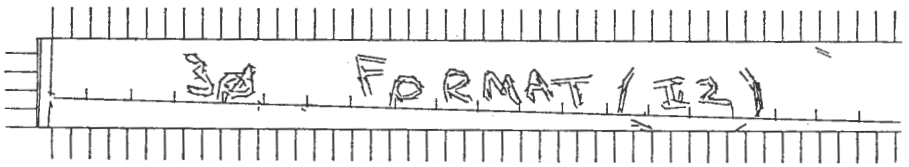


Figure 3

Figure 3 shows an example of such a primal sketch. We decided to use the primal sketch rather than intensity or gradient data as the basic input for successive stages of the program for a number of reasons. Obviously, the edge data (thresholded on length) are a lot cleaner than the raw data. Secondly, identification of the coding sheet lines and blips is easier for an entire line than for a much smaller character area. A third reason for using the primal sketch is that the segmentation process can be based on stroke (edge) information rather than on some sort of intensity histogram, as was the case in an earlier version of the program (Brady and Wielinga, 1976). Another advantage of the use of a primal sketch is that during the segmentation process the program can have a "quick look" at the character area to determine rough size and shape information and to do some statistics on the strokes present in the area. This information can be used to classify the character roughly as being "roundish", "straightish", a decender (possibly a bracket) or as an operator (in general smaller than alphanumeric characters). A last reason to introduce the primal sketch is detection of curves. Curves can easily be detected (and described) as a set of small, partially overlapping edge elements, for example the "O" and "R" in figure 3.

The information gathered in the segmentation stage (blob data and tentative character information) is sent to the Fortran reasoner and a dialogue between segmenter, character-reader and Fortran reasoner is initiated. It should be stressed that the output from the segmenter is not always reliable. It is possible that "noise strokes" (e.g. scratches or dirt on the original sheet, or strokes that are part of the coding-sheet lines or blips, but which are not identified as such) are interpreted as punctuation marks or as operators. Descender information, and in general, size information, is not reliable in cases where segmentation between characters is difficult. Equals signs are often not small enough to be identified as operators. These problems can often be overcome in a dialogue between the segmenter and the Fortran-reasoning program as described below.

Reasoning about Fortran

The task of a FORTRAN reasoner in our program is to exploit consistency between information about different parts of the sheet, based on knowledge about the FORTRAN programming language. There are two obvious ways to do this: 1) Bottom-up: as if a human, reading the sheet, came upon the realisation that it was FORTRAN. 2) Top-down: knowing that it is FORTRAN, attempting to impose a structure on it.

The bottom-up solution is like trying to find the best-fit from a universe of interpretations, given some partial information about the data. The top-down solution is more immediately approachable, and is plausible as an explanation of the way we read difficult handwriting, searching for an explanation of the confused marks in front of us. Although humans don't often have to use this method when reading our data, it is a mode of behaviour worth investigating

which may cast light on the organisation of processes in other visior tasks.

Most work on the reasoner up to date has assumed that the coding-sheet finder would provide 'blob' data like that shown in figure 4. The reasoner is told the length and position of blobs, whether they are 'operator' blobs (including punctuation and equals signs) or 'alphabetic' blobs. Its task is to guess statement identities given this information, and to indulge in a dialogue with the character and writing processes in the program, both inviting and providing information about the data.

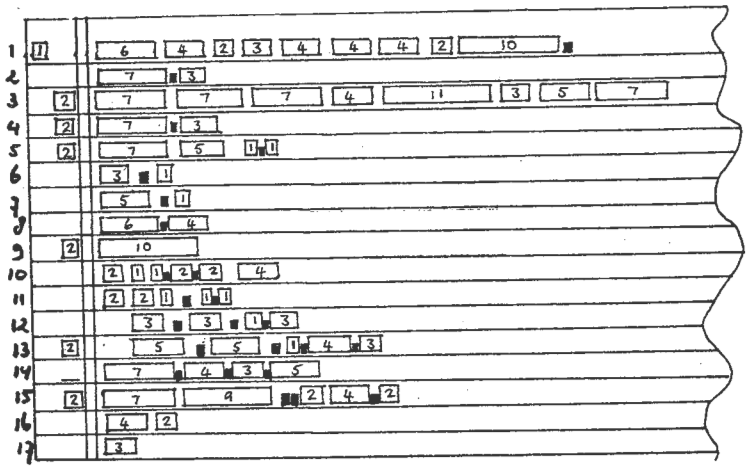


Figure 4

The part of the reasoner which guesses statement identities is implemented. Working rather like a top-down parser, it attempts to assign roles to the blobs on each line, simply working through the possibilities in turn. It assumes at present that this is unreasonable for consecutive alphabetic blobs to run together into one blob, reasonable if two blobs are separated by a parenthesis or if an alphabetic blob is followed by a numeric blob. It assumes initially that 'words' are never split into separate blobs (except for 'GO' 'TO'). With these simple assumptions it produces the following first guesses for the lines shown above:

1. comment | FORMAT
2. WRITE | READ
3. FORMAT
4. FORMAT | WRITE | READ

5. FORMAT
6. assignment
7. assignment
8. assignment | READ
9. BACKSPACE | FORMAT | REWIND | PAUSE
10. logical-IF
11. DO | arithmetic-IF
12. assignment
13. assignment
14. WRITE | READ
15. FORMAT
16. CALL | GOTO | REAL | STOP
17. END

Figure 5

In figure 5, the correct guess is underlined. It is surprising that such a simple algorithm, using such simple knowledge, can arrive at such a performance, often guessing correctly and always including the correct guess among the few proferred. The performance is sustained with other data - though if you know about FORTRAN syntax it would be trivial to construct an example to confuse it. We'll be happy if our system works on non-pathological examples at first, though later it will of course be necessary to be able to 'take back' early guesses and then, we hope, it will be able to handle programs which aren't written as clearly as this one.

The program is at present being developed to produce a graph which shows each statement's role in the control flow. The most obvious use of this is to divide declarations from statements, thus rejecting the 'REAL' guess on line 16, for example. Most inter-statement knowledge relies on control-flow information so the graph is essential for us to move away from reasoning about single statements. It makes some apparently bizarre inferences more plausible - such as the one which runs 'line 17 is END, line 16 isn't a comment or a FORMAT, therefore line 16 must be RETURN, STOP or GOTO'. In early versions of the program we were so impressed by the power of knowledge about the END line that it looked at the last line first of all, knowing it to be END, and then at the line above, knowing it to be RETURN, STOP or GOTO. If it might be RETURN, then these last lines formed part of a subprogram, and therefore ... ! Later we rejected this as too ridiculous and made the program look at the sheet from top to bottom. Now it will have to produce this inference as a natural result of reasoning from the control graph - the last node in a program unit can't let control 'drop through' to the END line.

All of the knowledge used so far, however, and all of that envisaged in the near future, is about the syntax of FORTRAN. This reliance on syntactical knowledge is a strength of the project - although an understanding of the program's purpose would enable us to make much more powerful inferences and employ much more powerful constraints, such an understanding is beyond the state-of-the-art. The knowledge so far incorporated enables us to cut down

the search space of the rest of the system enormously, and often enables us to propose single character 'acid tests' to distinguish between different interpretations of a line. This avoids many simple errors which an unknowledgeable system might make - for example, Duda and Hart (1968) after filtering the output of a character recogniser, had a line interpreted as D7 11 I=1, 100 - the obvious interpretation in blob terms is that it is a DO statement, so the possibility of the second character as '7' would never arise in the first place.

A difficulty with incremental simulation (Rovner, Nash-Webber and Woods, 1974) is that the associated modules may fail to meet their original specification. This has happened with the FORTRAN reasoner. The data shown in figure 4 are unrealistic. In some ways they're too accurate - the segmenter may provide unreliable information and in others they're too undifferentiated - the segmenter can provide information on many individual characters within the blobs. A true dialogue involves helping the character experts with their problems as well as spontaneously offering interpretations.

Character reading using partial knowledge

Once the Fortran reasoner has made a first guess at the identity of a statement (or has decided that no reasonable guess can be made just on the basis of the blob information) a dialogue between the Fortran reasoner, the segmenter and the character programs is initiated. This dialogue may take the form of simple requests to the character system like "verify an F", with a straightforward yes or no answer. In cases where difficulties arise, either in the Fortran reasoning or in the character reading process, more elaborate dialogues may occur: Fortran: "I think this statement is READ, WRITE or FORMAT; can you discriminate?"; charactersystem: "No I can't read it, but the 5th character could be a bracket, does that help?"; Fortran: "Yes, I'm now quite sure it is READ(...), could you verify?"; charactersystem: "Yes, it could very well be READ(...)".

The consequence of this rich interaction is that the character system has to be able to adapt its behaviour according to the requested information and to the partial information it is provided with. Moreover it must also be able to "be aware" of its own reasons why it believes certain evidence. This is because questions of confidence may arise, e.g. when a hypothesis made by the Fortran Reasoner strongly conflicts with character evidence, the character reader has to be able to contemplate the structures it has built, and possibly reconsider its interpretation of the evidence. Even internally a conflict may arise during the reading of a character, e.g. in the case of difficult segmentation or ligatures: "it looks like O but I have an unexplained stroke".

The requirements sketched above have strong implications on the structure of the character system and on the way in which knowledge about characters is represented: (1) the control structure must be flexible: the program must be able to change its strategy; (2) character knowledge should be packaged, in such a way that partial information can be represented and properly used; (3) the program must be able to assign roles to individual pieces of evidence within the

character models.

To meet these requirements we decided to implement a system based on "frames" (Minsky, 1975). In Brady and Wielinga (1976a) we elaborate further on the considerations which lead us to choose a frame-type representation for character knowledge, and give more details about our current implementation, which was inspired by an early version of KRL (Bobrow & Winograd, 1976). Frames as we have implemented them are information structures containing knowledge both in declarative and procedural form. Two important types of components in a frame are SLOTS and ACTIONS. SLOTS name and describe pieces of information in a frame, while ACTIONS describe procedurally how to use (or to obtain) information in the frame, and what to do when certain conditions in a frame are fulfilled. Figure 6 shows a typical frame representing a model for 'V'.

```
[V isa LETTER  with.slots

  LEFTSTROKE  ← [* isa STROKE  with.slots
                   SLOPE  ← <anyof LEFTDIAGONAL VERTICAL>
                   POS  ← LEFT]

  RIGHTSTROKE  ← [* isa STROKE with.slots
                   SLOPE  ← <anyof  RIGHT DIAGONAL VERTICAL>
                   POS  ← RIGHT]

  INTLR      ← [* isa INTERSECTION with.slots
                  STROKE1  ← ! LEFTSTROKE
                  STROKE2  ← ! RIGHTSTROKE
                  RELANGLE  ← ACUTE
                  POS ←  BOTTOM ]

with.actions
[when.filled  <allof  LEFTSTROKE RIGHTSTROKED]
 [$( test verified (INTLR) then confirm (V)
     or test converge.at.bottom(LEFTSTROKE, RIGHTSTROKE)
          then test check.touch(LEFTSTROKE, RIGHTSTROKE)
               then confirm (V) <> possible (U)
               or deny (V) <> suggest (U)
        or deny (V) <> suggest ([AHU])
 $)]

[when.filled  INTLR]  [confirm (V)]
[before.confirmed]
[ $( test distance (endpoint(RIGHTSTROKE),
                    intersect point (RIGHTSTROKE, LEFTSTROKE))>DELTA
     then possible (V) <> transformto(Y) <> verify(Y)
   or test smallvertical.stroke (right)
      then deny (V) <> transformto(U)<> verify(U)
   or handle.troublesome evidence ()<>
      resultis TRUE
 $)]
]
```

Figure 6

The frame contains descriptions for the two strokes (SLOTS: LEFTSTROKE and RIGHTSTROKE) and for the intersection between them (SLOT : INTLR). The actions describe what to do when appropriate strokes have been found: certain checks have to be made to make sure that 'V' is indeed the right character and not 'U' or 'Y'.

To illustrate the working of our current character system, we will describe the way in which the program behaves when confronted with the character in figure 7, having no partial information on the identity of the character. The program starts with a bottom-up search for big strokes in the primal sketch. Two strokes will be found. A database search for applicable frames (i.e. frames containing slots for two diagonal or vertical strokes, one at the left, one at the right) will return a number of character frames (e.g. A, H, U, V, Y. ...) and some frames which describe stroke relations like INTERSECTION and VCOMBINE.

Figure 7

The system currently uses 'hard-compiled' knowledge to decide which type of frame is the best candidate to try first - in bottom-up mode stroke relations.

VCOMBINE - a stroke relation which checks whether two strokes are part of one bigger stroke - is tried, refuted and proposes to try INTERSECTION. This frame is hypothesised and confirmed, and its slots RELANGLE and POS are filled with ACUTE and BOTTOM respectively. When the INRERSECTION frame is confirmed, the list of candidate character models is checked for models which match this type of INTERSECTION. The model for V is hypothesised and its slots are filled, invoking as a side effect the whenfilled action. Since the INTERSECTION is already verified, the V frame will be confirmed, and the before.confirmed action will check whether the distance between the intersection point and the end point of the right stroke is larger than a certain threshold DELTA. Since this is the case, the information in the V frame will be mapped onto a Y frame. The character system concludes that the character is likely to be 'Y', but that 'V' is still an alternative possibility.

