EXPRIENCE WITH KRL-O
ONE CYCLE OF A KNOWLEDGE REPRESENTATION LANGUAGE

Daniel G. Bobrow, Terry Winograd,
and the KRL research group'

The projects and implementation described in this paper were done at Xerox Pah) Alto Research Center. Palo Alto, California by Dan Bobrow, Ron Kaplan, and Martin Kay from Xerox PARC; Jonathan King. David levy, Paul Martin. Mitch Model, and Terry Winoj-rad from Stanford; Wendy I .ehncrl from Yale; Donald A. Norman from *V.C.* San Diego; Brian Smith fiom MIT; and Henry Thompson from U.C Berkeley.

The goal of the KRL research group is to develop a knowledge representation language with which to build sophisticated systems and theories of language understanding. This is a difficult goal to reach, one that will require a number of years. We are using an iterative strategy with repeated cycles of design, implementation and testing. An initial design is described in an overview of KRI (Bobrow & Winograd, 1977). The system created in the first cycle is called KRL-o, and this paper describes its implementation, an analysis of what was learned from our experiments in using KRL-o, and a brief summary of plans for the second iteration of the cycle (the KRi.-i system). In writing this paper, we have emphasized our difficulties and disappointments more than our successes, because the major lessons learned from the iterative cycle were in the form of problems. We mention only briefly in the summary of experiments those features of Krelo that we found most satisfactory and useful.

In order to put our experiments in some perspective, we summarize here the major intuitions we were testing in the design of KRL-O:

1. Knowledge should be organized around conceptual entities with associated descriptions and procedures.
2. A description must be able to represent partial knowledge about an entity and accommodate multiple descriptors which can describe the associated entity from different viewpoints.
3. An important method of description is comparison with a known entity, with further specification of the described instance with respect to the prototype.
4. Reasoning is dominated by a process of recognition in which new objects and events are compared to stored sets of expected prototypes, and in which specialized reasoning strategies are keyed to these prototypes.
5. Intelligent programs will require multiple active processes with explicit user-provided scheduling and resource allocation heuristics.
6. Information should be clustered to reflect use in processes whose results arc affected by resource limitation and differences in information accessibility.
7. A knowledge representation language must provide a flexible set of underlying tools, talher than embody specific commitments about cither processing stiatogies or the representation of specific areas of knowledge.

Some *of* these intuitions were explored in GUS (Bobrow, et al, 1977), a dialog system for making airline reservations. GUS used ideas of procedural attachment (Winograd, 1975), and context dependent description (Bobrow & Norman, 1975). Experience with GUS led to some changes to our ideas for KRL-o, although GUS and KRL-O were basically concurrent projects; we started programming GUS just prior to intensive design on KRI-o. The GUS system was primarily an attempt to explore the integration of already existing programming technology for a performance demonstration, while KRL-o was a first attempt at outlining a new basis for representation.

I. Building the KRL-O System

KRI -o was implemented in JNTERI.ISP (Teitelman, 1975), along the lines described in Bobrow and Winograd (1977). The design was specified mostly during the summer of 1975. The initial KRI-o implementation was programmed primarily by Bobrow, Levy, Thompson, and Winograd during December and January, with parts of the development being done by the rest of the KRL group. It included basic structure manipulating facilities, a reader and printer for KRL structures, a simple agenda manager and scheduler, a procedure directory mechanism, and a matcher which handled only the most elementary cases. Many more pieces were built into this system by people working on the test projects over the following (» months. The system was first implemented on the MAXC computer at Xerox PARC and later transferred to the SUMI.-X PDP-10, (where one of the projects was done as an AIM Pilot project), and to the IMSSS PDP-10 at Stanford. When the test projects were complete, the system was retired from active duly.

As an experimental system, there was no commitment to continue support after the initial purposes were satisfied. Despite its avowed experimental natuie, however, building KRL-o was a major system programming effort; programming any "new AI language" for users is larger task than just trying out the new ideas. Having the many facilities *of* INILRITSP to build on eased our programming burden, but a number of new facilities were built for the project:

- ► Ron Kaplan developed a set of utilities, including special data structure manipulation and formatted printing routines, as a base for much of the implementation. The entire utility package (called USYS) was interfaced so smoothly that the user could think of it as simply an extended INTLERSP This package will be used in the development of KRL-1.
- ► An on-line cross-reference and documentation system (called the N.Utls system) was used to coordinate the efforts of the people doing interactive debugging of a shaicd set of programs. The facility was designed and built by Ron Kaplan and Martin Kay. It communicated with the editor and file package

facilities' in INTERLSEP so that the. rrogiammcr was prompted for a comment whenever programs or nroiJ declarations were created or edited. The information available to the system (e.g. procedure name, variable names, etc.) was combined with user supplied comments in a standardized data base which could be interrogated on line. The programmer was automatically warned of potential naming conflicts with anything anywhere else in the system. It also provided facilities for entering comments associated with global variable names and file names. The file of names grew to contain over 1000 entries during the course of implementing KRL-o. Ior the KRL-1 implementation we are extending the interface to work with Masterscope. the INTLRITSP cross-reference and program analysis package written by Larry Masinter.

- A simulated match interface was built by Paul Martin, which enabled the programmer to intercept calls to the matcher and gather data on what kinds of problems came up before programming the necessary extensions. The user returned an answer for the match, and on future identical matches the same answer was used.
- A tracing facility for the matcher was implemented by Jonathan King, to facilitate debugging *of* programs which were organized around matching

As problems came up in using KRL-o, they were handled in several ways. Those which seemed general and could be handled within the existing framework were set up as tasks tor the KRL-o programming effort. Usually design discussions were shared by everyone, and the implementation clone by the person whose program faced the problem. 'Those problems which were either too specialized or obviously beyond the scope of our current design were programmed around by the problem-finder. Most of these cases led to changes in the KKI-1 design to accomodate solutions more naturally. Because ;<RL-() was embedded in INTERSP, "patching" was usually straightforward in that it was the same as what would have been involved in trying to write the program in a bare INTERIEP in the first place. Of course, sometimes these "patches" interacted with other parts of the Kin. code in unpredieted and confusing ways. "I hose problems for which there was no acceptable way to escape were chalked up It) experience, and the goals of the program reduced accordingly. Usually this was in cases where there had been an unresolved question as to how much.li the program should be expected to handle. Issues raised by these problems were a major driving force in t he KRL-1 design.

