

MCC Nonproprietary Technical Report AI-055-85

CYC:
Using Common Sense Knowledge
To Overcome Brittleness and
Knowledge Acquisition Bottlenecks

Douglas B. Lenat, Mayank Prakash, Mary Shepherd

July 15, 1985

c Microelectronics and Computer Technology Corporation
9430 Research Blvd., Austin, TX 78759
Nonproprietary Version of Technical Report AI-037-85

CYC: Using Common Sense Knowledge To Overcome Brittleness and Knowledge Acquisition Bottlenecks

**Doug Lenat
Mayank Prakash
Mary Shepherd**

July 15, 1985

This is a brief sketch of MCC's *CYC* project, an effort to build, over the coming decade, a large knowledge base of real world facts and heuristics. In section 1, we discuss the need for the project -- i.e., its long term milestones. As the title of this report suggests, the idea is that the two major limitations to building large intelligent programs might be overcome by using such a knowledge base. Section 2 provides an example of how such a knowledge base could be used to solve problems of the type that stymie current expert systems, namely those requiring common sense and analogical reasoning. We then report how the project is being conducted currently (section 3). This includes the project's philosophy and methodology, and an extended example of how we are currently putting that into practice. Together, sections 2 and 3 discuss qualitatively, and by example, how we intend to meet the goals posed in section 1. Section 4 presents a more quantitative discussion of feasibility and milestones.

1. Introduction

The major limitations in building large software have always been (i) its brittleness when confronted by problems that were not foreseen by its builders, and (ii) the amount of manpower required to build a coordinated system. These problems are no less true for Artificial Intelligence. Indeed, the recent history of expert systems *highlights* brittleness and knowledge acquisition as the two bottlenecks.

1.1. Making AI Programs More Flexible

How is it that a person can flexibly cope with unexpected situations? The answer seems to be that as our specific "expert" knowledge fails to apply, we draw on increasingly more general knowledge. This general knowledge is less powerful, so we only fall back on it reluctantly. The general knowledge itself breaks down into a few types: there is real world factual knowledge, of the sort that one might find in an encyclopedia; there is common sense, the sort of knowledge that an encyclopedia article *wouldn't* contain -- it would assume the reader knew without being told (e.g., an object can't be in two places at once.) There is a third, important, immense, yet nonobvious source of general knowledge we rely on frequently: *analogy to any and all of the specific knowledge we have*, no matter how far-flung its "field" from our present problem. The totality of our knowledge can -- through analogy -- be brought to bear on any specific situation we face; to make this a "real time" process we employ heuristics for prioritizing which analogies to consider first, and we may also use our brain's parallelism to good effect here.

The CYC project is an attempt to tap into the same sources of power: to provide a comprehensive skeleton of general knowledge, plus a growing body of specific knowledge. Using the former directly, and the latter through analogy, we hope to provide assistance for expert systems, natural language understanders, and so on, as they get "stuck" on problems.

If parallelism is needed, we are prepared to develop ways to utilize it, but we do not view it as a panacea, hence it is not a key element of our initial project plan.

1.2. Making it Easier to Add New Knowledge

Interestingly, the large real world knowledge base project necessary to open the brittleness bottleneck also provides an answer to the *other* bottleneck problem, knowledge acquisition (KA). Let's see why this is so. How do people learn and understand new information? In a 1984 essay, Marvin Minsky points out that humans rarely learn "what", we usually learn "which". In other words, we assimilate new information by *finding similar things we already know about*, and recording the exceptions to that "analogy". This leads to amusing mistakes by children ("will that Volkswagon grow up to be a big car?") and adults (e.g., cargo cults), but these are just extreme cases of the mechanism that we all are using all the time to assimilate new information. In other words, we deal with novelty the best we can in terms of what we already know (or, more accurately, what we *believe*) about the world.