Future work - the Dialogue

Now that we have got to grips with the parts of the problem, we will concentrate our efforts on the interaction between the various sections. Space does not allow us to show examples, but we have some simulated dialogues about lines of the coding sheet in figure 1. Line 8 ('READ(5,30)N') for instance is quickly identified as 'READ(an,nn)a' (an = alphanumeric, n = numeric, a = alpha), using a discrimination test on the comma (the alternative possibility - an assignment statement - requires an equals sign).

In the case of line 9 ('FORMAT(I2)') of the same sheet the blob information is less conclusive: there are four different types of statements possible, and moreover the segmenter has some difficulty in distinguishing the first bracket from 'I'. So, more evidence about the individual characters is needed to find a reliable hypothesis about the identity of the statement. It is precisely this kind of problem that our research in the near future will focus on.

References

Bobrow D.G. & Winograd, T. (1976) 'An overview of KRL, a knowledge representation language', to appear in Journal of Cognitive Science, 1976.

Recent progress in the Essex Fortran coding sheets project.

Bornat, R. 'Reasoning about hand printed Fortran programs', Proc. of the AISB Summer Conference, Edinburgh, 1976.

Bornat, R and Brady, J.M. (1976) 'Using knowledge in the computer interpretation of FORTRAN coding sheets', Int. J. Man-Machine Studies, 8, 13.

Bornat R and Brady, J.M. (1976) 'Finding blobs of writing in FORTRAN coding sheets projects', Proc. of the AISB Summer Conference, Edinburgh, 1976.

Bornat, R. and Wielinga B.J. (1976) 'The EVIL programming system', University of Essex Computing Centre memo (in preparation).

Brady, J.M. and Wielinga, B. J. (1976) 'Seeing a pattern as a character', Proc. of the AISB Summer Conference, Edinburgh, 1976.

Duda, R.O. and Hart, P (1968) 'Experiments in the recognition of hand-printed text-II', Proc. FJCC, 1139-51.

Marr, D. (1976) "Analysing natural images', AI memo 334, MIT AI Lab, 1976.

Munson, J.H. (1968) 'Experiments on the recognition of hand-printed text-I', Proc. FJCC, 1125-39.

Minsky, M. (1975) 'A framework for representing knowledge', in P. Winston (Ed.), The Psychology of Computer Vision, New York, McGraw-Hill, 1975.

Roberts, L.G. (1963) 'Machine perception of three-dimensional solids, opto and electro optical information processing', Tippett, J.T. et al (Eds.), Cambridge, MIT Press, 159-197.

Rovner, P., Nash-Webber, B. and Woods, W. (1974) 'Control concepts in a speech understanding system', Proc. IEEE symposium on speech recognition, 1974.

THE AXIOMATISATION OF STRIPS AS A PREDICATE CALCULUS PROGRAM

Donald Kuehner

Department of Computer Science

University of Western Ontario

London, Ontario

Abstract

It has been shown by Kowalski and van Emden that predicate calculus can be treated as a programming language. The axiomatisation of a problem is interpreted by a resolution theorem-prover as a program for the solution of the problem. Certain symbol manipulating algorithms can be very concisely stated as predicate calculus programs. An example is STRIPS, the robot planning algorithm of Fikes and Nilsson. STRIPS can be stated using eight axioms, so that an eight-line program is the result. A stronger version of STRIPS, Warren's WARPLAN, can be written as a twenty-line program.

Predicate calculus as a programming language

Recently van Emden [2,3] and Kowalski [6,7] have been considering the use of first-order predicate calculus as a programming language. The axiomatisation of a problem, when converted to the clausal form of resolution theory [9], can be considered to be a program for the solution of the problem. The resolution theorem-prover PROLOG [1], has been used as an interpreter for programs written in predicate calculus.

The logical statement $A \Leftarrow B\&C$, has clausal form $Av\sim Bv\sim C$.

As a programming procedure, this is written as +A-B-C. The procedure call, -A, is responded to by the procedure whose name is +A. The body of this procedure contains the procedure calls -B and -C. The unification of resolution becomes the identification of the parameters of the calling statement with the dummy parameters of the procedure.

Two examples

It is easy to construct LISP-type lists using nestings of the function CONS, and the empty list NIL. The two-element list [A,B] is represented by CONS(A,CONS(B,NIL)). Thus x is a list if x = NIL or if there exist y and z such that x = CONS(y,z). This is equivalent to the procedures

+ISLIST(x) -IS(x,NIL)

+ISLIST(x) -IS(x,CONS(y,z)).

These procedures form a program for testing whether or not x is a list.

The following recursive procedures construct a new list by appending the second list onto the end of the first list. Capital letters are used for constant values, and small letters are used for variables.

(a) +APPEND(NIL,list2,list2)

(b) +APPEND(CONS(head1,tail 1), list2, CONS(head1,newtail))
-APPEND(tail 1, list2, newtail)

Procedure (a) states that if the first list is the empty list, then the new list is the same as the second list. Procedure (b) states that the head of the new list is the head of the first list, and that the tail of the new list is constructed by appending the second list to the tail of the first list.

These procedures could be called by the following main program

(c) -APPEND(CONS(A,NIL), CONS(B,CONS(C,NIL)), newlist)
-OUTPUT (newlist).

When these three clauses are submitted to a resolution theorem-proving program, clause (c) is distinguished as the set of support [8,11]. When (c) and (b) resolve, the resolvent is

(1) -APPEND(NIL,CONS(B,CONS(C,NIL)), newtail)
-OUTPUT(CONS(A, newtail)).

The left-most literal of (1) can be unified with (a) to produce

(2) OUTPUT(CONS(A,CONS(B,CONS(C,NIL)))).

This clause can be thought of as resolving with the clause (e) +OUTPUT(x) which has the side effect of printing the value of x.

A proof procedure for executing programs

A predicate calculus program is usually written using Horn clauses. These clauses have at most one positive literal. Most Horn clauses are either procedures of the form $+A-B_1 \ldots B_n$, or assertions of the form +A. There is also the negated goal of the form $-B_1 \ldots -B_n$ and the terminal clause which is empty. It is easy to see that the resolvent obtained from two Horn clauses is itself a Horn clause.

An efficient inference rule for doing resolution with Horn clauses is Selective Negative Linear (SNL) resolution [8]. SNL is selective in that it chooses one literal of a clause to resolve on, and must not resolve on any other literal until that literal has been used. It is negative because its support set is negative and every resolvent must be negative. A resolution is linear if one parent of each resolvent is an input clause.

The search strategy selects the left-most literal of the

support clause or a resolvent. When attempting to resolve on such a clause, the input clauses are tried in the order in which they are written. When an input clause is found which does resolve, no lower clauses are tried unless that branch of the search fails. This is depth-first search.

In general a depth-first search is not exhaustive, and so the proof procedure is not complete. However, there is some indication that program termination may be assured by carefully ordering the clauses within the program, and the literals within each clause.

The need for an extended predicate calculus

Literals with side effects such as OUTPUT(x) are provided chiefly for the convenience of the user. This corresponds to Green's answer predicate [5].

Certain semi-logical tests seem to require a special mechanism. Sometimes the truth of an essention can be tested within predicate calculus, but the testing of its negation cannot. For example, the procedure which tests whether x is a list, would also succeed if x were a variable. To test whether x is an explicit list, a +NONVAR(x) procedure must be written.

The following use of the special-purpose literal NOBRANCH allows the testing of negation.

(a) +NONVAR(x) -UNIFY(x,CONSTANT) -NOBRANCH -FAIL

(b) +NONVAR(x)

(c) +UNIFY(y,y)

(d) +NOBRANCH {has search strategy side effect}.

Assume that node n of a search tree has label -NONVAR(variable).

This could resolve with (a) or (b). The search strategy will first try (a). This succeeds, producing node n+1 labelled -UNIFY(variable,CONSTANT) - NOBRANCH -FAIL. This resolves with (c) producing node n+2 labelled -NOBRANCH -FAIL. The following resolution is with +NOBRANCH which as a side effect directs the search strategy to allow no further branching from the node above the one where -NOBRANCH first appeared, namely the node n. There is no +FAIL among the input clauses, so this branch of the search fails. The search would normally backtrack to node n and resolve -NONVAR(variable) with (b). But this is forbidden, so the search must backtrack further.

If node n had been labelled with -NONVAR(A), then node n+1 would have been labelled -UNIFY(A,CONSTANT) -NOBRANCH -FAIL. This would fail to unify, so the search would backtrack to node n and resolve successfully with (b). Thus (b), which always unifies, is accessible only if (a) fails at -UNIFY(x,CONSTANT).

The axiomatisation of STRIPS

Certain symbol-manipulating algorithms can be stated very concisely as predicate calculus programs. Fikes and Nilsson [4] describe an algorithm STRIPS which a robot can use to make plans. The program for implementing STRIPS in predicate calculus, PC-STRIPS, can be written as eight clauses. This economical program was suggested when modifying Warren's WARPLAN [10], which appears in the last section.

In order to understand the PC-STRIPS program, it is convenient to look at an example of the sort of data upon which it will operate. This data, expressed as Horn clause assertions, describes the initial world and the actions with which the robot can change this world.

Any action by the robot changes the state of its world.

The ADD predicates list the new situations which hold in the world after the action. The DEL predicates list the old situations which must be deleted. The PRE predicate states the conjunction of preconditions which must be present in the world before the action can be begun.

(D1) +GIVEN (ATROBOT(A))

(D2) +GIVEN (AT(BOX,B))

(D3) +ADD (ATROBOT(place2), MOVE(place1,place2))

(D4) +PRE (ATROBOT(place1), MOVE(place1,place2))

(D5) +DEL (ATROBOT(place1), MOVE(place1,place2))

(D6) +ADD (AT(object,place2), PUSH(object,place1,place2))

(D7) +ADD (ATROBOT(place2), PUSH(object,place1,place2))

(D8) +PRE(AT(object,place1)&ATROBOT(place1),
PUSH(object,place1,place2))

(D9) +DEL (AT(object,place1), PUSH(object,place1,place2))

(D10) +DEL (ATROBOT(place1), PUSH(object,place1,place2))

A simple task, expressed as a negated goal, might be

(G) -SOLVE(AT(BOX,C), START, plan) -OUTPUT(plan).

A conjunction of three goals written as goal1&goal2&goal3 represents the function CONJ(goal1,CONJ(goal2,goal3)). A sequence of acts written as act1&act2&act3 represents SEQ(SEQ (act1,act2),act3). Thus goals are accessible from the left and actions from the right.

It is now possible to state the clauses which form the PC-STRIPS program.

(S1) +SOLVE (goalatom&goalist, actsdone,allacts)
-SOLVE (goalatom, actsdone, newacts)
-SOLVE (goalist, new acts, allacts)

This isolates the next goal. The sequence of "allacts" is

intended to have "actsdone" as an initial subsequence.

(S2) +SOLVE (goalatom, START,START)
-GIVEN (goalatom)

If the only act done is the START, then it is checked whether the goal atom is given.

(S3) +SOLVE (goalatom, actlist&act, actlist&act)
-ADDED (goalatom, actlist&act)

If a sequence of acts has been done, it is checked whether the current goal atom was added by one of them.

(S4) +ADDED (goalatom, actlist&act)
-ADD (goalatom,act)

This checks to see if the most recent act added this goal atom.

(S5) +ADDED (goalatom, actlist&act)
-DEL (goldatom,act)
-NOBRANCH
-FAIL

(S6) +ADDED (goalatom, actlist&act)
-ADDED (goalatom, actlist)

If the goal atom was deleted by the most recent act, then (S6) is not accessible. If the goal atom was not deleted, then -DEL of (S5) fails to unify, so (S6) is tried next. Eventually, the following clause may be tried

(S7) +ADDED (goalatom, START)
-GIVEN (goalatom)

If the goal atom was not given and has not been added by the acts done, then it must be added by a new act.

(S8) +SOLVE (goalatom, actsdone, newactlist&newact)
-ADD (goalatom, newact)
-PRE (newgoalist, newact)
-SOLVE (newgoalist, actsdone, newactlist)

The preconditions of this new act form a new goal list which must be solved before returning to the previous goal list. This completes the PC-STRIPS program.

WARPLAN

Warren [10] has devised a modification of STRIPS, called WARPLAN, which allows the insertion of a new action into a previously evolved action sequence. This insertion is evoked when the new action destroys a previously achieved and protected subgoal. Warren's rather opaque, forty-six-line predicate calculus program, has been re-written to conform with PC-STRIPS, and has been simplified to the following twenty-line program.