A very re ugh draft of a manual was distributed, but became rapidly obsolete as the system evolved. It was highly incomplete (for example, the section on the matcher consisted of a single paragraph describing why the section was going to be difficult to write). It was never completed or re-edited, and those doing the programming had to rely on discussion with the implemented and on the source code of the interpreter for up to date information. It worked reasonably well, with some frustration, but not enough so that anyone ever felt moved to volunteer the time to do the writing needed to produce a real manual and keep it current. We were somewhere around the upper bound of the size of project (number of people, amount of programming) where so informal an approach was feasible.

2. Experiments using KRL-o

KRL-O notation and programs were tested in nine different small projects. Each of these projects was intended to test some aspect of the KKL-0 language or system. They took from 3 to 15 person-weeks of effort each. In most cases, the goal was to produce an actual running program which could handle enough examples to convince us that it did what the original program was intended to. In no case was an effort made to do the kind of final debugging and polishing which would make the program robust or usable by anyone but the original author. We will describe three of these in detail: a cryptarithemetic problem solver; a story analysis program; and a medical diagnosis system. We list below the other projects that were done to give a flavor of the range of projects tried:

- LLGAL -- done by Jonathan King -- an implementation of a portion of a legal reasoning system sketched by Jeffery Meldman (1975) in his doctoral dissertation. This program forced consideration of matching in which both patterns and data could specify bindings that were needed.
- ARCH IS -- done by Paul Martin -- a concept learning progam based *on* Patrick Winston's (1975) program for recognizing visual scenes. Matching sets of descriptions, and the use of instances as patterns were the interesting parts of this project
- COIL -- done by Wendy l.ehnert -•- a new program for drawing inferences about objects, based on methods related to those of conceptual dependency. This program used the contingent description mechanism to select knowledge to be used in a particular context, and the agenda lo interweave syntactic and semantic processing, of input English.
- FLOW -- done by Dan E'obrow and Don Norman -- a program sketch whieh .simulated a person's access to long term memory while using, a recently learned simple computer language. The indexing mechanism of KKL was used lo simulate propel lies of human associative retrieval (including, errois of various kinds).
- PIIYSIOI.OCJY — done by Buan Smith — a program sketch which explored the problems of using KRl o for a system which could reason about physiological processes. This project forced consideration of the gaps in KRL-O with respect to specifying temporal and causal structures, and the need for stronger structuring to factor information in units by viewpoints, e.g., information about the heart as viewed as a mechanism, versus information when viewing it from an anatomical perspective.
- KINSHIP -- done by Henry Thompson — a theoretical paper, using the KRL-O notation as a basis for comparing kinship terms in English and Sherpa. The attempt to communicate results of encoding to non-computer scientists led to a simplified notation which has contributed to the syntax for KRL-I.

Cryptarithmctic

The initial test program was a simple cryptarithmctic problem solver (see Newell and Simon, 1972 for a description of the domain) written by Terry Winograd and debugged and extended by Paul Martin. It exercised the basic data structures, agenda, and triggering facilities, and was successfully tested on several problems (including DONALD ♦ GERALD = ROBERT with D=5). No attempt was made to provide complete coverage of the class of problems handled by humans. Interesting aspects of the design included:

- Use of triggers to combine goal directed and data directed processing

- ► Use of "patterns" to suggest strategics
- ► Use of levels on the agenda to control ordering of strategics
- ► Use of multiple descriptors to accumulate information about the value of a letter
- ► Use *of* contingencies to handle hypothetical assignments
- ► Use of the signal table to control work within hypothetical worlds

Much of the processing was associated with procedures attached to the units for Column (a vertical column in the addition problem) and Letter. The Unit for Column is given below. It gives some idea of the use* of procedural attachment to propagate information, search for patterns sm h as a column with Two Blanks and trigger arithmetic piocssing (using the i.isi* function ProeessColuajn).

```
{COLUMN UNIT Basic
<SELF [ ]>
<LeftNeighbor (a Column)>
<RightNeighbor (a Column)>
<topLetter (a Letter)>
<bottomLetter (a Letter)
   (triggers (WhenKnown
                (DoWhenknown (topLetter) Column
                      (ClryToFuthers SpecifyUNIT
                           '(TwoBlanks OneBlank TwinAddend)
                           'Addcnd'l'ype]>
<suml,cttcr (;i Letter)
   (triggers (When Known
                (DoWhenKown (topl.ettcr hotomLettcr) Column
                      ((.Check SumEqual Addend UNIT]>
<lopnicif (;t Digit)
   (triggers (VVhenknown (Assign 'topletter (Process Column)))>
OmtinmDigit (;i Digit)
   triggers (Whenknown (Assign 'ho!toml.etter)(Process('aluiiin)))>
<sumldigit (a Digit)
   (triggers (Whenknown (Assign 'suml,etter)(I'roeessColunin)))>
<sum { (can Integer)
      (which IsStiinOf
           (Alllteitis (Ilie Ciirryln) (the |npDigil)(Ilie holtoniDigit)))}
    (triggers (Whenknown (Process Column)))>
<c;iiryln { (an Integer)
           (XOR 0 l)
           (the carry Out from Coluuman(the righINeigh)mr))(; CARRYOUT)}
   (triggers (Whenknown ((GoFill '( ARRN <-)Ul )(l»rocess( Coluum)))>
<carryout it { (;in Integer)
           (XOR 0 l)
           (The carryin from Column (the IrftNclghhor)) (; CARRYIN)}
   (triggers (WhenKnown (GoFill ( ARRYIN) (I'nnessi oliimn)))> |
```

There was a set of recognized patterns for columns (for example, a column with the sum letter identical to one of the addends) and a set of pattern driven strategies was associated with each. Each strategy was a l l'.c procedure which used the KRl-o structures only as a data base. Some of the strategies caused values to he computed. Whenever a new value was filled into a column, triggers caused data driven strategies to be suggested, such as trying to bound the possible value of other letters based on this information. Constraints on values were added in the form of new descriptions for the value of the letter, for example specifying that the value must be an even or odd integer, Laeh such description was added to the existing description *of* the value *of* that letter, so that at any point in the compulation, some letters had a value described as a specific digit, while others had complex desciptions, such as "Greater than 3 and odd". Each time a new description was added, a trigger in the unit for Letter caused a procedure to be run which matched each still-unassigned digit against the accumulated description, and if only one matched, it was assigned.