Another way of saying this is that the more we know, the more we can learn. This may sound trivial, but its converse is equally valid and more to the point: without starting from a large initial foundation, it's difficult to learn. We have to date seen no AI system which tries to do knowledge acquisition "from strength", from a given large body of knowledge. This is why we expect the CYC system to facilitate knowledge acquisition; it would be the right system to build even if KA were our only goal. If the knowledge base is large (and representative) enough, then adding a new piece of knowledge ought to involve no more than pointing to -- and thereby connecting -- a few existing pieces

There is a very weak form of this process in use today, in building expert systems, namely the simple expedient of "copy&edit": As you build up your expert system's knowledge base, and you're about to enter the next chunk (rule, frame, script,...), it is often faster to find a similar chunk and just copy it and then edit that copy, than it is to type that new chunk into the system from scratch.

One limitation to current uses of copy&edit is that, if the growing system only knows a couple hundred things (rules, frames, etc.), then those form the total universe of potential objects from which to do the "copy". We expect it will be increasingly likely to find an existing close match (from which to copy) given CYC's eventual size. To this extent, CYC is aiming at is in the same spirit, the same source of power. But there are two more powerful ways in which CYC may help the copy&edit process along. The first is to suggest -- or help the user choose -- which existing units to copy from. The second is that CYC would use analogy to do much of the "editing" itself, automatically, after the knowledge base builder got the process started. Let's consider an example of how these two new aids to knowledge acquisition would work:

Suppose one has begun to explicate the "medical-treatment as warfare-against-disease" analogy. CYC has been told about this analogy (or discovered it), has seen "enemy soldiers" map to "bacteria", "invasion" to "infection", "containment" to "containment", "drugs" to "weapons", and so on. CYC could then decide that certain new concepts ought to get created: the *susceptibleToDrugs* slot, and its inverse, *effectiveAgainstBacteria*, would map into brand new concepts, new kinds of slots, called *susceptibleToWeapons* and *effectiveAgainstEnemySoldiers*. Moreover, as new units are created, CYC should use its growing model of the analogy to decide which substitutions to make in the unit body instead of forcing the user (knowledge base builder) to make them all manually. For instance,¹

```

makesSenseFor(susceptibleToWeapons) =
analogue(makesSenseFor(analogue(susceptibleToWeapons))) =
analogue(makesSenseFor(susceptibleToDrugs)) =
analogue(Bacteria) =
EnemySoldiers

```

In a similar fashion, the *makesSenseFor* slot of *effectiveAgainstEnemySoldiers* would be filled automatically with *Weapons* (i.e., each kind of weapon has its own characteristic effectiveness against each type of enemy troop).

Suppose the knowledge base builder now wants to add the analogue of a rule about the use of infiltrators to subvert and undermine the enemy's effort from within. Instead of having to make all the editing changes by hand, the CYC system could use its knowledge of the overall analogy -- of what mapped to what -- to do most of the needed substitutions itself. Moreover, CYC itself could suggest creating the analogue to that rule in the first place.

Just as "copy&edit" in its present form is highly cost-effective, saving the knowledge base builder a large amount of time, we expect CYC's analogy-driven "copy & automatically-edit & manually-touch-up" to be an even larger step in that same direction.

¹ Throughout this paper, a call to get the value of the *f* slot of frame *x* is written in functional notation: *f(x)*; this carried through to the current CYC program as well.

1.3. Functionality of the CYC System

The two long-term motivations behind CYC are therefore to make our large programs less brittle, and to enable them to be built more easily. These have led us to work towards a large knowledge base of real world factual and common sense knowledge, which includes heuristic knowledge.