(KW1) +PLAN (goalatom&goalist, goalsdone, actsdone, allacts)
-PLAN (goalatom, goalsdone, actsdone, newacts)
-PLAN (goalist, goalatom&goalsdone, newacts, allacts)
(KW2) +PLAN (goalatom, goalsdone, START, START)
-GIVEN (goalatom)
(KW3) +PLAN (goalatom, goalsdone, actlist&act, actlist&act)
-ADDED (goalatom, actlist&act)
(KW4) to (KW7) are the same as (S4) to (S7)
(KW8) +PLAN (goalatom, goalsdone, actsdone, newactlist&newact)
-ADD (goalatom, newact)
-PRE (newgoalist, newact)
-PLAN (newgoalist, goalsdone, actsdone, newactlist)
-PRESERVES (newact, goalsdone)
(KW9) +PRESERVES (newact, lastgoal&goalsdone)

(KW10) +PRESERVES (newact, lastgoal)
-DEL (lastgoal, newact)
-NOBRANCH -FAIL
(KW11) +PRESERVES (newact, lastgoal)
(KW12) +PLAN (goalatom, goalsdone, actsdone&lastact, newactlist&lastact)
-ADD (goalatom, newact)
-RETRACE (goalsdone, lastact, oldgoals)
-ACHIEVE (goalatom, newact, oldgoals, actsdone, newactlist)
-PRESERVES (lastact, goalatom)
(KW13) +RETRACE (goalsdone, lastact, oldgoals)
-REBUILD (goalsdone, lastact, earlygoals)
-PRE (goals, lastact)
-APPEND (goals, earlygoals, oldgoals)
(KW14) +REBUILD (lastgoaldone&othergoalsdone, lastact, earlygoals)
-ADD (lastgoaldone, lastact)
-REBUILD (othergoalsdone, lastact, earlygoals)
(KW15) +REBUILD (lastgoaldone&othergoalsdone, lastact, lastgoal lastgoaldone&oldgoals)
-REBUILD (othergoalsdone, lastact, oldgoals)
(KW16) +REBUILD (TRUE, lastact, TRUE)
(KW17) +APPEND(goalatom&goalist1,goalist2, goalatom&goalist1and2)
-APPEND (goalist1,goalist2,goalist1and2)
(KW18) +APPEND (goalatom, goalist2, goalatom&goalist2)
(KW19) +ACHIEVE (goalatom, newact, goalsdone, actsdone, newactlist&newact)
-PRE (newgoals, newact)
-PLAN (newgoals, goalsdone, actsdone, newactlist)

-PRESERVES (newact, goalsdone)

(KW20) +ACHIEVE (goalatom, newact, goalsdone, actsdone&lastact, newactlist&lastact)

-RETRACE (goalsdone, lastact, oldgoals)

-ACHIEVE (goalatom, newact, oldgoals, actsdone, newactlist)

-PRESERVES (lastact, goalatom)

References

(1) Battani, G., and Meloni, H. Interpreteur du langage de programmation PROLOG. Group d'Intelligence Artificielle, U.E.R. de Luminy, Marseille, (1973).

(2) van Emden, M.H., and Kowalski, R.A. The semantics of predicate logic as a programming language. Report M.I.P.-R-103, Dept. of Machine Intelligence, University of Edinburgh, (1974).

(3) van Emden, M.H. Programming with resolution logic, Machine Representation of Knowledge, (Elcock, E.W., and Michie, D. Eds), Reidel, Dordrecht, (1976).

(4) Fikes, R.E., and Nilsson, N.J. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2, (1971), 189-208.

(5) Green, C. The application of theorem-proving to question-answering systems. Technical note 8, Artificial Intelligence Group, Stanford Research Institute, (1969).

(6) Kowalski, R.A. Logic for problem-solving. DCL Memo 75, Dept. of Artificial Intelligence, University of Edinburgh, (1974).

(7) Kowalski, R.A. Predicate calculus as a programming language. Proc. IFIP 74, North Holland, (1974).

(8) Kuehner, D.G. Some special purpose resolution systems. Machine Intelligence 7, Edinburgh University Press, (1972).

(9) Nilsson, N.J. Problem-solving Methods of Artificial Intelligence. McGraw-Hill, (1971).

(10) Warren, D. WARPLAN; a system for generating plans. DCL Memo 76, Dept. of Artificial Intelligence, University of Edinburgh, (1974).

(11) Wos, L.T., Carson, D.F., and Robinson, G.A. Efficiency and completeness of the set of support strategy in theorem-proving. Journ. ACM 12, (1965).

A LINGUISTIC APPROACH TO AUTOMATIC THEOREM PROVING

Sharon Sickel
Information Sciences
University of California, Santa Cruz, Ca.
Research supported by the Office of Naval Research Grant # 76-C-0681

ABSTRACT

Generalizing the concept of a path in Clause Interconnectivity Graphs, we define the set of simple (i.e., cycle-free) paths that begin at a specified subset of nodes. Where the search of the CIG for a proof in the predicate calculus was previously defined in terms of the edges of the CIG, here the simple paths themselves become the atomic elements of the search, thereby increasing the "chunk" size of the operands. We can further define forms similar to regular expressions in which the terminal symbols represent those simple chunks. The forms become templates that model proofs, i.e., they can be mapped onto resolution proofs of the unsatisfiability of the clauses making up the CIG. In general a template represents an infinite number of paths but an algebraic computation on information derived from the templates yields valid proofs without an exhaustive search through intermediate stages of the search tree. Overall, the method leads to a reduction in both the computation time per step as well as in the combinatorics of the search itself. The representation also lends itself to an heuristic based on integer programming by using a simple difference function based on the chunks.

Introduction

A system for formal theorem proving is presented, using the Clause Interconnectivity Graph as its basic data structure. Proofs found here can be mapped onto proofs using resolution and factoring as rules of inference (as opposed to Modus Ponens, for example). The search method bears little resemblance to that of resolution methods, however.

The Clause Interconnectivity Graph (CIG) [5] has been used as a representation for proving first-order predicate calculus theorems. A CIG is a four-tuple:

< Nodes, Edges, Subst, Clause > where

Linguistic Approach...

Nodes is a set of graph nodes, one for each literal of each clause,

Edges is a symmetric relation between nodes such that <a,b> $\in$ Edges iff the literals associated with nodes a and b have opposite signs and unifiable atoms.

Subst is a mapping: Edges → substitutions such that Subst(<a,b>) is a most general unifier of the atoms of the literals associated with nodes a and b, and

Clause is a mapping: Nodes → $\mathcal{P}$(Nodes) where $\mathcal{P}$ means powerset; Clause partitions the nodes so that literals in the same clause have corresponding nodes in the same partition.

For example, suppose that we are dealing with integers defined by Peano's axioms, and we define the predicate, Even:

Even(0)

Even($s^n(0)$) → $\overline{\text{Even}}(s^{n+1}(0))$†

$\overline{\text{Even}}(s^n(0))$ → Even($s^{n+1}(0)$)

and theorem Even($s^{60}(0)$). Then the CIG is shown in Figure 1.

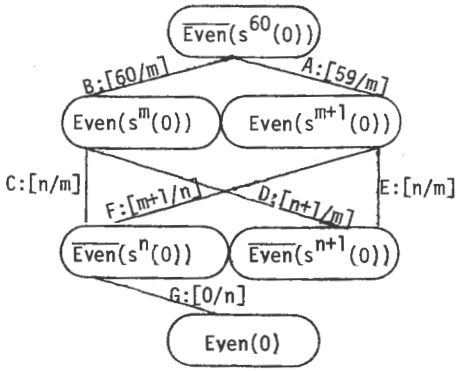


Figure 1. A Clause Interconnectivity Graph with labeled edges. The predicates and terms are left in the nodes for expository purposes only. They are neither included in the CIG definition nor are they used in the search for a proof.

† "s" means "successor"; s(0) = 0; $s^n(0) = s(s^{n-1}(0))$ for n > 0.

Edges is a symmetric relation. However, when we involve an edge in the search, the analogy is made to moving from one element of an ordered pair in Edges to the other element in that pair. Therefore when an edge is used, we think of it as being directed. Given an edge <a,b> and assuming direction a → b, we can make the following definitions.

Deleting_literal is a mapping: Edges → Nodes where

Deleting_literal(<a,b>) = b and

Residual_literals is a mapping: Nodes → $\mathcal{P}$(Nodes) where $\mathcal{P}$ means powerset.

Residual_literals(b) = Clause(b) - {b}.

A proof derived from a CIG corresponds to a particular kind of search on the CIG. The proof search resembles the following process:

> Choose a clause to be the starting clause (a clause that is likely to be used in the proof, a member of the set of support, etc.). Place a marker on each of the nodes in the partition representing the starting clause. Each of those markers may be moved along any edge connected to its present position. Then the parent marker is removed (from the deleting node) and children markers are placed on each of the other nodes (the residual nodes) in the partition arrived at from the move. Then the process is repeated on all of the existing markers; they in turn become parents, being replaced by children. The goal is to eliminate all markers.† This process corresponds to unrolling the graph into trees.

From looking at the CIG in Figure 1, it is easy to see that some move sequences could be done an arbitrary number of times, e.g., moves D,F,D,F,... successively, or E,C,E,C,... We call such sequences loops.

Assuming starting clause Even(0), the first move is determined, namely G. That leaves a marker on the node corresponding to $\overline{\text{Even}}(s^{n+1}(0))$. From this node we could begin one of the loops mentioned above. Let us consider a sequence of moves involving one of the loops; $G(DF)^kDA$, meaning move along G, then around D and F k times, then along D, then A. Intuitively G links up the integer

† This process is over-simplified. There are restrictions concerning the substitutions, and there is another allowable move that admits non-input steps. For a complete description, see [5].

0 with the start of an induction. The DF loop adds the value 2 to the current value. Move A jumps out of the induction to the value that we seek. In other words, the $G(DF)^k$ part is successively proving that 0 is even, 2 is even, 4 is even, etc., until we arrive just short of the given value. The D and A steps together add 2 to the value. In this case, k will have the value 29.

Once we have discovered $G(DF)^kDA$, proofs of the evenness of all even, positive integers should be equally easy in all systems. But we know that they are not. Using traditional deductive systems on this axiomatization, the length of the proof of Even($s^n(0)$) increases linearly with n, and required resources generally increase exponentially with the length of the proof. In this method, however, the discovery of the proof is of the same inherent difficulty regardless of the magnitude of n. The approach involves:

1) mapping the CIG onto a context-free grammar [1]
2) mapping the context-free grammar onto a set of expressions similar to regular expressions.
3) mapping each regular expression onto a composition of substitutions.
4) checking to see if any of the expressions represent a legal substitution. If so, that expression can be mapped onto a proof.

Chunking

The previously presented search schemes on CIG's dealt with looping by preferring non-loop moves, preventing run-away development of infinite loops. However, even in some simple cases, we may need to travel a loop many times. One example of this is the proof of evenness in which we should be able to prove Even(6000) easily once the general method is discovered. The proof itself may be long, but the search time should be identical to the search time in proving Even(60) or Even(6). In fact, it is possible to use this method not only to prove individual theorems, but also to derive generalized algorithms to do computations within a theory.

Once we know the basic steps needed for a proof, the repetition of one or

more of those steps a large number of times should not cause us any trouble. We need to discover these basic steps or chunks. One might imagine that the moves that correspond to edges might serve satisfactorily as chunks. However, there is some obvious clumping that takes place. The CIG in Figure 2 has three natural chunks, C_1:f, C_2:deg, C_3:abc, because the moves within each chunk must be taken together. Note that C_3 denotes a loop, and we can travel in either direction on a loop, so we can denote cba as C_3^{-1}. In this case, the chunking partitioned the edges, but that will not necessarily be the case.

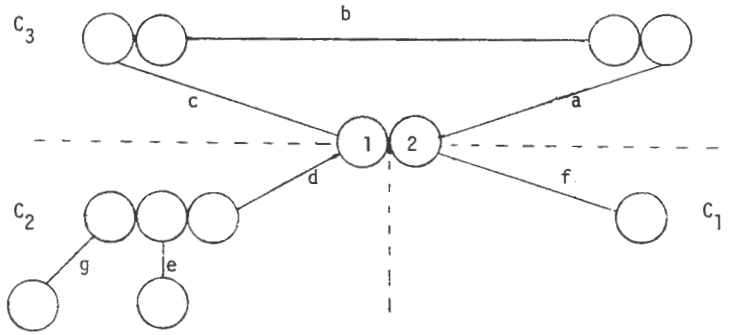


Figure 2. A CIG divided into its three natural chunks.

We can derive the chunks by finding all ways of moving and replacing the markers such that if a marker is on the same node as one of its ancestors, we freeze that marker, but continue to move other available markers. The starting configuration for each chunk is a single marker sitting on some node. The chunk is said to be related to that node. Intuitively, the chunk represents the refutation of the literal that the related node represents. This process identifies all of the natural pieces of the graph. Since no repeated looping is allowed, this is a terminating process.

We classify the chunks into two types, terminal and loop. A terminal chunk is one in which all markers have been eliminated. A loop chunk has one or more frozen markers. In Figure 2, C_1 and C_2 are terminal chunks; C_3 is a

loop chunk.

Chunks to Context-Free Grammar

Chunks as described in the previous section are trees, since 1) a parent marker may be replaced by one or more children markers and 2) no marker can ever be its own descendant. We wish to write the chunks as linear sequences so that we can use them in constructing a grammar. We produce this flattening by doing an end-order traversal [4] of the "chunk tree". The flattened form is a sequence of directed edges and nodes, s_1, s_2, ... s_n. We can make context-free productions by putting s_1, s_2, ... s_n on the right-hand-side and the associated node on the left-hand-side,

$$N \rightarrow s_1 s_2 \ldots s_n.$$

The intuitive notion is that to eliminate N you must add $s_1, s_2, \ldots s_n$ (possibly including N). We can now construct a context-free grammar G:

nonterminals: {S} ∪ Nodes (where {S} ∩ Nodes = ϕ)

terminals: Edges

productions: {all $N \rightarrow s_1 s_2 \ldots s_n$ as described above}
∪ {$S \rightarrow N_1 \ldots N_k$ | $N_1, \ldots, N_k$ represent all literals in starting clause}

start symbol: S

In the ground case any string in the language of G, i.e. any string that is derivable from S and consists entirely of terminals (in this case edges), represents a proof. Therefore, once the chunking is accomplished, determining theoremhood of the statement in question is equivalent to asking whether a given context-free grammar generates a non-empty language, which is a trivial problem.