When new strategies were suggested by a new value being tilled in, or by the match of one of the patterns describing columns, all of the triggered strategies were put onlo the agenda. They were assigned priority levels on the basis of a fixed scheme: Level l was immediate propagation of information (e.g. if the value of a letter is determined, then that value gets entered into all of the places where the letter appears). Level 2 was for straightforward arithmetic computations. Level 3 for the strategy being worked on currently, Level 4 for other simple strategies, Level 5 for more complex and less likely strategies. Level 6 for last-ditch strategies (brute force trial and error) and Level 7 contained a single entry which caused the problem to be abandoned.

This rather *ad hoc* use of agenda levels achieved a number of goals. The use of Level 1 foi simple propagation served as a kind of data locking scheme to maintain consistency. As long as there were more results to be propagated, no other part of the program would run. This meant, for example, that if some letter were assigned to a digit, no other letter could be assigned to the same digit before the result had been properly recorded. The use of a separate level for the current strategy allowed it to trigger sub-strategies without getting put aside for work on a different strategy. Ibis meant that each strategy could run to completion. The use of levels to distinguish how promising different strategies were allowed the system to focus its effort on whatever were the most likely things at the moment. Placing last-ditch strategies on lower levels when they were thought of made it easy for the program to fall back on them -- they automatically ran if nothing at any higher priority was scheduled. This provided a weak global structuring in what was inherently a data-driven process.

The mechanisms for multiple worlds and contingent descriptors made it possible to deal with hypothesized values while using the normal mechanisms. When all but two possible values had been eliminated for some letter, and no other strategies were pending, the program chose one *of* them, and created a hypothetical world, in which the letter had that value. Describing the letter as having that value hypothetically caused all of the same triggering as would nonconling.enl assignment of the value, leading to propagation of new information, computations, strategies, etc. However, by modifying the signal table, all derived information was asserted as contingent on that hypothetical world. This special signal table also affected the processing in two other ways: hirst, only simple strategies were allowed to be placed on the agenda. Second, it a contradiction occurred, the hypothesis was rejected instead of the problem being declared impossible. If a hypothec;, was rejected, the contingent descriptors were not removed, but would not be accessed by programs looking for descriptions in other hypothetical worlds, or in the world of actually inferred facts.

Sam

David levy implemented and tested a program which reproduced the simple text analysis and questioning aspects of the SAM program (Schank ct. al, 1975) which uses scripts in analyzing short "stories" containing stylized sequences of events. It used Ron Kaplan's GSP parser(Kaplan. 1973), and a grammar written by Henry Thompson for the initial input of the stories. It processed two stories (Schank, p. 12), summarized them and answered a number of simple questions. It was a full fledged language-processor in that it

took its input in English and generated English output. Questions were entered in an internal representation. Its main features were:

- ► Interfacing an existing parser (Kaplan's GSP) with a KRt,-o program which used the results of the parsing for further analysis
- ► Using slots to represent the basic elements (both events and participants) of scripts, and perspectives to represent instances of the scripts.
- ► Using the notion of "focus lists" as the basis for determining definite reference, including reference to objects not explicitly mentioned in the input text. It used the index mechanism to speed up search through the focus lists.
- ► Using the matcher in a complex way to compare story events to prototypical script events, with side effects such as identifying objects for future reference
- ► Using units describing lexical items and English grammatical structures as the basis for analysis and generation, using signals and procedural attachment

SAM's basic processing loop consisted of parsing, construction of conceptual entities followed by script lookup:

Parsing A sentence from the story was *fed* to GSP, which produced as output a surface syntactic parse identifying clauses, noun phrases, etc. as a KRt declarative structure. For example, for the sentence "John went to a restaurant" GSP produced the following rather shallow syntactic structure:

```
(» Declare with clause *
   (a Clause with
      surfaccl'orm = ".lohn went to a restaurant"
      verb = (A)
      subject = (a NounPhrase with
'                heed                - JO/IN)
      prcpPI = (a PrcpositionalPhr ase with
                          preposition - TO
                          object = (a NoiinlMirasc with
                                        head - RESTAUANTiK terminer - A))))
```

Construction of conceptual entities. The next step was to map this syntactic object into a set of conceptual objects with the help of declarative and procedural information stored in the prototypical syntactic units (Clause, NounPhrase, etc.) and in the lexical units. For example, the Clause unit specified that the filler of the verb slot would guide the mapping process for the entire clause, and the lexical representation of each verb included a case frame mapping from syntactic to conceptual structures. Following is a partial description of SAM's representation of the verb "go":

```
[(GO UNIT Individual
   <self {(a Verb with
              root = "Go"
              past * "went")
         (which IsAConslitucntOf
            (a Clause with
               referent =
                  (a Go with
                     goer = (the referent from Nounlphrase
                                 (The subject from (lause (a Clause)))
                     source = (the referent from NounPhrase
                                 (the object fror.i PreppositionalPhrrtse
                                    (a PrcposilionalPhrase with
                                       prepositioa = I ROM)))
                     destination = (the referent from NounPhrase
                                    (the object from PrcposilionalPhrase
                                       (a PrepositionalPbrase wits
                                          preposition - TO))))))}>]
```

As a description was created for each conceptual object (e.g. as it was determined that the appropriate Idler for the *goer* slot in the above example was (a Person with name = "Jolm")), this description was matched against a list of units in a *focus list* which contained the conceptual objects thus far created. It the description matched one of these objects, the slot was filled with a pointer to this object, and this object was moved to the front of the focus list. In order Lo make the search through the focus list faster, I he index faclilty was ased to find good potential matches from the list. If the description matched no object, a new object (a KRI unit) was created, the description was attached to it, and this objt.et was pushed onto the front of the focus list. In this way referents were established and maintained.