Besides the two primary goals, we expect such a knowledge base to be capable of other functions as well. Drawing on the breadth of knowledge and analogical reasoning capabilities of the system should enable a new generation of learning and theory formation programs, story generation programs, and ICAI programs. On the following page is a list of the various uses we expect CYC to ultimately be put to, the functionality we are aiming for:

A. Exploring (by humans, of the knowledge base)

This is the promise of a qualitatively new kind of reference work, the improvements coming from the dynamic nature of the interaction; the multiply connected nature of the knowledge (rather than linear text), the three dimensionality and aural "soundspace" that aid in navigating and apprehending, and the custom-tailored nature of the experience: filtering through individual user models; organizing material based on the individual user and his/her current purpose, rather than based on traditional article boundaries and sequencing.

B. Question answering (by the system, for humans)

By interfacing with natural language systems, we hope to provide question answering capabilities which draw on restricted forms of the same features discussed above, under point A.

C. Teaching (of humans, by the system)

This is similar to A and B, but here the machine has more of the initiative. It makes up the questions, sets goals for each part of the sessions, models not just the knowledge and preferences of the student but also their learning abilities, strengths, and weaknesses, and uses this model to "diagnose and treat" them.

D. Teaching (of the system, by humans)

CYC itself should make it easier for people to tell it new things. This is discussed elsewhere in this paper, notably sections 1.2 and 3.2, where we explain and illustrate the use of analogy in knowledge acquisition.

E. Learning (by the system, from examples)

The flip side of "Teaching" is having the system learn by generalizing from specific experiences. Since some of those experiences will be directly with human users, this kind of learning supports function D, as the CYC program automatically builds up and refines models of each user. Other types of learning by examples will support functions F and H.

F. Solving Problems (for other programs, by the system)

Not only question-answering for people should be possible, but also helping other programs, such as expert systems, solve problems. This is illustrated in detail in section 3.1, below. CYC should also help natural language understanding programs do disambiguation, in effect by simulating the possible meanings and seeing which ones do and don't make sense in (CYC's version of) the real world. As with most of these functionalities, this can call on most of the others as subroutines to help it succeed, and can be called by them to help them succeed.

G. Finding Analogies (by the system)

Analogy-finding can be used both as a problem solving method (in service to other uses listed in this chart) or for its own sake; i.e., finding analogies that might be of interest to the user, or simply to discover analogies worth "banking" against possible future use.

H. Theory formation (by the system)

This will be a return to the line of machine learning by discovery begun in AM and Eurisko. We believe that in that subfield, no less than expert systems, the bottleneck problem is the paucity of knowledge that learning programs begin with; CYC will provide a testbed for a program to begin learning from an already large knowledge base.

2. How the CYC System will be used

The two subsections below present, respectively, examples of CYC using general and specific knowledge to aid expert systems.

2.1. Using a Broad Corpus of General Knowledge

In this example, CYC functions as a sort of back-end processor, a behind-the-scenes resource that can be drawn upon in times of need. A rule is entered, in the expert system, with its "unless" conditions absent, and CYC then fills them in automatically. This could be done either to improve knowledge acquisition (do it when the rule is first entered) or to improve performance (don't do it until the expert system gets "stuck" and needs common sense knowledge to guess what sort of exceptional case it's in the midst of.)

Before we see how CYC might help, let's see what the problem really is. In a standard expert system, the rules refer to technical terms which are related to each other in various ways but are not "tied in" to a global model of the real world. What we propose is precisely that tying in. Consider the following dialogue, between a human (called "Expert") trying to build an expert system, and a conventional AI expert-system-building "shell" tool (called "TOOL"). The \$ signs indicate variables.

TOOL: Please type in the next rule. IF:

Expert: AskedInInterview(\$doctor,\$patient,"Do you have \$x ?")= Yes

TOOL: THEN:

Expert: Has(\$patient,\$x)

TOOL: Thank you. Here is my rephrasing of the rule in English:

If doctor AskedInInterview patient "Do you have x ?"
and result was Yes,

Then conclude patient has x.

Is this correct?

Expert: Yes.

TOOL: Please type in the next rule. IF:

.

.

.