The general case is more difficult, however. Each edge has an associated substitution, and for a string of edges to be acceptable, all of their substitutions must be mutually consistent. Consistent($\alpha_1, \alpha_2, \ldots \alpha_n$) iff $\alpha_1 \odot (\alpha_2 \odot (\ldots \odot \alpha_n)))$ is defined, where $\alpha \odot \beta = \gamma$ such that γ is a most

general substitution satisfying $(L\alpha)\gamma = (L\gamma)\alpha = L\gamma = (L\beta)\gamma = (L\gamma)\beta$ for an arbitrary literal L [5]. Since all terminal strings must abide by consistency, this is in fact a context-free attribute grammar [3] and can have the power of a type 0 grammar. This fact eliminates the usefulness of the result that tells us there is an upper bound on the length of the shortest string in the language. However, the grammar form provides us with some valuable heuristics as we shall see later.

Regular-like Expressions

Given a context-free grammar, it would be convenient to represent the language generated in regular expression style. To do that, we need to extend the definition of regular expression. In addition to "|", meaning "or", concatenation meaning "and", and "*" meaning "repeat zero or more times", we add exponent "n" to mean repeat exactly n times.† For the grammar constructed in the previous section, if all productions that have node N on the left-hand-side have one of $t_1,\ldots t_n$ (terminal chunks), or $r_1N,\ldots r_kN$ (loop chunks), then, intuitively, the expression $(r_1|r_2|\ldots r_k)^*t_1|\ldots|t_n$ represents the refutation for N and we denote it

$$N \overset{*}{\Rightarrow}{}^{\ddagger} (r_1|r_2|\ldots|r_k)^*(t_1|\ldots|t_n).$$

I.e. we can go around loops as long and in whatever order we choose, but we must finally end with a terminal.

In the example in Figure 2,

① $\overset{*}{\Rightarrow}$ $(abc)^*deg$, ② $\overset{*}{\Rightarrow}$ $(cba)^*f$.

It may be that by the above recursion method and by simple back-substitution for nonterminals of right-hand-sides having the corresponding nonterminals on the left, we can derive $S \overset{*}{\Rightarrow} p_1p_2\ldots p_n$ where $p_i \in$ Edges. For the example of Figure 2, the grammar is:

† This notation appears frequently in the literature on formal languages.

‡ $A \overset{*}{\Rightarrow} B$ means B can be derived from A by an application of zero or more productions.

({S,①,②...}§,{a,b,c,d,e,f,g}, P, S) where P:

S → ①②

① → a b c ①

① → d e f

② → c b a ②

② → f

By back-substitution we get: $S \overset{*}{\Rightarrow} (abc)^*def(cba)^*f$. Now by replacing each terminal by its substitution and interpreting concatenation of substitutions to mean ⊙, we can easily determine whether there exist non-negative integers n and m such that $subst^n(abc) \odot subst(def) \odot subst^m(cba) \odot subst(f)$ is defined. Note that we have replaced whole chunks by their substitutions. The substitution of a chunk is the ⊙ composition of the substitutions of the edges making up the chunk. Each time a loop is repeated a new instance of the clause at the endpoints of the loop is added. For this reason, a loop repeated n times will have n distinct instances of the variables. Loop substitutions, then, must be abstract descriptions including an unknown number of instances of variables. For example the substitution $[f(x_n)/x_{n+1}]$ specifies that each new instance of x is replaced by function "f" applied to the term substituted for the last instance of x.

For example, the grammar built from the CIG in Figure 1 having Even(0) as the start clause would cause S to generate (among others) the expression $G(DF)^*DA$. The corresponding substitution θ is

$$[0/n] \odot [n+1/m\ ,\ m+1/n]^* \odot [n+1/m] \odot [59/m].$$

$$\left.\begin{array}{l} m_i = n_i + 1 \\ n_{i+1} = m_i + 1 \\ (1 \le i) \end{array}\right\} \Rightarrow \begin{array}{l} m_i = 2i+1 \\ n_i = 2i \\ (1 \le i) \end{array}$$

§ The other nonterminal names and their productions are irrelevant to this discussion.

Differentiating between instances of variables, Θ becomes $[0/n_0] \circledcirc [2i/n_i, 2i-1/m_{i-1}] \circledcirc [n_k+1/m_k] \circledcirc [59/m_k]$ where $1 \leq i \leq k$. $m_k = 59 = n_k+1 = 2k+1$, therefore k = 29, indicating that the refutation consists of G, twenty-nine repetitions of (ΣF) and finally D and A. We will not go into how to generally describe loop substitutions, decide which instances of a variable are referred to by other substitutions, or compute the exponent of loops. However, for a given expression that is a regular expression extended by exponents and contains no node names (i.e., is completely terminal), it is straightforward to answer those questions. Due to lack of space the algorithms will be presented in a subsequent paper.

Integer Programming Heuristic

There will be grammars derivable from CIG's that do not easily admit the extended regular expressions. They include 1) grammars in which the self-referencing non-terminal appears in the middle of the right-hand-side (e.g., N → aNb) and 2) grammars in which a nonterminal can generate a string containing two copies of itself, e.g., $N \overset{*}{\Rightarrow} \alpha NN\beta$ where α and β are possibly empty strings of symbols, i.e., $\alpha,\beta \in (\text{Edges} \cup \text{Nodes})^*$. In the latter case, it is difficult to see the general recursion pattern since the length of the resulting string is exponential with the number of repetitions. In both cases keeping track of which instances of the variables to put in each substitution is a horrendous job in general.

By weakening the grammar, allowed by its particular use in this application, and not by distinguishing between different instances of the same variable, we can always derive an extended regular expression reduced to terminals, the terminals possibly reordered from what the grammar would actually generate.

Every chunk has a (possibly empty) effect on the total substitution in a solution. Terminal chunks have a fixed effect. Loop pieces may have a recursive effect. E.g., $[f(x_k)/x_{k+1}]$ has the effect of adding f to the accumulated effect and applying it to the new "x".

By combining the information from the reordered extended regular expression and the chunk effects, it is possible to write integer programming problems[2] whose solutions are likely candidates for proofs. In this way, the effects serve as difference functions for the chunks (operators) in much the way as is done in an operator difference table. The integer program tells us how many applications of each operator there are in likely candidates. The structure of the original grammar can then be used to check the validity of that candidate. An example of this is the "Even" problem in which we need to change the term from "0" in the start state to "$s^{60}(0)$" in the goal state. Therefore the sum of the effects of the chunks used must sum to exactly sixty applications of "s". In some cases, the start and goal states are not so clearly known and we have to phrase the problem slightly differently such that the original terms used in the solution plus the effects of all applied chunks sum to zero.

In cases where the regular expression forms are exactly known, the integer programming heuristic is substantially improved because the proper placement of variable instances is known. We may then break the problem into subproblems - one for each variable.

Work on the integer programming heuristic and computation of effects of more complex loops is currently in progress.

References

1. Hopcroft, John, and Jeffrey Ullman. Formal Languages and Their Relation to Automata. Addison Wesley, Menlo Park, CA. (1969)

2. Hu, T.C., Integer Programming and Network Flows. Addison Wesley, Menlo Park, CA. (1969).

3. Knuth, D. E., Semantics of Context-free Languages, Mathematical Systems Theory, 2 (Feb. 1968).

4. Knuth, D. E., The Art of Computer Programming, Vol 1. Addison-Wesley Menlo Park, CA (1969).

5. Sickel, Sharon, A Search Technique for Clause Interconnectivity Graphs, IEEE Transactions on Computers, Special Issue on Automatic Theorem Proving, (Aug. 1976).

AN EFFICIENT UNIFICATION ALGORITHM

Lewis Denver Baxter

Department of Computer Science, York University, Ontario

Abstract

An algorithm which solves the first-order unification problem is presented and shown to have a practically linear time complexity, relative to the length of the input expressions. The algorithm is composed of a transformational stage followed by a sorting stage. During the former stage, sets of pairs of expressions are transformed into a partition of expressions, which is equivalent with respect to unifiability. The partition is represented as a forest of trees and by using the technique of path-compression on balanced trees, a practically-linear complexity is achieved. In the sorting stage, the output partition induces a directed graph, which is then topologically sorted. If successful, the sort indicates the most general unifier.

Introduction

The unification problem arises from automatic theorem-proving. It is to determine, given two expressions e_1 and e_2 containing variables, whether there exists a substitution of these variables by expressions which, applied to e_1 and e_2, makes them equal.

The first unification algorithm, discovered by Robinson [4] and based on simple string data structures and the physical manipulation thereof, was of exponential complexity. A later algorithm, also by Robinson [5], represented expressions by trees and performed substitutions by manipulating pointers to these trees. Unfortunately, this algorithm was of exponential complexity due to an inefficient method of determining if a variable occurs in an expression. This defect was easily remedied by Venturini-Zilli [7] who proved that this improved algorithm had a quadratic time complexity.

Whereas the above algorithms were based on the original "left-to-right" processing of the input expressions, a new algorithm, composed of a transformational stage followed by a sorting stage, was discovered by Baxter [1]. The use of good data structures applied to this algorithm results in the practically linear algorithm presented here. ("Practically linear" means linear times a very slowly growing function.)

Notation

We will assume familiarity with the notation found in the literature [4, 5]. Briefly, an expression is either a variable or a constant (function) symbol of degree (number of arguments) n, followed by n expressions. A term is defined here as an expression which is not a variable. The length of an expression is the total number of occurrences of variables and constants. The substitution $\{v_1 \leftarrow e_1, \ldots, v_n \leftarrow e_n\}$ refers to the simultaneous replacement of the variables v_i by the corresponding expressions e_i. The application of the substitution σ to the expression e is written: $\sigma(e)$. The substitution σ unifies a set of expressions $\{e_1, \ldots, e_n\}$ iff $\sigma(e_1) = \ldots = \sigma(e_n)$. σ unifies a partition of classes of expressions iff σ unifies each class in the partition. We abbreviate most general unifier to mgu.

Description

Our algorithm consists of two stages: a transformational stage followed by a sorting stage. The former inputs, in general, a set of pairs of expressions and outputs a partition of expressions. This stage may fail due to the attempt at unifying two expressions beginning with different constant symbols. The sorting stage constructs from this output partition a directed graph (digraph) and determines if it contains a circuit by trying to topologically sort the digraph. If a circuit is found then unification fails because we cannot unify a variable with an expression in which it occurs. If no circuit

is found, the topological ordering indicates the mgu of the input set. We now describe these two stages in more detail.

Transformational Stage

The two main sets used in this stage are S, a set of unordered pairs of expressions, and F, a partition of expressions. Initially, S is the input set S_I to be unified and F_I, the initial value of F consists of all the subexpressions occurring in S_I, each in a class of its own. Finally, S will be empty and F will be the output partition F_0. We present this stage in the form of an abstract algorithm:

```
algorithm TRANSFORM:
begin
   Initialize S to S_I and F to F_I ;
   repeat until S is empty :
   begin
      Delete a pair of expressions, {e_1 , e_2}, from S ;
      if e_1 ≠ e_2
      then begin
             Find classes T_1, T_2 ∈ F
             such that e_1 ∈ T_1 and e_2 ∈ T_2 ;
             if T_1 ≠ T_2
             then begin
                    if T_1 contains a term f'(e'_1, ..., e'_n)
                    and T_2 contains a term f"(e"_1, ..., e"_m)
                    then if f' ≠ f"
                         then UNIFICATION FAILS
                         else Add to S the pairs:
                              {e'_1, e"_1}, ..., {e'_n, e"_n} ;
                    Merge T_1 and T_2, that is,
                    replace T_1 and T_2 by T_1 ∪ T_2 ;
                  end;
           end;
   end;
end.
```

In order to obtain an efficient algorithm from this, we must now specify appropriate data structures. Expressions are represented by trees in which each vertex corresponds to some symbol occurring in the expression. If a vertex corresponds to a constant symbol of degree n, then it has n sons, each corresponding to an argument. Also, different occurrences of the same variable are represented by different pointers to the same vertex of a tree.

The set S is represented by a stack of pairs of pointers to the corresponding tree representations of the expressions. For example, the set:

{ {w, F(x, G(y))}, {G(F(F(y, x), z)), G(w)} }

is represented:

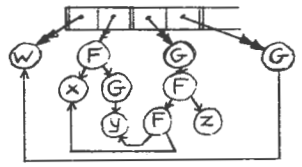


Figure 1

The partition F is represented as a forest of trees. Each class in the partition is represented as a tree, each vertex of which points to an expression. Since we must quickly determine if a class contains a term, the root of a tree points to some term, known as the designated term of the class. For example, the partition:

{[u, v, G(F(w, x)), G(z)], [x, H(w), H(t), s], [F(w, x), y, z, F(r, s)],
[w, r, t]}

is represented as follows. Note that each expression is, in fact, a pointer to its tree representation. The large arrows indicate the designated terms.

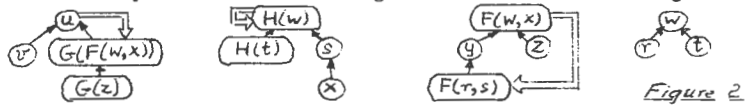


Figure 2

We now describe how to efficiently manipulate these data structures required by the algorithm, TRANSFORM. Rather than checking if e_1 and e_2 are equal expressions, we only check if their corresponding pointers are equal. Further, we can easily extract the arguments of an expression by examining its tree representation. The operations to be performed on S are simply: to delete a pair from S and to add pairs to S. These are easily accomplished when S is represented by a push-down pop-up stack.