This scheme handled pronominal as well as definite reference, from the word "she", for example, the conceptual description (a I-eiualei't'isoa) was constructed, a description which would match the last mentioned reference (if any) to a female person (e.g. "the waitress").

Script lookup. Next the program tried to identify the conceptual event just created as a step in an active script. It did this by stepping through the script from the last event identified, and *matching* the description of this prototypical event to the event just created from the input sentence. This process exercised the KRt matcher rather heavily. Once the step in the script (represented as a slot) was identified, this slot was filled with the new conceptual event. In addition, any previous steps not explicitly filled by story inputs were then filled by creating conceptual events from the prototypical descriptions contained in the script. These events too were added to the focus list. The program also dealt with *what-ifs* or predictable error conditions, but these will not be discussed here.

The result of this iterative process was therefore the construction of a representation for the story consisting of:

- ► a set of syntactic units representing the surface syntactic form of the input sentences
- ► a set of conceptual units representing story objects: people, events (including inferred events), physical objects
- ► a focus list containing these objects
- ► a (partially) instantiated script, whose event slots were filled with the conceptual events in the focus list

Having analyzed a story, SAM could then summarize, paraphrase, and answer questions.

The different stages of processing in the analysis of inputs were controlled through the use of special signal tables, these tables provided special responses to the addition of descriplions to units. For example, the search for a referent was keyed by a signal set off by the addition of a perspective of type Noun Phrase. The generation process used a different set of signal tables to direct the inverse process of building a surface syntactic construction from a conceptual object. SAM was an interesting exercise in system construe!ion, useful mainly as a tool for understanding problems in representation and debugging KRI-o. When finished, it did not, and was not intended to, rival the power of the Yale group's original program.

Medical

Mitch Model implemented and tested a progiam for medical diagnosis based on a model for diagnosis which had not been dnectly implemented before (Rubin, 197.')). In writing the program, it was necessary to fill in a number of details, and correct some minor inconsistencies in the original. The program successfully duplicated, with some minor exceptions, the performance described for Rubin's hypothesized system. Part of the reason for the exceptions was incomplete specifications in Rubin's thesis, but there was also a major problem in that the implementation LISP code and data base completely filled the storage available in the KRL system. (This program, SAM, and con were the most extensive tests, and all ran into space problems discussed below). Some of the major features of the implementation were:

- ► The use of the abstraction hierarchy to represent the set of disease types and finding types, with information and procedures attached at different levels of generality.
- ► The use of KRL-o triggers to implement the conceptual "triggering" of potential diagnoses on the basis of having relevant symptoms described
- ► The use of signals to provide run-time monitoring of what the system was doing as it generated new hypotheses and evaluated them
- ► A direct encoding of the declarative "slices" of Rubin's version into the declarative forms of KRL-O. This included extensive use of the "Using" descriptor (a declarative conditional) to explicitly represent the decision trees in the units for diagnosing different conditions

There were four major kinds of representational objects in the system.

- ► "Elementary hypotheses" which corresponded to the "slices" of Rubin's thesis; these were named after the disease [c.g *Glomerulitis* or *Renal Infarction]* the data structure was intended to represent. Elementary hypotheses had descriptions in slots to indicate such things as likely symptoms, links to other elementary hypotheses that might be related, and how to evaluate how well the patients symptoms would be accounted for by a diagnosis of this disease.
- ► "Elementary hypothesis instances" were data structures created for each diagnosis the system decided might account for the presented symptoms; these contained pointers to the findings that suggested the diagnosis, and a pointer to the elementary hypothesis representing the disease of the diagnosis. It also contained values for how well the diagnosis accounted for the s>mploms$_f$ obtained by applying the evaluation information represented in the elemental') hypothesis to the specific details of the elementary hypothesis instance.
- ► "Findings" were units for specific symptoms, facts, hisloiical inhumation, phvstcal examination data, or lab data (e.g.. */■'ever, Hcnutinriii,* or *liinpsy)*\ a finding was mostly a hook on which to hang pioccdura) information about what to do when the patient exhibited soinel.hin» abnormal with respect to the particular kind of finding.
- ► Finding instances were the input to the system, having a structure similar to that Rubin suggested in her thesis, having slots for such things as finding, duration, severity, and normality. There were also further specified finding instances such as symptom instance.

The system worked essentially as follows. A unit might be described by:

```
(a SymplomInstance with
   main(Concept = Hematuria
   presence = "present"
   severity = "yross"
   time - (a TitmToint with
                direction z "past"
                magnitude = (a Quantity with
                                  unit - "days"
                                  number = 3)))
```

A *WhenKnown* trigger on the *presence* slot of the *Symptom Instance* prototype would be set off; examination of the specific description caused this entity to be described also as: (ii Symptoinhistancc with normality = "abnormal") Further triggers and traps might result in the creation of new elementary hypothesis instances, according to the information found in the description. After all the information propagation activity, each of the currently active elementary hypothesis instances would be evaluated based on information found in the corresponding elementary hypotheses. Based on the evaluation, the status of the elementary hypothesis instances might be changed to reflect possible dispositions of the hypothesis such as acceptance, rejection, or alteration.

The indexing facility was used to facilitate operations such as obtaining a list of all the hypotheses activated by a finding. Funclionals and *ToMatch* triggers on prototypes were defined to handle special time-related matches to enable the system to tell, for example, that "3 days ago" is more recent than "1 year ago" or that "48 hours" is the same as "2 days". Signal tables were used locally to govern the handling of error-like occurrences and globally to effect trace and printout; different degrees of detail were specified by use of several signal tables, and it was thus quite simple to change modes by pushing or popping a table. The agenda was used for organizing the flow *of* control in a manner similar to that described for the Cryptarilhmalic program. The built-in triggering mechanisms provided the means for a very natuial modeling of the kind of medical reasoning discussed in Rubin's thesis.