Now nothing is actually "wrong" with the rule, and the expert may continue to enter hundreds of additional rules which help the system diagnose various kinds of medical problems. The system appears to work fine on many cases, even though it does not really understand much about symptoms, disease, doctors, patients, interviewing people, and so on. All the system knows about AskedInInterview is that it is an operator, a Lisp function it can call, which returns one of the values

Yes, No, and Unknown. To the expert system, it is not much different from the various lab test operators, for instance, such as GramStainTest and AerobicityCheck.

As various cases are run, many mistaken diagnoses may be tracked to this rule, and each one may result in a new clause being added to its If part; e.g., what if \$x were the kind of thing someone might lie about, such as "a history of insanity in your family"? Indeed, many expert system languages (such as MCC's Proteus) provide for explicit "Unless" conditions to be added to each rule.

When actually coding the new Unless clause referred to above ("unless the patient might lie about \$x"), the kinds of things someone might lie about would have to be listed explicitly (insanity in the family; use of illegal drugs; country of birth; etc.) Here is an opportunity for the knowledge base builder to accidentally overlook some items. As new embarrassments (e.g., herpes) and societal mores (e.g., abortion, homosexuality) come and go, the rule would have to be updated.

We've now sketched the problem. It comes down to an *knowledge illusion*: the rule appeared to make sense to us, but only because we human beings already knew what the various terms it mentioned all meant: doctors, patients, intake interviews, and symptoms. Consider the rule from the point of view of the expert system. It might just as well have been written

If the result of F083(A,B,C) was D,
Then conclude F002(B,C).

(We're willing to admit that the program might understand "the result was", but will not grant that it understands the meaning of "Has". There are dozens of things that the average five year old child knows about "having a symptom" and "having a disease" that no current medical expert system knows.) When recast in the harsh light of GENSYMs, the limits of expert system "common sense" become all too clear.

How can CYC help? Let's replay the original session in which the rule was added, the way it might look if CYC were present in the dialogue:

TOOL: Please type in the next rule. IF:

Expert: AskedInInterview(\$doctor,\$patient,"Do you have \$x ?") = Yes

TOOL: THEN:

Expert: Has(\$patient,\$x)

TOOL: Thank you. Here is my rephrasing of the rule in English:

If doctor AskedInInterview patient "Do you have x ?"
and result was Yes,

Then conclude patient has x.

Is this correct?

Expert: Yes.

CYC: Please help explain what the *AskedInInterview* operator is all about. Is it a

- (a) LabTest
- (b) Querying
- (c) VisualObservation
- (d) LogicalDeduction
- (e) Other

Expert: b.

CYC: Is *\$doctor* intended to be any member of the set *Doctors*?

Expert: Yes.

CYC: Please help explain what set the variable *\$patient* is intended to fall into. Is it

- (a) A set called *Patients* which you'll tell me about now
- (b) A set I already know about (if so, type its name)
- (c) May be known to me (we'll browse through KB now)
- (d) A new set you'll tell me more about now (if so, type its name)
- (e) Help!

Expert: a.

CYC: Please edit the new *Patients* unit:

At this point, the expert enters a screen editor, to record these facts: *Patients* is a subset of *People* and *RecipientsOfService*, and the service being received is *MedicalCare*, and by definition something is wrong with the *Health* of the typical patient. All these are concepts *CYC* knows – the names of units it already has in its KB (*People*, *RecipientsOfService*, *TypicalMember*, *MedicalCare*, *Health*), and they in turn participate in relations with many other units representing sets (*Doctors*, *Symptoms*), attributes (*MentalHealth*, *PhysicalHealth*), and scripts (*ProvidingService*). The expert makes most of these connections by pointing to the related units, as s/he navigates through "knowledge space". This navigation can be implemented simply by a Zog-like network of menus, or less simply by using three-dimensional graphics, joysticks, helmets, and so on. At present, we are using an Inspector-like frame editor; the figure below is a snapshot of the screen during the entry of the *Patients* unit:

Please enter name of new slot:
serviceReceived

Please enter new value:
MedicalCare

Inspector for unit - Patients.

elementOf	(CollectionsOfPeople)	TypeInVal
elements	<u>Nil</u>	PPVal
subsets	(People RecipientsOfService)	EditVal
newSlots	(symptoms lifeExpectancy diagnosis lengthOfIllness)	PutVal
partitionings	(BySymptoms)	EVAL & PutVal
serviceReceived	(MedicalCare)	CopyValTo
myElementOf	(Units)	CopyValFrom
myCreator	(Shepherd)	ReplaceInVal
myCreationdate	(" 2-Jul-85 11:32:31")	InspectVal
		Undo

Once the Patients unit has been entered, the dialogue continues:

CYC: Please help explain what set the variable \$x is intended to fall into. Is it

- (a) The set of OwnedObjects [which I already know about]
- (b) Another set I already know about [if so, type its name]
- (c) May be known to me [we'll browse through KB now]
- (d) A new set you'll tell me more about now [if so, type its name]
- (e) Help!

Expert: Symptoms.

TOOL: Please type in the next rule. IF:

.

.

.

Notice in the last interchange that CYC's guess about the meaning of "Has" -- as used in the question "Do you have \$x?" -- was wrong but plausible (i.e., that "Has" meant "owns as a possession"). It did already know about the set Symptoms, so when the expert typed that in it understood what "Has" meant; it was finished "butting in", and returned control to the Tool.

The difference in the two dialogues -- without CYC and with it -- is that CYC doesn't let the expert get away with ambiguity and jargon, with new terms it doesn't understand. It continues to ask about all the various unknown terms and variable names, forcing the expert to describe and explain them at higher and higher levels, until s/he finally is using only terms which CYC already understands (such as Goal, Querying, BodyPart, LogicalDeduction, Remove, Weapon, Symptoms, HasAsAPossession, HasAsAnAttribute, Doctors, etc.)

This seems to make a fair amount of extra work for the knowledge base builder. We have claimed that it more than compensates for this. To see how this is so, we need to peek at some of what CYC knows:

Querying

basicKindOf: 2WayCommunicating
actors: (message c elementOf Questions) (message d)
constraints: (d is result of (RecallMatching b c))

2WayCommunicating

basicKindOf: Communicating
action:
 (Communicating a b c)
 (Communicating b a d)

Communicating

basicKindOf: Informing
actors: (communicator a) (recipient b)(message c)
action:

(Openingcommunications a b)
 (Telling a b c)
 (RecallMatching b c)
 constraints:
 (CanUnderstand b a)
 (Desires a (Communicating a b c))

Recall-Matching

basicKindOf: Remembering

Remembering

basicKindOf: Informing
 actors: (rememberer a)(memory b)
 constraints: Accurate
 accuracy: (# - - Moderate variance High)

AllWaysItFails

defn: (Lambda (u &rest) (MapUnion (Genl* u) 'WaysItFails))
 elementOf: (InheritedSlots UnlessFillingSlots)
 builtFrom: (Genl* WayItFails)
 combiner: (MapUnion)
 makesSenseFor: (Processes)
 filledWith: (Predicates)
 dataType: (Set)

WaysItFails

defn: (Lambda (u &rest) (Negate (Constraints u)))
 elementOf: (DerivedSlots UnlessFillingSlots)
 builtFrom: (Constraints)
 combiner: (Negate)
 usedInBuilding: (AllWaysItFails)
 makesSenseFor: (Processes)
 filledWith: (Predicates)
 dataType: (Set)

Constraints

elementOf: (ScriptSlots PrimitiveSlots)
 usedInBuilding: (WaysItFails)
 makesSenseFor: (Processes)
 filledWith: (Predicates)
 dataType: (Set)

Desires

With the above knowledge, CYC can automatically compute many of the Unless conditions for our rule, conditions we would otherwise have to get by introspection or hard experience. We call for (AllWaysItFails 'AskedInInterview). Note that all we do is ask for the value of a particular slot of a particular unit, namely the AllWaysItFails slot of the AskedInInterview unit. This is typical of the way in which computation is initiated in CYC.