The efficiency of the transformational stage depends on the method of performing two operations on the partition, F: to FIND which class in F an expression belongs; and to MERGE two classes of F.

To FIND which class an expression belongs, we traverse a path from the vertex of the tree corresponding to the expression to the root; this root is effectively the name of the required class. The cost of a FIND is proportional to the length of the traversed path. This will be reduced if we employ a collapsing heuristic: after finding the root, we collapse the path directly onto the root. Formally, if $v_1 \to v_2 \to \ldots \to v_n$ is the unique path from the vertex v_1 to the root v_n, then we replace the edge $v_i \to v_{i+1}$ by the edge $v_i \to v_n$ for i=1, ..., n-2. The following figures illustrate the representation of the class $[e_1, e_2, \ldots, e_{17}]$ before and after FINDing the class which contains the expression e_{15}.

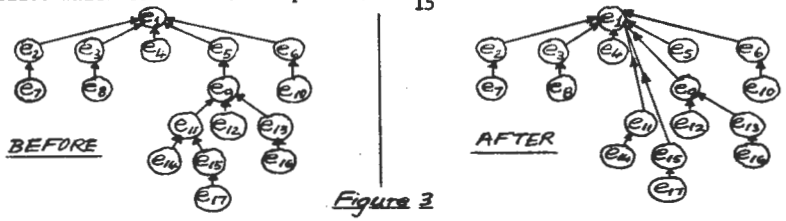


Figure 3

To MERGE two classes, we make one tree representing one of the classes a subtree of the tree representing the other class. To decrease the average path length and hence the cost of subsequent FINDs, we employ a balancing heuristic: make the "light" tree a subtree of the "heavy" tree, where the comparatives refer to the number of vertices in the tree. In the case when the "heavy" tree contains only variables and the "light" tree contains some term, we have to ensure that the new root points to the designated term. For example, after merging the first and third classes represented in Figure 2, we obtain:

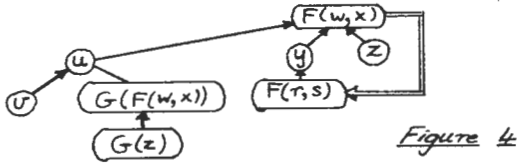


Figure 4

Sorting Stage. From F_0 we will first construct an abstract intermediate digraph, which is naturally induced by F_0. It has as

vertices the classes in F_0. Its edges are constructed by examining each class in F_0. Given a class T in F_0, let e be any term, say $f(e_1, \ldots, e_n)$, in T. (If no such term exists, then T contributes nothing to the set of directed edges.) Let e_i belong to the class T_i (i=1, ..., n), then T contributes the set of directed edges: $T \to T_1, \ldots, T \to T_n$. For example, the partition of Figure 2 induces the following digraph, where underlined expressions denote the designated term of a class.

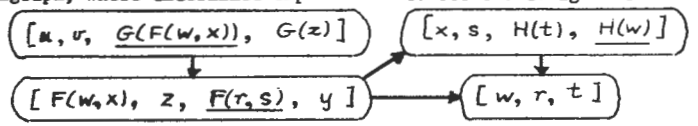


Figure 5

In practice, we must construct a related digraph directly from the forest representation of F_0. The vertices and edges of this digraph are obtained as follows. For each vertex, v, in the forest, which corresponds to a variable and which is not a root, let r be the root of the tree to which v belongs; add the directed edge: $v \to r$. Also, for each root, r, let $f(e_1, \ldots, e_n)$ be the designated term of the tree having root r and let r_i (i=1, ..., n) be the root of the tree to which e_i belongs; add the directed edges: $r \to r_i$. For example, the forest representation of Figure 2 induces the digraph:

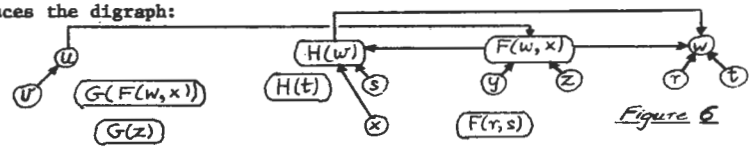


Figure 6

We now attempt to topologically sort this constructed digraph (embed its vertices in a linear order), using the well-known linear algorithm [3]. If the digraph cannot be sorted then unification fails, otherwise the topological ordering indicates the mgu. Let $v_1, \ldots, v_n$ be the subsequence of the linear order which corresponds to variables only. Then the mgu is $\{v_1 \leftarrow e_1, \ldots, v_n \leftarrow e_n\}$ where e_i is the designated term of the class to

which v_i belongs; if no such term exists then e_i is the variable which corresponds to the root of the tree to which v_i belongs.

Details of the proof of correctness are found in [2]. In the transformational stage, the mgu of S_I is the same as that of F_0. This is proved by showing that the assertion:

$\forall\sigma$ (σ unifies S_I iff σ unifies S and σ unifies F) holds each time the loop of the algorithm is entered. The correctness of the sorting stage depends on the following special properties of F_0: All the terms in each class of F_0 begin with the same constant symbol; and the "hereditary" property: If $f(e'_1, \ldots, e'_n)$ and $f(e''_1, \ldots, e''_n)$ belong to the same class of F_0 then for all i, e'_i and e''_i belong to the same class of F_0.

Complexity

The complexity of the transformational stage is practically linear, that is, of order $nG(n)$ where G is a very slowly growing function. The complexity of the sorting stage is linear.

We now define G using the definitions of [6]. Define the function A on pairs of integers by:

$A(0, x) = 2x$ for $x \geq 0$; $A(i, 0) = 0$ for $i \geq 1$; $A(i, 1) = 2$ for $i \geq 1$ and $A(i, x) = A(i - 1, A(i, x - 1))$ for $i \geq 1$ and $x \geq 2$.

Define $G(n) = \alpha(n, n)$ where α is a functional inverse of A :

$\alpha(m, n) = \min\{z \geq 1 \mid A(z, 4 \lceil m/n \rceil) > \log_2 n\}$ $m, n \geq 1$.

G is "practically" constant, since $G(n) \leq 3$ for $n < 2 * 2 * \cdots * 2$ (65536 occurrences of 2), where "*" denotes exponentiation.

Ignoring the cost of FIND and MERGE instructions, the tranformational stage has linear complexity. The results of Tarjan [6] tells us that the additional time to process a sequence of FIND and MERGE instructions, using the technique of path-compression on balanced trees, requires practically linear time. Details are found in [2].

References

[1] BAXTER L.D. (1973), "An efficient unification algorithm", Research Report CS-73-23, Department of Computer Science, University of Waterloo.

[2] BAXTER L.D. (1976), "The complexity of unification", Ph.D. Thesis, in preparation, Department of Computer Science, University of Waterloo.

[3] KNUTH D.E. (1968), The Art of Computer Programming, Volume I: Fundamental Algorithms, Addison-Wesley.

[4] ROBINSON J.A. (1965), "A machine-oriented logic based on the resolution principle", JACM 12, 1, 23-41.

[5] ROBINSON J.A. (1970), "Computational logic: the unification computation", in Machine Intelligence 6, American Elsevier, 63-72.

[6] TARJAN R.E. (1975), "Efficiency of a good but not linear set union algorithm", JACM 22, 2, 215-225.

[7] VENTURINI-ZILLI M. (1975), "Complexity of the unification algorithm for first-order expressions", Research Report, Consiglio Nazionale Delle Ricerche Instituto per le applicazioni del calcolo.

CAN FRAMES SOLVE THE CHICKEN AND EGG PROBLEM?

William S. Havens

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada

Abstract

The types of search strategies that have been proposed for frame systems are discussed. They are shown to be essentially top-down, hypothesis driven mechanisms. It is claimed that these mechanisms are inadequate for a large class of recognition problems. "The Chicken and Egg Problem" is presented. A new model of recognition for frame systems is proposed and an example of its operation is given.

1. Introduction

The concept of frames as a paradigm for the representation of knowledge is an intuitively appealing idea which has generated a great deal of interest in the A.I. community. There has been however only limited progress in formalizing and developing the theory into a useable computational model. According to Minsky's[4] original paper, frames are data structures for representing stereotypical objects, concepts, and situations. Each frame contains a set of terminal slots which may initially contain default assignments about the stereotype the frame represents. When the frame is called upon to represent some particular instance of its stereotype, the defaults behave as expectations of what kind of information to look for to fill the slots.

This model for frames has a number of unfortunate consequences. First, it forces the use of top-down, goal directed search strategies. A candidate frame is chosen to represent some situation on the basis of some initial expectations about that situation. This frame then proceeds to attempt to fill its slots by making observations and by calling

on the efforts of other "sub-frames". The frame is guided in its search by the expectations it has coded within it. In the case of an improper first choice of a candidate frame, the mechanism for choosing an alternate candidate is completely driven by the failure of the first frame to succeed. This is of course classical automatic backtracking with all its inherent problems. Minsky, recognizing this fact, proposed a modification to backtrack search that avoids the duplication of effort for identical sub-goals. When a frame discovers from observation that it is not applicable to a given situation, it consults a similarity network which recommends a replacement candidate. The frame then attempts to map its "correctly" filled terminal slots into the slots of the new candidate frame and then passes control to it. This scheme assumes both that a mapping exists between each failing frame and each next candidate and that the similarity network is sufficiently "complete" that relatively few inexplicable failures occur. Such "surprises" force the system to rely entirely on backtracking to continue the search.

Secondly, the model requires a frame to be the currently active candidate before its expertise can be of any assistance in the recognition process. This means that the search process will spend a good deal of its time proposing specific candidate frames one after another based only on the types of failures that can successfully be processed by the similarity network. Only when the proper frame is finally chosen will the knowledge specific to recognizing instances of that frame be available. That specific knowledge must be available much earlier to intelligently guide the search process.

For example, consider a frame-based scene recognition system presented the scene of Figure 2. From the information present in the scene, the system must select the prism frame to represent the image. The prism frame supposedly contains expert knowledge on the best way to recognize prisms. But the system is not told that it is "seeing" a prism; indeed that is the system's task. The knowledge that prisms are polyhedrons composed of polygonal bases connected by parallelogram faces is contained within the expectations of the stereotypical prism frame. Yet, unless the system already had the prism frame active to provide it with these expectations, it could not use this knowledge to find the frame from the information in the scene. Mackworth[3] has called this "The Chicken and Egg Problem".

2. A Model of Recognition

To remedy the difficulty, a new model of recognition for frame systems has been developed. Frames in this model follow in principle the form proposed by Minsky. Frames are organized about stereotypes and are encoded as descriptions of the frame's expectations about the real world. The model, however, inverts the concept of what a frame does. A frame recognizes instances of itself not only by comparing its internal expectations against external observations, but also by matching its evolving instance with the expectations of other frames. That is, the frame is responsible for recognizing what higher structures it can be part of. Each frame exists as an individual recognizer in a system of such recognizers, the frame system. Instead of being an inherently top-down search process, now the recognition can proceed using simultaneously both top-down and bottom-up

techniques.

The recognition model consists of three phases. They are called expectation, matching, and completion. Initially the system exists as a top-level frame containing a set of expectations about what it expects to find during its observations. As each input observation is made, it is matched against this set of expectations. Any successful matches in turn cause the expectations to compute a next generation of expectations. This process iterates until such time as a particular sequence of expectations and the observations they match have satisfied a frame's internal criteria for the recognition of some concept, object, or event. This begins the completion phase. The completion phase creates an instance of that frame. This instance then enters the matching process. At this point, the frame acts as an abstract internal observation and itself participates in the matching process with the expectations of other frames. If it succeeds in matching the expectations of some other frame, then it will be composed into the evolving description of that frame. In our vision example suppose the system discovers a triangle. The triangle frame then creates an instance of this particular triangle and attempts to match the instance against the expectations of other frames. If the match is successful, a new set of expectations are generated and new observations taken.

The role of the frame in this model is an active process. Each frame is organized about a procedure called a scenario. A scenario contains the knowledge to perform the iterative cycle of attempting to match some relevant input observation or abstract internal observation against the frame's expectations.

When stimulated from the outside world or some other frame, the scenario enters into a contractual negotiation with that stimulus, the matching process. If the match is successful, the frame propagates its scenario, i.e., it creates a new set of expectations about its role in the world. and the process continues.

The matching process is characterized as a negotiation. When two frames negotiate a match, one frame will be attempting to match the expectations of its scenario against a second frame's attempt to perform a completion. That is the second frame is attempting to compute the last step in its scenario. It is trying to justify its existence by computing its place in some higher scenario. This process is recursive. Computing a frame's progress in its scenario causes the frame to negotiate a match with the expectations of other frames which in turn causes those frames to recompute their progress in their own scenarios. At each level, each frame is attempting to discover how it "fits" into some higher scheme of things. In this model, no long chains of expectations about all things possible in the world are required. Neither does the system need a mechanism for trying one frame after another mapping each time the terminals of the failing frame into the next candidate frame. Instead, a frame's scenario is responsible for knowing which higher frames it can plausibly be part of. The scenario then attempts to match those frames thereby activating them only when needed.

The matching phase is also the vehicle by which non-determinism, i.e., local ambiguities in the real world, is handled. The frame which is computing its completion may match

with more than one other frame, thereby spawning a number of different interpretations. Later, as observations remove the ambiguity, the fallacious interpretations can be deleted. A good analogy is perhaps to a capital investment market. A buyer, the completing frame, has some capital to invest, the description he has worked hard to complete. But he wants to invest wisely. He may consider the offers of a number of sellers, i.e., he may attempt to match the expectations of a number of frames that are attempting to complete their own scenarios. By matching, he eliminates some sellers and decides to spread his investment among the others, perhaps investing most heavily in those frames that match his requirements best. Later as events unfold, the contracts he has written can specify which investments are to be continued and which cancelled depending on the dividends they show.