3. The problems

As we had hoped, these projects pointed out many ways in which KRL-o was deficient or awkward. People were able to complete the programs, but at times they were forced into *ad hoc* solutions to problems which the language should have dealt with. The problems can be grouped as:

- ► Basic representation problems -- ways in which it was difficult to express intuitions about the semantic and logical structure of the domain
- ► Difficulties in manipulating descriptions explicitly
- ► Shortcomings in the matcher
- ► The awkwardness of the t.isr-KRI. interface
- ► Facilities which should have been available as standardised packages
- ► Infelicitous syntax
- ► Cramped address space

Due to the embedding of KRI.-o in iN'M-.RI.isi', none of these problems were fatal, E'ven with the difficulties, we found it possible to write complex programs rapidly, and to experiment with interesting representation and processing strategies. This list also does not include the social and organizational problems which are bound to infect any effort of this nature. Everyone on the project exhibited heroism and stoicism, persisting in their programming without a manual and in a rapidly evolving language which kept slipping out from under the programs almost as fast as they could be modified.

Basic representation problems

KRI.-o embodied a number of commitments as to how the world should be represented. Some of these seemed intuitively justifiable, but did not work out in practice. Others weie too vague to implement in a way which seemed satisfactory.

The categorization of units: Lach unit had a category lype (as described in Bohrow and Winograd (I[(]>77, *pp* 10-J2)) of *Individual, Manifestation, Basic, Specialization,* or *Abstract Category.* This was based on a number of intuitions and experiments about human reasoning, and on the belief that it would facilitate mechanisms such as the quick rejection of a match if there was a basic category disagreement. In practice, these distinctions turned out to be too limiting. In many *of* the hierarchies for speciali/ed domains (such as medicine) there was no obvious way to assign B*asic. Specialization,* and *Abstract.* In dealing with units describing events, the notion of *Manifestation* \va> not precise enough lo be useful. It was generally felt that although the concepts involved were useful, they had been embedded at too low a level in the language.

Viewpoints: One of the major issues in developing KKL was the desire to have facilities for "chunking" knowledge into relevant units. This proved to work out well in most cases, but there was an additional dimension of organization which was lacking. For many purposes, it is useful to combine in a single unit information which will be used in several contexts, and to associate with each piece of the description some identifier of the context (or *viewpoint)* in which it will be used. In the natural language programs, it seemed natural to classify descriptions associated with words and phrases according to whether they related to the structure of syntactic phrases, or to meaning. In the physiology sketch, there were clear places where different viewpoints (e.g. looking at the form of an organ or looking at its function) called for using different information. There were two primitive mechanisms for doing this factoring in KRI.-O -- attaching features to descriptors, and embedding information in contingencies. Both were used, but proved clumsy and felt *ad hoc.*

The relation between prototype and concept: KRL is built on the assumption that most of the information a system has about classes of objects is stored in the form of "prototypes" rather than in quantified formulas. In general, this proved to be a useful organizational principle. However, there were cases of complex interactions between instance and prototype. In the medical domain, for example, a disease such as AcuteRenallailure could be thought of as an instance of the prototype for Disease but could also be thought of as a prototype for specific cases of this disease. There are a number of issues which arise in trying to represent these connections, and although Kkl.-O did not make obviously wrong choices, it also did not make obviously right ones. In general, we seem to have been hoping that too many consequences would just naturally fall out of the notation, when in fact they take more explicit mechanisms.

Further specification hierarchies: In simple network or frame systems (see, for example Goldstein and Roberts, 1977) there is a natural notion *of* hierarchy, in which each descendant inherits all of the slots (or cases) from its parent. Thus, if a (Give is a fuither specified Act then it has a slot for actor as well as its own slots for object and recipient. In a system based on multiple description, the inheritance of slots is not as suaighlfoiward. Tins is especially tine when there is an attempt to do Merlin-like reasoning and use perspectives to "view an x as a y". The basic inheritance mechanism in KKl.-o does not include automatic inheritance of slots. This is vital for cases in which there are multiple descriptions using the same prototype units. However, it makes it awkward (though possible) to prog.ram the cases where the slots are to be inherited simply. Therefore, we included a mechanism for "further specification" which allowed a unit to inherit slots (along with their attached procedures) from a single parent. This was not fully implemented into the system, and was a dangling end in the implementation.

The factoring of context-dependent descriptions: One major design decision in KRI was the use of an object-factored data base, rather than a context-factored one. The unit for a particular object contained all of the different contingencies representing the facts about it in different worlds. This proved quite successful; however, when combined with the kind of descriptions provided by mappings, another issue arises. Using the example of the cryptarithmetic units given earlier, consider the problem of representing what is known about a column in the addition problem if worlds are used to represent hypothetical assignments. Imagine that we know that in the unmarked global world, Column I is an instance of Column, with values for topLettcr, bottoniLcttcr, etc. If in a hypothetical World 1 (in which some value is assumed for a letter) we infer that its sum is 17, we want to add a contingent descriptor. This could be done in two ways:

```
[ColumnI  UNI I  Individual
   <sclf |   (a Column with
                topleter=     A
                     ..)
             (during World I then (a Column w i t h sum = 17))} > j

[ Column I  UNIT Individual
   <sclf {   (a Column with
                topL.cllor s A
                sum = (during World I then 17)
                     ...)}>!
```

These are equivalent at the semantic level, and the first was chosen in the initial implementation -- all factoring into contexts was done at the top level of slots. However this proved to be tremendously clumsy in practice, since it meant

that much of the information was duplicated, especially in cases of recursive embedding. This was exacerbated by the fact that features (See Bobrow and Winograd, p. 14) demanded factoring as well, and were used for a variety of purposes, such as the viewpoints mentioned above. There was a reimplementation midway in the life of kRl -f> in which the basic data structures were changed to make it possible to merge as much of the shared informal ion as possible. There are a number of difficult tradeoffs between storage redundancy, running efficiency, and readability when debugging, and we never found a fully satisfactory solution within KKI.-O.