The program looks at the AskedInInterview unit, and sees that there's no slot there labelled AllWaysItFails; in fact, there's no slot at all on that unit, except a basicKindOf slot that points to Querying -- this is the fact that the user told CYC in the above dialogue, when s/he selected item "(b) Querying", in response to the question about what AskedInInterview was an instance of. CYC doesn't give up at this point, it consults the Defn slot of the unit in the system called AllWaysItFails. There it finds that it can compute the desired value by unioning together the (values of the) WaysItFails slots of all the generalizations of AskedInInterview. A quick scan of these generalizations (which include following basicKindOf and action links) shows them to include AskedInInterview itself, Querying, 2WayCommunicating, Communicating, Recall-Matching, Remembering, Informing, and on up to even more general units (MentalProcessing, Processing, Anything). (Actually, the process is slightly more complex, as it involves expanding out the action slot of each of these units, and carrying along a series of substitutions, to get to the next ones.)

The system finds that none of these units has a WaysItFails slot, but instead of giving up it consults the Defn slot of the unit in the system called WaysItFails. There it finds it can compute such values by negating the Constraints listed for each of those units. Some of them, namely Communicating and Remembering, do have Constraints slots listed. Others don't, and CYC consults the Constraints unit to see if it has a Defn from which constraints can be computed -- and there is none, so it quits at this point. The value returned is therefore the union of the negations of the constraints it did find. Here are a few snapshots of this process in action:

(AllWaysItFails '(AskedInInterview d p q))

d = doctor

p = patient

q = "do you have x"

(Union

(WaysItFails (AskedInInterview d p q))

(WaysItFails (Querying d p q))

(WaysItFails (2WayCommunicating d p q (RecallMatching p q)))

(WaysItFails (Communicating d p q))

(WaysItFails (OpeningCommunications d p))

(WaysItFails (Telling d p q))

(WaysItFails (RecallMatching p q))

(WaysItFails (Remembering p q))

(WaysItFails (Informing p))

(WaysItFails (MentalProcessing p))

(WaysItFails (Processing p))

(WaysItFails (Communicating p d (RecallMatching p q)))

(WaysItFails (OpeningCommunications p d))

(WaysItFails (Telling p d (RecallMatching p q)))

(WaysItFails (RecallMatching d (RecallMatching p q)))

(WaysItFails (Remembering d (RecallMatching p q)))

(WaysItFails (Informing d))

(WaysItFails (MentalProcessing d))

(WaysItFails (Processing d))

(WaysItFails (Anything)))

(Union

(Negate (Constraints (Communicating d p q)))

(Negate (Constraints (Communicating p d (RecallMatching p q))))

(Negate (Constraints (Remembering p q)))
 (Negate (Constraints (Remembering d (RecallMatching p q))))

((NOT (CanUnderstand p d))
 (NOT (Desires d (Communicating d p q)))
 (NOT (CanUnderstand d p))
 (NOT (Desires p (Communicating p d (RecallMatching p q))))
 (NOT (Accurate (Remembering p q)))
 (NOT (Accurate (Remembering d (RecallMatching p q)))))

The old, "bare" expert system would either have required the rule-enterer to introspect and try to catch all manner of Unless conditions at the time the rule was written, or else would have become stuck now and then when one of the unanticipated Unless conditions occurred in a real case. If called on ahead of time (when the rule was entered), CYC could have proposed several Unless conditions; if not called upon until the expert system was stuck, CYC would then propose many new options to explore, to ask the user about, at that time -- questions derived from much higher-level, more general, "common sense" sorts of knowledge, not technical medical knowledge. In either case, here are some of the Unless conditions CYC would propose:

The patient didn't understand the doctor;
 The doctor didn't really want to make the patient understand the question;
 The doctor didn't understand the patient;
 The patient didn't really want the doctor to know the real answer;
 The patient mis-remembered the answer;
 The doctor mis-remembered the significance of the patient's answer.