3. A Detailed Example

This example describes the operation of the model as a recognizer for line drawings of polyhedral objects and is similar to an example given by Kuipers[2]. The line drawing presented as input to the recognizer is shown in figure 2 and is in the form of a network of vertices and edges. Each vertex and each edge is represented as a primitive frame. Each vertex knows its type, which is either an L-vertex, a T-vertex, an ARROW-vertex, or a FORK-vertex. Vertices also know the edges they are formed from and the approximate size of the angles between their edges. Each edge knows only the two vertices it connects. In this example, polyhedral objects are composed of polygonal faces which are in turn composed of edges and vertices. Figure 1. shows this composition hierarchy.

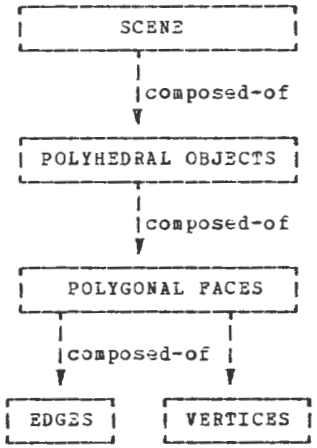


Figure 1.

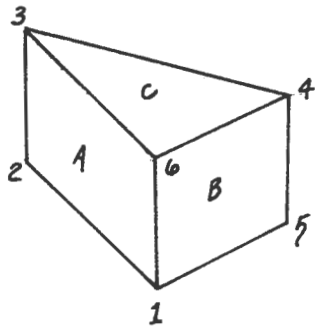


Figure 2.

The top-level frame is the resident expert at recognizing scenes. Its goal is to match the instances of edges and vertices in the data to the polygonal face frames' expectations of how edges and vertices can make up polygon faces, then to match these faces to the polyhedral object frames' expectations of how faces can make up polyhedral objects, and finally to match these objects to its own expectations of how polyhedral objects can form scenes. The top-level frame's scenario must be generally applicable to the recognition of all scenes of polyhedral line drawings. It begins by looking at vertices on the periphery, as they are pregnant semantically and less ambiguous than internal vertices. If the enumeration of peripheral vertices fails to complete the recognition of a scene, then it selects interior nodes to examine. Else it fails.

The Chicken & Egg Problem

This general top-level scenario is not the only scheme the system will use. The frames for polygon faces are experts in their own domains, the recognition of faces. Each face frame, depending on the type of face it is looking for, has a scenario especially tailored for effective recognition of that type. Likewise, the scenarios of the polyhedral object frames contain the knowledge to guide the search for polyhedral objects.

The top-level frame first chooses to examine peripheral vertex 2. Vertex 2 is an instance of the L-vertex frame. The scenario associated with each vertex frame is only to attempt its completion phase because its existence was explicitly given in the data. Therefore, the L-vertex attempts to match its given description against the expectations of those face frames that it can plausibly be part of. It can be the corner of either a parallelogram face or a triangle face. It must find instances of these two frames to match. From its knowledge of line drawings, it knows that if face recognizer frames already exist for the particular face that it must be part of, they will be associated with its neighboring vertices. That is, this vertex can use the original input data as a semantic network to access instances of face recognizer frames to match. The neighbors of vertex 2 are vertices 1 and 3, neither of which have bound to them face recognizer frames. So vertex 2 creates new instances of both the parallelogram and triangle frame, succeeds in matching them both, and binds them in the network at vertex 2.

Note the occurrence of non-determinism at this first vertex. Minsky and Kuipers would choose one hypothesis, perhaps that the face is triangular. Later, if that hypothesis fails,

they would then have to execute some mapping of terminals from the triangle frame into the parallelogram frame. In this model, the L-vertex creates two descriptions of its role in the evolving face labelling and successfully matches one against the expectations of the triangle frame and the other against the expectations of the parallelogram frame. Note also the composition process. A description of the L-vertex has been incorporated into the evolving syntheses of both face frames.

The top-level frame is again faced with making an observation, so it continues with its scheme of enumerating peripheral vertices. This time it chooses vertex 1, and this vertex has the responsibility of finding a face frame that it can match. It "looks" at vertex 5 by first consulting edge 1-5 but no expectations are lurking there, and likewise for vertex 6 via edge 1-6. But when it looks at vertex 2, vertex 1 finds both the parallelogram and triangle frames. It must negotiate a match with both. When vertex 1 attempts to match the triangle frame, the match fails because the expectations of the triangle are that the sum of the angles of vertex 2 and vertex 1 will be somewhat less than 180°. In this case, they equal 180°. The triangle hypothesis is rejected and its frame instance is deleted. When vertex 1 attempts to match the parallelogram frame however, the match succeeds. The parallelogram frame expects a neighbor of vertex 2 to be either a FORK-vertex, ARROW-vertex, or T-vertex. Since it represents a parallelogram, it expects that the sum of the angles of vertex 2 and an angle of one of its neighbors to be approximately 180°. The parallelogram frame now propagates its scenario, resulting in the creation of a new set of expectations.

Its scenario, by this time, feels sure that it is going to succeed. The angle measurements are a good cue for the parallelogram because opposite angles must be equal. The frame consults edge 1-6 again to access vertex 6, asks the vertex for an angle measurement, and discovers an angle equal to the angle of vertex 2. The search process has now switched from a bottom-up search driven by the vertices into a top-down search directed by this parallelogram.

By this time, the parallelogram frame's scenario is very near to finding the completed parallelogram and it consults the neighbors of vertex 6 looking for the particular neighbor that is also a neighbor of vertex 2. When vertex 3 is found, its angle is checked against the proper angle of vertex 1. They are equal and the recognizer concludes that it has found a parallelogram face. It then composes face "A", an instance of the parallelogram frame, from vertices 1,2,3 and 6.

The recognition process now ascends one level. Face "A" is trying to match the expectations of polyhedral object frames. Again the input data can be used as a semantic network to look for instances of these frames. From the fact that vertices 1,3, and 6 have more than two edges, we know that they are also part of some other faces. If these other faces had been recognized before face "A", there would be expectations for one or more object frames bound to these vertices. In this case, no other faces have been discovered, so polyhedral object frames which can have parallelograms as faces are created and bound to vertices 1, 3, and 6.

The process continues with the vertices creating, propagating, and completing face recognizers. In turn, the

faces continue the process of creation and propagation of object recognizers. In this example, when an object frame finally performs a completion, it immediately matches the scene recognizer frame. The search has succeeded and the system returns a composed instance of the object to the user.

4. Conclusion

I would like to apologize for the imprecision in this model. The ideas are new and have not had time to fully coalesce. We are currently in the process of implementing the model as a high-level programming language called MAYA[1]. At present the implementation is approximately fifty-percent complete. It is hoped that MAYA will provide a good experimental domain in which to further explore the theory of frames.

Bibliography

1. HAVENS,W.S., A user's guide for MAYA, working paper, Dept. of Comp. Science, UBC, Vancouver, Canada, 1976.

2. KUIPERS,B.J., A frame for frames: Representing knowledge for recognition, in D.G.Bobrow & A.M.Collins(Eds.), Representation and Understanding, Academic Press, New York, 1975.

3. MACKWORTH,A.K., How to see a simple world, TR-75-4, Dept. of Comp. Science, UBC, Vancouver, Canada, 1975.

4. MINSKY,M., A framework for representing knowledge, in P.H.Winston(Ed.), The Psychology of Computer Vision, McGraw-Hill, New York, 1975.

A Formalism for Modelling

Hector Levesque, John Mylopoulos, Gordon McCalla,
Lucio Melli, and John Tsotsos
Department of Computer Science
University of Toronto

Abstract

This paper describes a formalism for the construction and use of a model representing knowledge of some domain. Some of the features of the formalism are the use of an ISA HIERARCHY, a PART-OF HIERARCHY and procedural attachment for objects that are part of the model.

1. Introduction

This is an extension of the formalism proposed by Abrial [1] for the construction and use of a model representing knowledge of some domain. Our main goal has been to develop a representation that is sufficiently powerful to describe its own operation at a level that is more "natural" than that, say, of LISP. The models built are explicit in that all semantics of concepts in the model can be described using the formalism, and examinable in that the parts can always be inspected at various levels of detail. In this sense, our approach has been declarative. Moreover, models are incomplete in that, at any given time, the system using them has only a partial knowledge of the domain represented. It must, therefore, take this into account when answering questions and be prepared to receive new information, determine its acceptability and modify the model accordingly. Similarly, it must distinguish between information that is definite and final from that which is tentative or valid only in certain situations.

The knowledge included in the model may be defined at different levels. There are simple "facts" like:

John is a person.

The sex of Joe is masculine.

Mary is not the wife of Bill.

simple rules like:

All students are persons.

Every person has two parents of whom he is the child.

and more elaborate rules like:

The sex of a person is not subject to change.

A person's uncle is the brother of one of his parents.

A person can have only one location at any given time.

The approach we will take in this paper is bottom up in that we will describe informally the basic operations of the model, only hinting at the more interesting higher level constructs that can be derived. Although no explicit syntax is given in the paper, we present a number of sample expressions and programs to illustrate various aspects of the formalism's descriptive power. All such examples are numbered for reference purposes.

2. Constructing a Model

The most primitive type provided by the formalism for the construction of a model is the object which is simply any single conceptual unit that can be referred to as a whole. An object enters the "perception field" (becomes part) of the model with new and is removed by kill. Thus,

john := new (1)

creates a new object with a unique internal name and "john" as external name.

A fundamental notion to the organization of the model is the class which simply represents a collection of objects sharing common properties. These objects are instances of the class and may themselves be classes. When specifying a class as being a subclass of another, we are informing the model that, unless otherwise indicated, all instances of the subclass are in fact also instances of the superclass. The class of all objects that may be part of the model is called "object". All classes are therefore subclasses of "object". For example,

person := new (2)

person => object (3)

creates a new object called "person" and defines it as a subclass of "object". Syntactically, (2) and (3) can be combined into

person :=> object (4)

and asserted with

male :=> person (5)

female :=> person (6)

student :=> person (7)

female-student :=> student (8)

female-student => female (9)

to set up an organization of classes generally referred to as the "ISA HIERARCHY".

To specify that an object is an instance of an existing class we will use the notation "->" as in:

john -> male (10)

bill := new (11)

bill -> person (12)

(11) and (12) can be combined into

bill :-> person (13)

To denote the fact that an object is not a subclass or instance of a class, we use the notation "¬" followed by the operator, as in:

female-student ¬=> female (14)

john ¬-> person (15)

When introducing a subclass or an instance, it is often necessary to provide definitional information for it. For example, if we assume that a student is defined by a student number and a department, to simply say that

jim :-> student (16)

does not give sufficient information about "jim". We can write

jim :-> student with num<-702377167,dept<-dcs (17)

to provide the appropriate information.

Relations

A very important primitive class is that of binary relations or simply relations which are maps from one class (the domain) to another (the range). Instances of binary relations will be called links and they relate an instance of the domain and an instance of the range.

Relations are created like any other class. The most generic one is called "relation". For example:

children :=> relation with domain<-person,range<-person,
d-interval<- <0,∞>, r-interval<- <2,2> (18)

The arguments indicate that "children" is a relation from "person" to "person" such that for each instance of the range there are exactly 2 domain instances. Thus a person can have 0 to infinity children, which are persons, and furthermore is the child of exactly 2 persons. Further examples:

wife :=> relation with domain<-male, range<-female,

d-interval<- <0,1>, r-interval<- <0,1> (19)

sex :=> relation with domain<-person, range<-sex-value
d-interval<- <1,1>, r-interval<- <0,∞> (20)

Relations like other classes may be organized into an ISA HIERARCHY. For example, in

oldest-child :=> children with d-interval<- <0,1> (21)

the domain, range, and r-interval are inherited from "children". We can define very general relations like

inter-personal :=> relation with domain<-person,
range<-person (22)

must-hold :=> relation with d-interval<- <1,1> (23)

and then create new subclasses as restrictions of these.

We will henceforth use "R" to represent a relation, and "x" and "y" to represent instances of the domain and range respectively. Therefore "R:x->y" instantiates the relation provided the cardinality constraints of the d-interval and r-interval are not violated (in which case a failure occurs). For example:

wife : john -> mary (24)

children : john -> bill (25)

To negate an instantiation, we write:

wife : john ¬-> mary (26)

3. Examining a Model

Logical Information

To obtain logical information from the model, we present it with a "conjecture" and receive as reply one of true, false, or unknown. There are two primitive conjectures: the equality test and the test of a relation.

The equality test is always of the form "x=y" and is a test for identity of internal names. The value of such a conjecture is unknown when one of the two arguments has an unknown value.

To find out if a relation "R" holds between "x" and "y" we write "R:x?y". For example, consider the "children" relation of (18) and suppose

children : john -> bill (27)

children : mary ¬-> bill (28)

then we have that

children : john ? bill is true (29)

children : mary ? bill is false (30)

children : jill ? bill is unknown (31)

If we now assert that

children : susan -> bill (32)

then (31) conjectured now would be false.