Data structure manipulation

KKL-o was not a fully declaralively recursive language in the sense that machine language and pure I isi» are. It was not possible to write KRL-o descriptions of the KRL-O structures (e.g. units, slots, descriptions) themselves, and use the descriptive mechanisms to operate on them. Instead, there were a number of LISP primitives which accessed the data structures directly. People ran into a number of problems which could be solved by explicit surgery (i.e. using the LISP functions for accessing KKL data structures, and RPLACA and RPLACD) but which gave the programs a laint of *ad hoccry* and overeomplexity. As an exercise in using KRL representational structures, Brian Smith tried to describe the KRL data structures themselves in KRL-O. A brief sketch was completed, and in doing it we were made much more aware of the ways in which the language was inconsistent and irregular. This initial sketch was the basis for much of the development in KRL-1.

Deletion of information: One of the consequences of seeing KKI-structures as descriptions, rather than uninterpreted relational structures was a bias against removing or replacing structures. Descriptions are by nature partial, and can be expanded, but the most natural style is to think of them as always applicable. Thus, for example, if" a slot was to contain a list (say, the list of digits known to have been assigned in a cryptarithmetic problem), the descriptor used in an instance was the Items descriptor, which is interpreted as enumerating some (but not necessarily all) items in a set. If the description of some object changed over time, then it was most naturally expressed explicitly as being a time-dependent value, using the Contingency descriptor. There are some deep representational issues at stake, and the intuition of thinking of descriptions as additive was (and still is) important. However, it led to an implementation which made it impossible to delete descriptions (or remove items from lists) without dropping to the level of LISP manipulations on the descriptor forms. This caused problems both in cases where values changed over time, and in cases where the programmer wanted the program to delete unnecessary or redundant descriptors in order to gain efficiency. Although deletion and replacement were doable (and often done), they went outside of the KRL semantics in a rather unstructured way.

Explicit manipulation of descriptions: For some of the programs, it was useful to have parts of the code which dealt with the descriptions themselves as objects. For example, in the cryptarithmetic program, the set of descriptions being added to the value slot of an individual Digit could be thought of as a set of "constraints", and used in reasoning. One might ask "What unused digits match all of the descriptors accumulated for the value of the letter A". This is quite different from asking "Which unused digits match the description 'the value of letter A'". Similarly, in the implementation of Winston's program, the descriptions themselves needed to be thought of and manipulated as relational networks. The ability to use descriptions in this style gave power in writing the programs, but it had to be done through LISP access of the descriptor forms, rather than through the standard match and seek mechanisms.

Problems with the matcher

Specifying the match strategies: The matcher in KRL-0 took a KRL-O description as a pattern, and matched it against another description viewed as a datum. For each potential descriptor form in the pattern, there were a set of strategies for finding potentially matching descriptions in the datum. The ordering of these named strategies, and the interposition of special user-defined strategies was controlled by use of the signal mechanism. This was designed to give complete flexibility in how the match was carried out, and succeeded in doing so. Many specialized match proceses were designed for the different projects. However, the level at which they had to be constructed was too detailed, and made it difficult to write strategies which handled wide ranges of cases. The strategies were mostly reflections of the possible structures in the datum, and did not deal directly with the meaning of the descriptors. This led to having to consider all *of'* the possible combinations of forms, and to programs which did not function as expected when the descriptions contained different (even though semantically equivalent) forms from those anticipated.

Returning bindings: Since patterns for the matcher were simply KRL-0 descriptors, and there was no coherent meta-description language to define procedural side effects, it was very difficult to extract the bindings from a match. This was handled in the legal example, which most needed it, by providing special signal tables for these matches, again leading to a feeling of *ad hocery* to get around a basic problem in matching.

Control of circularities: In using matching as a control structure for reasoning, it is often useful to expand the match by looking at the descriptions contained in the units being compared. Consider the units:

```
[Give UNIT Basic
   <self (A Receive with
              received* (ihe given)
              receiver = (the receiver)
              River « (the giver))>
   <giver (A Person)>
   <receiver (A I*erson)>
   <given (A PhysicalObject)>]

[Receive UNIT Basic
   <sclf (A Give with
              given* (the received)
              receiver = (the receiver)
              giver * (the giver))>
   <giver (A Person)>
   <rcccivcr (A Person)>
   <received (A Physic»IOI>jcct)>]

[Event 17 UNIT Individual
   <seir (A Give with
              giver = Jane
              receiver = Joan
               given = (A Hummer))>
```

If asked whether the pattern (A Receive with received = (A Hammer)) matches Lvent17, the rnatchcr needs to look in the unit for Give in order to see that every (Give is indeed a Receive, and to match up the slots appropriately. However, this can lead to problems since descriptions in units could

easily be self-referential, and mutually cross-referential. In a slightly more complex case, the matcher could try to match a (;ive by looking at its definition as a Receive, and then transform that to a Give, and so on. Some of the early match strategies we developed fell into this trap and looped. The simple solution that was adopted to limit such circular expansion was to adopt a depth first expansion policy, and to limit the total depth of expansion (recursion through definition). This obviously works both in this case, and to limit arbitrarily large non-circular searches. In the limited data bases we used, it never caused a match to be missed when the programmer expected it to be found. But it is a crude device which docs not provide adequate control over search.

Inefficiencies due to generality: Since the matcher was designed to allow a wide range of stiallies, a fairly large amount of processing was invoked Tor each call. Often, the programmer wanted to check for the direct presence of a certain descriptor, and to avoid the overhead, dived into LISP. Thus, instead of writing:

```
(Match .'I vent 17
        '(A dive with giver - Junr)
        Simple strusture Match Table
```

it was possible to write:

```
(F.Q Jane
     (GctIfem ((.elfillcr 'giver
                    (Get Perspective 'Give
                         (GetSlot 'self Event 17)))))
```

Given that the SimpleStructureMatchTable caused the matcher to look only at direct structural matches, the two forms were equivalent, and the second avoided much of the overhead. Many problems arose, however, in cases where later decisions caused the description form to be different (for example, embedded in a contingency) but to reflect equivalent information.