The conjecture

isa : student ? person (33)

asks whether "student" is a subclass of "person". On the other hand, "x?y" tests whether "x" is an instance of class "y". Arguments can be passed as in:

jim ? student with dept<-math (34)

The actual operation of testing is very dependent on the class being tested.

Value Information

There are essentially two ways of obtaining value information from the model. The first is fairly trivial and involves using the name of a previously defined object. The second method is to access a relation, that is, to present it with an instance of the domain and receive as value(s) instance(s) of the range.

When the maximum cardinality of a relation is 1, the notation "R(x)" denotes the range instance "y" (if it exists) such that "R" maps "x" into "y". For example:

sex(john) (35)

wife(joe) (36)

The value of such an expression is an instance of the range, unknown, or nothing. The value is unknown when the minimum cardinality specifies that there must be an instance of the range although no such instance is known. The value is nothing when there need not be an instance of the range. For example, "sex" of (20) is of the first type, while "wife" of (19) is of the second type. To indicate that "joe" does indeed have a wife whose identity is unknown we write:

wife : joe -> unknown (37)

When the maximum cardinality of a relation exceeds 1, the concept of a generator is needed to produce values one at a time. To create a new generator, we use the notation

g :-> generator with class<- c (38)

where "c" is a class. Now "g" is a generator which uses a snapshot of class "c" taken at the time of instantiation, to produce instances of "c" known at that point.

For relations, a subclass of "generator" called "accessor" is used to produce instantiations. To create an accessor we use "g:->R[x]". For example,

w :-> children[john] (39)

makes "w" a generator of children of "john".

4. Abstract and Indefinite Objects

The objects we have considered so far are concrete in the sense that they enter the perception field of the model at the time of their creation and leave at the time of their destruction. For some objects, however, it is unreasonable to speak of them as entering or leaving the perception field since the model is assumed to have a complete knowledge of them. Thus they are never defined explicitly but only referred to. We call these objects abstract. Typical abstract objects are numbers, identifiers and tuples. Of course, abstract objects may have other names as in:

four := 4 (40)

tuple-25 := <1,1,'Jack',Jack> (41)

Note that although a tuple is abstract, its entries need not be. We can have abstract classes as well, which are simply arbitrary collections of objects. For example,

truth-value := {true,false,unknown} (42)

sex-value := {masculine,feminine} (43)

In all cases, the distinguishing property of absract objects and classes is that their meaning is self-contained in the sense that they need not be related to other objects (i.e. "placed" on the ISA HIERARCHY) to be understood.

An important consequence of the incompleteness of the model is that, if at some time it has the same knowledge of two objects, this does not mean that they are the same object. Thus, when an object enters the perception field of the model, it must identify itself as new or known. However, it is often convenient to be able to postpone the decision until enough information has been gathered concerning the object. We call such objects

indefinite. We will use the operator a or an to create an indefinite object. For example,

murderer-of-Bill := a person (44)

evening-star := a planet (45)

morning-star := a planet (46)

versus

venus :-> planet (47)

with the understanding that they are to be treated differently from "definite" objects. In fact, unknown is really just a synonym for "an object".

We can also attach restrictions to these indefinite objects as to what identities they can possibly have. For example,

x := a student with dept <- math (48)

w := a person suchthat age(self) < 25 (49)

where suchthat specifies a condition that must be true for the object denoted by "w". This becomes important when an indefinite object is assigned an identity in some context with the operator "<-". Objects defined in terms of indefinite objects are indefinite. For example:

n := a number (50)

n-and-3 := n + 3 (51)

"n + 3" is a definite number only in a context where "n" is a definite number.

5. Extending the Operator Semantics

So far we have seen that given any class, there are essentially four operations defined on it (that do not create new classes). They are:

- add instances
- remove instances
- test for instances
- fetch instances

We have also seen how these operators have standard prerequisites and side-effects. Consequently, the semantics of a class are determined by its behaviour under its defined operations.

Extending the basic semantics of a class involves specifying special cases of prerequisites, effects and values when applying these operations to the class. This is done by relating the class to programs (one for each operation) which are then

interpreted automatically when applying the corresponding operator. In this sense, our approach is procedural. When no program is specified for an operation, the program of the superclass of the class can be used. In this case, a class inherits semantics along the ISA HIERARCHY.

Programs are definite objects that can be interpreted. We can divide programs into three subclasses: procedures which perform actions (for adding and removing instances), predicates which test conjectures (for testing instances) and functions which have values (for fetching instances). All programs can have preregs which are conjectures tested before the "body" is attempted. A false causes the program to fail. In addition, programs can have effects which are actions performed after the successful completion of the body. To relate a class to a program, we will use four primitive relations: to-add, to-remove, to-test and to-fetch. For example,

```
to-test: male ->
     program with
          test := sex : instance ? masculine
     end                                                    (52)
```

reduces a test for an instance of "male" to a test for masculine sex. Thus if we write "jim :-> person" and "sex : jim -> masculine", then "jim ? person" is true and "jim ? male" is true as well, since the above program will be interpreted with the built-in parameter instance assigned "jim" (i.e., "instance <- jim"). Similarly, if we have (using example (23))

```
product :=> object                                          (53)
cost :=> must-hold with domain <- product, range <- number
                                                            (54)
price :=> must-hold with domain <- product, range <- number
                                                            (55)
profit :=> must-hold with domain <- product, range <- number
                                                            (56)
```

to express the semantics of "profit" we write:

```
to-fetch : profit ->
     program with
          value := price(domain-inst) - cost(domain-inst)
     end                                                    (57)
```


Thus when evaluating "profit(w)" where "w" is a product, the above program is used with "domain-inst <- w".

If we define "spouse" (using(22)) as

```
spouse :=> inter-personal with d-interval <- <0,1>,
          r-interval <- <0,1>                               (58)
```

then to express the fact that the semantics of "spouse" is such that it can only hold between persons of opposite sex and it is symmetric we can write:

```
to-add : spouse ->
     program with
          prereq := ¬(sex(domain-inst) = sex(range-inst))
          effect := spouse : range-inst -> domain-inst
     end                                                    (59)
```

Here, the "prereq" is specified but the body (i.e., action) is not. This means that the action is inherited from "inter-personal" (see example (22)). Thus the action is the standard action of adding to an (inter-personal) relation. We can also refer to the standard action explicitly by std.

In addition to built-in parameters such as std, self, and instance, parameters can be associated explicitly to a class operation.

```
to-add : student ->
     program with
          num := a number
          dept := a department default dcs
          effect := do
               student-number : instance -> num
               student-department : instance -> dept
          end
     end                                                    (60)
```

For this program, "num" and "dept" are explicit parameters which can be assigned values every time an instance of "student" is added (see example(17)). We now present a program with loops that will serve to generate "uncles" of a person (assuming "parent" and "brother").

```
uncle :=> inter-personal                                    (61)
to-fetch : uncle ->
     program with
          value := for p <- parent[domain-inst]
```

```
                    for b <- brother[p]
                        return b
                    end
                end
            end                                          (62)
```

6. Structures

For various reasons, it is convenient to be able to treat groups of objects as units. Such units are called "structures" and the objects that constitute them, their "parts". Structures have properties not necessarily derivable from the properties of their parts (i.e., a gestalt). In fact, any object (as seen so far) can be considered as a structure with no parts. Thus a structure is a group of other structures. We call this organization of parts the "PART-OF HIERARCHY". The syntax we will use for the definition of structures is:

```
structure some-object with some-parts end                (63)
```

For example,

```
vector-1 := new                                          (64)
structure vector-1 with
    polar-coords := new
        structure polar-coords with
            angle := 45
            radius := 1.414
        end
    x-y-coords := new
        structure x-y-coords with
            x:= 1
            y := 1
        end
end                                                      (65)
```

defines a structure "vector-1" having as parts two new objects which are in turn structures having two abstract objects as parts. To refer to the "polar-coords" parts of "vector-1", we write "vector-1.polar-coords". Note that the above structure provides two views of the same object and that these views can be organized in many different ways depending on the emphasis desired.

When a structure A is a subclass or instance of a structure B, unless otherwise specified, A inherits the parts of the B. For example:

```
vector :=> object                                        (66)
structure vector with
    angle := a number suchthat (self >= 0 & self < 360)
    radius := a number suchthat self >= 0
end                                                      (67)
normalized :=> vector with radius <- 1                   (68)
```

Now if we write,

```
vector-2 :-> normalized with angle <- 30                 (69)
```

"vector-2.angle" is 30 and "vector-2.radius" is 1. Note that there is a difference between

```
vector-a :-> vector with radius <- 2                     (70)
```

and

```
vector-b :=> vector with radius <-2                      (71)
```

even though both have the same radius and angle (2 and unknown respectively), in that (70) asserts the existence of some (indefinite) vector whose radius happens to be unknown at the moment, while (71) defines a class of vectors that may or may not have instances.

One important feature of structures, is that they provide a way of declaratively specifying often used programs. For example, we can think of testing whether a structure is an instance of another structure (to-test) as a very general matching procedure that attempts to find matching correspondences between parts in each structure. We can therefore place these programs very high in the ISA HIERARCHY where they can be inherited by lower, more specific classes whose structure will determine their operation. Of course, if this type of processing is to be meaningful, the structures will have to be more general than those presented here. In particular, they will have to contain instances of relations, default mechanisms and various prerequisites and effects to be interpreted at appropriate times, to guide the processing and handle troublesome situations.

7. Conclusions

The ideas presented in this paper are adaptations from a number of sources. The original motivation is due to Abrial who led us tc consider a coherent self-describing formalism for a representation. An obvious but important influence was the semantic network literature which reinforced the idea of objects and links as basic building blocks cf the model. The idea of associating programs to objects as their definition is clearly related to the ACTOR notion of a distributed interpreter. The prerequisite and side-effect portions of a program correspond to the consequent / antecedent distinction of PLANNER, while the division of processing into four basic operations is a generalization of the three methods of CONNIVER. The idea of higher level structures is a beginning in the direction of "frames" with more than a syntactic influence from Bobrow and Winograd's KRL. Finally, the influence of SIMULA is evident in our concept of classes.

The formalism described here is incomplete, especially for programs and structures. Some unanswered questions are:

How does one instantiate a structure or match two structures? What is a context? How do programs "execute" or "compile"?

We hope that we have at least given an indication of how these may be handled. The answers will be formulated in terms of the constructs that have already been described and used. In this respect, the formalism, like LISP, is completely open-ended.

[1] Abrial, J.R., "Data Semantics", Data Management Systems, ed. by Klinhie and Koffeman, North Holland, 1974.

A DEMONSTRATION LANGUAGE COMPREHENSION SYSTEM (1)

John W. Ball, Liam J. Bannon and Mike M. Marmor (2)

University of Western Ontario, London, Canada

Abstract

This paper describes a demonstration natural language understanding system, developed as a class project. In the course of a few months, an implementation was constructed which could handle reasonably complex interrogative and imperative English sentences within a limited domain - a blocks micro-world. An ATN grammar was used in the parsing of input sentences, and the advanced facilities offered in the POPLER 1.5 system were utilized in the construction and manipulation of the world model. Several innovative features of our comprehension system are discussed, including a novel solution to the problem of relative clause comprehension.

Setion 1 Introduction

This paper developed out of a class project on language comprehension in a joint psychology/computer science half-course, under the direction of Dr. Zenon Pylyshyn at the University of Western Ontario. A demonstration language comprehension system was developed which performed adequately in a limited task domain. This paper outlines some of the major aspects of our system, its advantages and its limitations. It should also be noted that designing and implementing this system served as a most useful introduction to many of the fundamental problems of A.I. research on language comprehension, and we stress the beneficial pedagogic aspects of such a course design, i.e., a course which is project oriented.

The fact that anything of substance could emerge from such a project in a short space of time, reveals the rapid advances which have occurred in the A.I. field in the last few years. Such components as the ATN formalism, and the POPLER 1.5 system (3) (Davies, 1973), gave us a much needed basis for our

work, without which little could have been accomplished. An outline of the system is given in Section 2 below. Following this, some general remarks on the limitations of the system are discussed. A sample of output from the system is given in Appendix i.

Section 2 System Components

The run-time system occupies between 84K and 100K of core (including 45K for POPLER) on our PDP-10, depending on the length of the input sentence. The system may be conveniently divided up into three sections corresponding to the parsing system, the semantic routines, and the world model.

2.1 The Parser

The specific grammar used in our implementation is a modification of the ATN grammar constructed at U.B.C. (Jervis, 1974). The grammar was written in POP10 code (Blewett, 1974). Several modifications of the grammar were required, in order for it to run successfully in POP10.

A lexicon was developed, tailored for the "blocks" micro-world which we had decided upon as our task domain. An example of an entry in the lexicon is given below:

[arm n s kywd hand]

This states that the lexical item 'arm' is a singular noun whose keyword is 'hand'. The lexicon performs the mapping from a lexical item (e.g. arm) onto a keyword (e.g. hand). The keyword is always something which is significant to the blocks world, whereas the lexical entry might not be significant. This allows vocabulary growth without a corresponding growth in keywords.

As we build the parse fragments for noun and prepositional phrases, these semantic fragments are not interpreted. The interpreation phase is postponsed until the parse is finished and then the complete sentential form is evaluated. This strategy was decided upon for practical reasons

which we will amplify later. In retrospect, we found this procedure to be costly in terms of searching the data base, and we now hold that evaluation of the semantic fragments should occur during the parse itself, in order to prune the search tree as soon as possible.

An interrupt facility was programmed which can be used for a variety of purposes during the parsing stage, e.g. recognition of idiomatic expressions, punctuation, replacement of equivalent expressions, and various control functions.

A final development of the parsing system, which is not yet fully debugged, involved the design of a compiler-translator for ATN's which compiled an ATN grammar into POP10 source code. Each node-list was translated into a POP10 function definition, with the function name being the node name. Each arc list and sublist was translated into a call to a POP10 function contained in the parser's runtime system. The result was a 60 percent reduction in the space occupied by the ATN, as well as a slight reduction in execution time. It is interesting to note that the idea of compiling an ATN also occurred independently at another centre at this time (Burton and Woods, 1976).

2.2 The Semantic Routines

The semantic routines interface the parser with the blocks world. They are called by the parser at the noun phrase, prepositional phrase, and sentence levels, and they have the opportunity to fail and parse which is passed to them at any of these levels. After a sentence is interpreted by the semantic routines, the resulting interlingual representation of the sentence is placed in the POP10 editing buffer which serves as a communication medium between the parser-semantic routines and the world model. Code is added to run the interlingua in a marker frame to which a direct failure will be sent in the event that the interlingual form is uninterpretable in

the world model. Compilation of the buffer then initiates activation of the world model. If the blocks world is unable to understand the input, a failure is passed back up to the parser, and a new parse is attempted. Eventually, either the sentence makes sense in the world and is executed, or the parser cannot find any more acceptable parses and fails.

There are three main parts to the semantics:

(a) the replacement of terms by their keywords

(b) the translation of noun phrases into a set of constraints

(c) the construction of sentence level interpretations which could evoke procedures in the data base.

The interlingua generated by the semantic routines and input to the blocks world is very readable and often similar on the surface to the original English sentence, e.g.:

(a) pickup the large red block behind the pyramid

(b) (ACHIEVE[GRASP[(THE)(LARGE)(RED)(BLOCK)(BEHIND[(THE)(PYRAMID)])]])

Objects are characterized by stringing together constraint lists of actor forms. There is a special actor form for "the" which involves more complicated processing than the majority of actors because of its implied anaphoric reference and will be discussed further in 2.3.

By building our semantic representation (or interlingua) around constraint lists of actors, we achieve a simple first approximation rule of composition; viz the semantic representation of a constituent is obtained by concatenating the representations of its subconstituents (e.g.: [(LARGE)(RED)(BLOCK)]). This rule remains approximately the case up to the level of the clause, although some special considerations had to be taken into account. For example, to make this principle hold in the case of prepositional phrase, we had to make the effect produced by actors associated with prepositions depend on the context in which it occurred. For instance,

the actor (BEHIND[..]) functions differently in (1) and (2):

(1) (ACHIEVE[MOVE[(BLOCK)(BEHIND[(BOX)])]])

(2) (ACHIEVE[MOVE[(BLOCK)] [(BEHIND[(BOX)])]])

In (1) MOVE has only one argument so it interprets that argument as an object constraint list. Thus behind functions as a conventional restricting variable-assigning actor. In (2), however, the second argument to the MOVE function is interpreted as a constraint on locations and returns a location rather than an object in the blocks world.

This convenient uniformity could not be extended to include relative clauses, however. The reson is that whereas qualifying prepositional phrases always act as one-argument functions constraining the referent of the head noun in the dominating noun phrase, relative clauses are more complex in their behaviour. In fact, relative clauses have sentential forms in their underlying structure and the noun phrase being constrained can be referred to in any nominal position in the embedded sentence. Consider the following cases:

(3) the block which supports a cube

(4) the block which is supported by a cube

(5) the block which the pyramid is on.

In (3) the embedded sentence (i.e. relative clause) constrains its subject. In (4) its object, and in (5) the object of the preposition. Thus we need to indicate that a constraint is being imposed on X where X in each case is as in

(6) X supports a cube

(7) a cube support X

(8) the pyramid is on X.

Further, we want to restrict X to be filled by an object also meeting the constraint (THE) and (BLOCK). Since the parser properly interprets relative clauses such as (3) - (5) as embedded sentences such as (6) - (8) with X's filled in by "the block"), the simple rule of composition would not work. Instead, a device similar to lambda binding was employed which picks out from the semantic structure of the relative clause that part which is to be further constrained by the actors outside the clause. The device consists of the pair of actors (SUCHTHAT[..]) and (THATTHING) serving as declaration and variable respectively. Thus (3) - (5) after being parsed in terms of embedded sentences such as (6) - (8) are translated to (9) - (11) respectively.

(9) [(THE)(BLOCK)(SUCHTHAT[(THATTHING)(SUPPORTS[(CUBE)])])]

(10) [(THE)(BLOCK)(SUCHTHAT[(CUBE)(SUPPORTS[(THATTHING)])])]

(11) [(THE)(BLOCK)(SUCHTHAT[(PYRAMID)(ON[(THATTHING)])])]

as with the prepositional phrases, such structures are constructed recursively and can be indefinitely embedded.

2.3 The World Model

The micro-world is a simulated blocks world similar to that used by Winograd. The 3-D space of the blocks world is conceptually divided into distinct compartments, each compartment being a 10-unit cube. Objects occupy separate compartments in the world.

The knowledge of the blocks world consists of entities and processes. Each entity is a uniform symbol structure, represented as a set of attribute-value associations. A process is a procedure of the system which is elicited in the presence of a specific input stimulus - in this case a POPLER-compatible interlingual representation of an English sentence. The behaviour of the process may be a simple retrieval of a fact from the symbol

structure or a change of the content of the structure in response to an altered state of the world. There are more than 40 actors defined to allow for descriptions of objects in terms of their properties and relative locations.

An attempt was made to handle the problem of anaphora. Since all references to objects in the blocks world are extensional (except for 'one', described later), all noun phrases must be instantiated to a particular object. An anaphoric reference list (a stack of previously mentioned objects) is created to aid in this instantiation. When the special actor 'the' is encountered, it is assumed that the user is referring to a specific item in the world. If it is unique in the present world state, then no disambiguation is necessary, otherwise the anaphoric reference list is examined to attempt to individuate the reference. If the current context defined by the discourse-specific knowledge (i.e.: the anaphoric reference list) cannot effect the disambiguation, then a failure is sent out of a marker frame (originally set up in the buffer) back to a decision node constructed by the semantic routine in the S/ node of the parser, where another possible interpretation will be attempted. Two other actors, 'it' and 'one' are also allowed in the input string and their references are found by use of the anaphora mechanism. The actor 'one' is unique in our world as it is the only actor with intensional import in that it can refer to a class of objects rather than a specific object.

It should be noted that the world model performs some important semantic and syntactic checking in addition to the more pragmatic interrogation and maintenance of the data base itself, (e.g.: 'put the block.', or 'is the blue block?', though parsed as grammatical by our grammar rules, will fail).

Section 3 Conclusion and Discussion

The decision to postpone accessing the blocks world until the end of the parse phase, mentioned earlier, was an expedient. After the parse has produced a noun phrase, we have a semantic fragment available which could be evaluated in the micro-world. If it were meaningless, a backtrack in the parse could begin immediately, rather than having to wait until the end of the sentence parse.

The semantic checks made in our semantic routines are rather elementary and could be upgraded. The addition of case frames would probably increase the efficiency of the system. They are not used in our system, as the world model itself acts as a partial case frame filter. However, it would be less time consuming if these checks were done before entering the blocks world.

Despite the shortcomings mentioned above, we believe the system as it stands is a useful tool for the investigation of the problems of language comprehension by machine. It is hoped to continue work on the system next year.

Footnotes

(1) We would like to acknowledge the assistance that we received in this project from Zenon Pylyshyn, our instructor who provided the impetus for the whole undertaking, from Julian Davies for assistance with some technical details concerning the POPLER 1.5 system, and from Richard Rosenberg of U.B.C. who provided us with a LISP copy of both a parser and a grammar which served as a basis for the current project. We are grateful to Zenon Pylyshyn for his useful comments on a draft of this paper.

(2) This paper describes a course project involving work done by Gary Duggan, Dave C. Hogg and John McArdle in addition to the authors.

(3) POPLER 1.5 is a high-level A.I. system designed by Julian Davies at The University of Edinburgh. POPLER is a language based on the main features of PLANNER and CONNIVER, and is embedded in the PDP-10 system (a PDP-10 implementation of POP2).

References

1. Blewett, W.J., Semantics of English sentences for simple arithmetic using a recursive augmented transition network grammar. M.Sc. Thesis, University of Western Ontario, May, 1974.

2. Burton, R.R. and Woods, W.A., A Compiling System for augmented transition networks. In 6th International Conference on Computational Linguistics (Preprints). Ottawa, Canada, June 1976.

3. Davies. D.J.M., POPLER 1.5 Reference Manual. T.P.U. Report #1, Theoretical Psychology Unit, School of Artificial Intelligence, University of Edinburgh, May, 1973.

4. Jervis, J.E., An Augmented transition network for English. M.Sc. Thesis, University of British Columbia, May 1974.

5. Winograd, T., Understanding Natural Language, New York, Academic Press, 1972.

```
TYPE HI! (IN PLANNER MODE) TO BEGIN.

* hi!
.....................................
GO! where is the red pyramid ?

   EVALUATING :
      (INFER (LOCATE E*(THE) *(RED) *(PYRAMID)))

P1 IS AT [ 20 20 0]

RUNTIME (MILLIS) : 4539.....................................
GO! where is the red block ?

   EVALUATING :
      (INFER (LOCATE E*(THE) *(RED) *(BLOCK)))

   I DO NOT KNOW WHICH [(RED) (BLOCK)] YOU MEAN.

   SORRY.

RUNTIME (MILLIS) : 3445.....................................
GO! what is red ?

   EVALUATING :
      (INFERFORALL E*(RED))

[EACH OF [B4 B1 P13 SATISFIES [(RED)]]

RUNTIME (MILLIS) : 4080.....................................
GO! is the pyramid short ?

   EVALUATING :
      (YESNO [E*(THE) *(PYRAMID)] E*(SHORT))

 BY [(THE) (PYRAMID)]
 I ASSUME YOU MEAN WHAT YOU REFERRED TO AS [(THE) (RED) (PYRAMID)] (P1).

P1 IS [(SHORT)]

 YES.

RUNTIME (MILLIS) : 5214.....................................
GO! is a pyramid which is blue on the blue block ?

   EVALUATING :
      (YESNO [E*(PYRAMID) *(SUCHTHAT E*(THATTHING) *(BLUE))]
         E*(ON E*(THE) *(BLUE) *(BLOCK)))))

P4 IS [(ON E*(THE) *(BLUE) *(BLOCK))]

 YES.

RUNTIME (MILLIS) : 25192.....................................
GO! get something which is under a pyramid.

   EVALUATING :
      (ACHIEVE (GRASP E*(SOMETHING) *(SUCHTHAT E*(THATTHING)
         *(UNDER E*(PYRAMID))))))

NOT GRASPING ANYTHING NOW.

P5 GRASPED.

OBJECT P5 SET AT [ 110 40 0]

P4 GRASPED.

RUNTIME (MILLIS) : 15654.....................................

GO! is the blue block on the table ?

   EVALUATING :
      (YESNO [E*(THE) *(BLUE) *(BLOCK) *(ON E*(THE) *(TABLE))]))

   SORRY.

   EVALUATING :
      (YESNO [E*(THE) *(BLUE) *(BLOCK)] E*(ON E*(THE) *(TABLE))))

B2 IS [(ON E*(THE) *(TABLE))]

 YES.

RUNTIME (MILLIS) : 21143.....................................
GO! put the green one on the table in the white box on the table.

   EVALUATING :
      (ACHIEVE [PUT E*(THE) *(GREEN) *(ONE) *(ON E*(THE) *(TABLE)
         *(IN E*(THE) *(WHITE) *(BOX) *(ON E*(THE) *(TABLE)))]))))

   SORRY.

   EVALUATING :
      (ACHIEVE [PUT E*(THE) *(GREEN) *(ONE) *(ON E*(THE) *(TABLE)
         *(IN E*(THE) *(WHITE) *(BOX))))] E*(ON E*(THE) *(TABLE))))

   I ASSUME YOU MEAN BLOCK

   SORRY.

   EVALUATING :
      (ACHIEVE [PUT E*(THE) *(GREEN) *(ONE) *(ON E*(THE) *(TABLE))]
         E*(IN E*(THE) *(WHITE) *(BOX) *(ON E*(THE) *(TABLE)))))))

   I ASSUME YOU MEAN BLOCK

NOT GRASPING ANYTHING NOW.

B3 GRASPED.

B3 MOVED TO [ 100 100 10]

RUNTIME (MILLIS) : 36578.....................................
GO! is a block which is a block which is green green ?

   EVALUATING :
      (YESNO [E*(BLOCK) *(SUCHTHAT E*(THATTHING) *(BLOCK) *(SUCHTHAT
         E*(THATTHING) *(GREEN))))] E*(GREEN)))

B3 IS [(GREEN)]

 YES.

RUNTIME (MILLIS) : 14209.....................................
GO! is it the green one ?

   EVALUATING :
      (YESNO [E*(IT)] E*(THE) *(GREEN) *(ONE)))

       BY 'IT' I ASSUME YOU MEAN
       WHAT YOU REFERRED TO AS [(THATTHING) (BLOCK) (SUCHTHAT
         E*(THATTHING) *(GREEN))))] (B3)

    I ASSUME YOU MEAN BLOCK

 B3 IS [(THE) (GREEN) (ONE)]

  YES.

 RUNTIME (MILLIS) : 4915.....................................
```