Problems in the interface between KKL and I.ISP

One of the major design decisions in KRL-0 was the use of ! ISP for writing procedures, rather than having a KRL programming language. This was viewed as a temporary measure, allowing us to quickly build the first version, and work out more of the declarative aspects before trying to formulate a complete procedural language in the following versions. A number of awkward constructs resulted from the need to interface LISP procedures and variables to the KRL environment.

Limited procedural attachment modes: Only the simplest forms of procedural attachment were implemented. Thus, for example, there was no direct way to state that a procedure should be invoked when some combination of slots was filled into an instance. Procedures had to be associated with a single condition on a single slot. It was possible to build more complex forms out of this by having a trigger establish further triggers and traps (there are examples of this in the unit for Column given above), but this led to some rather baroque programming.

Communication of context: When a trap or trigger was invoked, the code associated with it needed to make use of contextual information about what units were involved in the invocation and what state the interpreter was in (for example in the use of hypothetical worlds). This was done simply by adopting a set of I ISP free variables which were accessible by any piece of code, and were set to appropriate values by the interpreter when procedures were invoked. This approach was adequate in power, but v/eak in slrucluie, and a number of the detailed problems which arose in the projects grew out of insufficient documentation and stability of what the variables were, and what they were expected to contain when.

Unstructuredness of procedure directories: The notion of having a "signal table" containing procedural variables was a first step towards breaking out of the normal hierarchical definition scheme of LISP. The intention in developing a KRI. procedural language is to develop a set of structured control notions which make it unnecessary for the programmer to fill in the detailed responses to each possible invocation. In the absence of this, KRL-o signal tables had much the flavor of machine code. A clever programmer could do some striking things with them (as in their use in SAM for controlling language analysis and generation), but in general they were hard to manage and understand.

Underdeveloped Facilities

The KRL overall design (see Bobrow and Winograd, p. 3) involved a series of "layers" beginning with the primilive underlying system and working out towards more knowledge-specific domains. Part of the ability to implement and lest the language so quickly came from deferring a number of problems to higher layers, and letting users build their own specialized versions of pieces of these layers as they needed them. In most cases this worked well, but there were some areas in which a certain amount of effort was wasted, and people felt hampered by not having more general facilities.

Sets and sequences: KRL-o provided only three primitive descriptors (Items, AlIItems, and Sequence) for representing sets and sequences. Notions such as subset, position in sequence, member of set, etc. all had to be built by the user out of the primitives, Everyone needed some of them, and it became clear that a well thought out layer of standard units and procedures would have greatly simplified the use of the language.

Indexing schemes: The index mechanism built into KRL-o was based on simple collections of key words. It was assumed from the beginning that this was to be viewed not as a theory of memory access, but as a minimal primilive for building realistic access schemes. One of the projects (I-low) attacked this directly, but the rest stuck to simple uses of indexing, and did not explore its potential in the way they might have if a more developed set of facilities had been provided initially.

Scheduler regimes: As with indexing, the scheduler mechanism of KRL-o was intended primarily as a primitive with which to build interesting control structures which explored uses of parallelism, asynchronous multi-processing, etc. The only structuring was provided by the use of a multi-layer queue Like the category types discussed above, it was an attempt to embed some much more specific representation decisions into a system which in most places tried for generality. It was not restrictive, since the system made it possible to ignore it totally, allowing for arbitrary manipulation of agenda items. However, because it (and no other scheme) was built in, it tended to be used for problems where other regimes would have been interesting to explore.

Notation

The KRL-o notation was strongly LiSP-based, using parenthesization as the primary means of marking structure.

This made it easy to parse and manipulate, but led to forms which were at times cumbersome. This was especially true because of the use of different bracketing characters ("()", "{}", "<>") for descriptors, descriptions and slots. At times a unit would end with a sequence such as "})}){>)". There was one simplification made during the course of the implementation, allowing the description brackets "{}" to be omitted around a description containing a single descriptor. The examples in this paper use this convention. In addition, better notations were needed for expressing sets and sequences, and were explored in the KINSHIP project.

Limited address space

One of the shortcomings which most strongly limited the projects was in the implementation, not the basic design. INTP.RI ISP is a paged system, based on a virtual memory which uses the full 18 bits of the PDP-10 address space. The philosophy has always been that, with some care to separate working sets, system facilities could grow to large sizes without placing extra overhead on the running of the program when they were not being used. This has led to the wealth of user aids and facilities which differentiate INIIRUSP from other t ISP systems.

As a result, more than hall of the address space is used by the IN II RI isp system itself. The KPL o system added another quarter to this, so only a quarter of the space was available for user programs (including program storage, data structure storage, and working space). Both of the extended systems (SAM and Medical) quickly reached this limit. This resulted in cutting back the goals (in terms of the number of stories and questions handled by NAM, and the amount of the sample diagnosis protocol handled by Medical), and also led the programmes to put a good deal of effort into squeezing out maximal use of their dwindling space. Some designs were sketched for providing a separate virtual memory space for KRL data structures, but their implementation was put *off* for later versions, since the lessons learned in using KP.l -o within I he space limitation were quite sufficient to give us direction for KRI.-i.

4. Current Directions

The projects described above were completed by the end of summer 1976. Since that time, we have been primarily engaged in the design of Kkl-I, and as of this writing (June 1[\]>77) are in the midst of implementing it. The development has involved a substantial shift of emphasis towards semantic regularity in the language, and a formal understanding of the kinds of reasoning processes which were described at an intuitive level in the earlier paper. Much of this has been the result of collaboration with Brian Smith at M.I.T, who is developing a semantic theory (called KRS for Knowledge Representation Semantics) which grew out of attempts to systematize and understand the principles underlying systems like KRL

The new aspects of KRL-I include:

- A uniform notion of *meta-description* which uses the descriptive forms of KRL-I to represent a number of things which were in different *ad hoc* forms in KRL-O. The old notions which are covered include features, traps and triggers, index terms, and a variety of other detailed mechanisms. The emphasis has been on providing a clear and systematic notion of how one description can describe another, and how its meaning can be used by the interpreter. A number of the problems related to the manipulation of description forms are solved by this approach.
- A more structured notion of the *access* and *inference* steps done by the interpreter. The interpreter is written in a style which involves operating on the *meaning* of the forms, rather than the details of the forms themselves. This makes possible a more uniform framework for describing matching and searching procedures, and the results they produce. It allows the language to be described in terms of a clear semantics (see Hayes, 1977 for a discussion of why this is important). We expect it to make the development of complex Match and Seek processes much easier.
- A notion of *data compaction* which makes it possible to use simple data record structures to stand for complex descriptor structures, according to a set of declarations about how they are to be interpreted. This enables the system to encode all of the internal structures (e.g. the structure which represents a unit) in a form which can be manipulated as though it were a full-fledged description.
- A compiler which converts simple Match, Seek, and Describe expressions into corresponding INTLRIISP record structure manipulations, reducing the overhead on those instances of these processes in which only simple operations are to be done, Tins should make it possible to preserve efficiency while writing much more uniform code, with no need to use explicit I.ISP manipulations of the structures Use *of* the notions of compiling and compaction allows the conceptually correct but notaiionally expensive use of uniform metadescription to be supported without excessive running cost in the common cases.
- A uniform notion of *system events* which allows more general kinds of procedural attachment, and includes traps, triggers, and signals. Also, by including much of the INTFRLISP interface in description form, it has become more uniform and understandable as well.
- A simplified syntax, in which indentation is used to express bracketing, eliminating the need for most parentheses. It also uses "footnotes" for attaching meta-descriptions, and has simple set and sequence notations.
- Simplified notions of categories, inheritance chains, and agendas, which avoid some of the specific commitments made in KRI.-O.
- Expanded facilities for sets, sequences, scheduling, time-dependent values, category hierarchies, matching information and multiple-worlds. These are all built up out of the simpler, uniform facilities provided in the kernel, but they represent a substantial body of standardized facilities available to the user.

We are currently exploring a number of different solutions to the address space problem. Until LISP systems with a larger address space are available, some sort of swapping mechanism will be necessary, but we see this as a temporary rather than long-term problem.

The cycle of testing on KRL-I will be similar to the one described in this paper, but with an emphasis on a smaller number of larger systems, instead of the multiple mini-projects described above. We feel that with KRL-t) we explored a number of important representation issues, but were unable to deal with the emergent problems of large

systems. Issues such as associative indexing, viewpoints, concurrent processing, and large-scale factoring of knowledge can only be explored in systems large enough to frustrate simplistic solutions. Several programs will be whiten in KRI.-i, on the order of magnitude of a doctoral dissertation project. Current possibilities include: a system for comprehension of narratives; a system which reasons about the dynamic slate of a complex nuilti-process program, and interacts with the user about that slate; and a travel arrangement system related to (it's (bobrow et. al., 1977). Current plans include much more extensive description and documentation of the system than was the case with KRI.-O.

We do not view KRI.-i as the final step, or even the next-to-last step in our project. In Bobrow and Winograd, 1977 (pp. 34-3(>) we discussed the importance of being able to describe procedures in KKI. structures. ouri plan at that time was to design a comprehensive programming formalism as part of KRI.-I. In light of the shift of emphasis towards better understanding the aspects which we had already implemented, we have postponed this effort for later versions, still considering it one of the major foundations needed for a full KKL system. There remains the large and only vaguely understood task of dealing in a coherent descriptive way with programs and processes. It is likely that to develop this aspect will take at least two more cycles of experience, and as we learned so well with KRI-O, there is always much much more to be done.

References

Bobrow, D.G., Kaplan. R.M., Kay, M., Norman, D.A., Thompson, H., and Winograd, T., GUS, a frame driven dialog system. *Artificial Intelligence,* 1977 V 8. No. 2.

Bobrow, D.G. and Norman D.A., Some principles of memory schemata, in D.G. Bobrow and A.M. Collins (fids.), *Representation and Understanding,* New York: Academic Press, 1975, 131-150.

Bobrow, D.G. and Winograd, 'I'., An overview of KKL-0, a knowledge representation language. *Cognitive Science,* V. I, No. 1, 1977

Hayes, f\. In defense of logic, (Draft paper) 1977

Kaplan, R. A general syntactic processor. In R. Rustin (Ed.), *Natural language processing.* New York: Algorithmic^ Press, 1973.

Lehnert. W„ Question Answering in a story understanding system, Yale University Computer Science Research Report #57, 1975.

Meldman, J.A.. A preliminary study in computer-aided legal analysis, MIT project MAC TK 157, 1975.

Minsky, M., A framework for representing knowledge, In Winston, P. (Ed.), *The psychology of computer vision,* McGraw-Hill, 1975.

Newell, A., and Simon, H.A., *Human Problem Solving,* Prentice Hall, 1972.

Norman, D.A., & Bobrow, D.G. On data-limited and resource-limited processes. *Cognitive Psychology,* 1975, 7, 44-64.

Rubin. A.D., Hypothesis formation and evaluation in medical diagnosis (MIT-AI Technical Report 316). Cambridge: Massachusetts Institute of Technology, 1975. .

Schank, R.C. (Ed.). *Conceptual information processing,* Amsterdam: North Holland, 1975.

Schank. R.. and the Yale AI Project, SAM -- A story underslander, Yale University Computer Science Research Report /M3, August, 1975.

Teitelman, W., INTERLISP reference manual. Xerox Palo Alto Research Center, December, 1975.

Thompson, H., "A Frame Semantics approach to Kinship", ms. Univ. of California, Berkeley. 1976

Winograd, T., Frame representations and the declarative procedural controversy. In Bobrow, D.G. and Collins, A. (Eds.). *Representation and Understanding,* New York: Academic Press. 1975.

Winston, P., Learning structural descriptions from examples. In P. Winston (Ed.). *The psychology of computer vision.* New York: McGraw-Hill, 1975.