Actually, many more Unless conditions might turn up; we haven't carried the above snippet of the knowledge base far enough to show their origins. For instance, OpeningCommunications might have failed because the patient was deaf and never heard the question, and the doctor mistook silence for an "I don't know" answer. Telling might have failed because of poor pronunciation, background noise, etc. (Telling is not the same as Understanding; it's like Speech Recognition, whereas Understanding is like Natural Language Understanding, comprehension once the words themselves have been successfully transmitted.)

This is the kind of thinking which makes doctors "robust" problem solvers compared to expert systems. Such possibilities can be mechanically generated by the program because it has tied previously-isolated terms, such as AskPatient, into a global knowledge base of general real world information. General information, about human communications and human memory, can thereby be brought to bear on the problem either dynamically, when medical expertise alone is not enough, or statically, to suggest Unless clauses for rules.

The point of this example is not to claim that specific Unless clauses about all these various possibilities couldn't be explicitly added to the "old" system, by Expert (the human expert physician who built it.) Rather, the point is to show how a single general knowledge base, like CYC, could *automatically* enable such reasoning, when needed, without having to be specially told these things by physicians. CYC could be useful as a "back end" processor to expert systems in many different fields: the knowledge used to help unstick the medical diagnosis system had nothing to do with medicine.

2.2. Using A Huge Corpus of Specific Knowledge

The preceding example is the most superficial kind of payoff of using CYC to underly the next generation of expert systems. We expect a much more subtle, and powerful, kind of synergy to arise eventually, as the system begins to compass more and more specific knowledge. Instead of relying just on general knowledge, the system might analogize to specific knowledge from similar or even unrelated fields of expertise. For example, if the doctor is stymied, one approach to deciding what to do next might be to view the situation as a kind of warfare, a combat against the disease, and perhaps some suggestion which is the analogue of advice from that domain might be of use ("contain the infection", "give the infection some minor chances, as the risk is worth the information learned about its behavior", and so on.)

It is difficult to present an example of this sort of synergy that doesn't appear contrived. We refer the reader to writings by Skemp, Hadamard, Poincare, and Edison, to name a few, who document cases of the use of detailed analogies to aid in solving difficult problems.² Edison describes (in notes only recently analyzed, and reported in Science 85) how most of his inventions started out as analogues of earlier ones; e.g., the motion picture camera started out looking like a phonograph and gradually evolved from there. Some evidence for the reliance on metaphor may come from the reader's own introspections: try to be aware, in the next few days, of how pervasively -- and efficaciously -- you employ metaphors to solve problems, to cope with situations, and in general to make decisions. Usually these metaphors are not very "deep"; rather, we get power from them because of the immense breadth of knowledge from which we can choose them. Ultimately, the test of this idea will be in the performance of the CYC system, once it has a large enough accumulation of specific knowledge.

² The careful reader will perceive that that is precisely what we are doing in this paragraph, by the way, tackling the difficult problem of how to convince you of this.

3. Building the CYC System

To build CYC, we must encode all the world's knowledge down to some level of detail (to tap into common sense, the source of power outlined in section 2.1), and then continue building a nontrivial fraction of detailed world knowledge in as well (to tap into metaphor, section 2.2's source of power.)

This may sound staggering, but consider that a one-volume desk encyclopedia does exactly that -- well, more or less that. The word "encode" in the above paragraph means that we must represent the knowledge in such a form that the program can find and use it when appropriate. For this reason, simply getting an online version of an encyclopedia would be of little use, as there is practically nothing that current AI technology could draw from the raw text. Rather, we must carefully re-represent the encyclopedia's knowledge -- by hand -- into some more structured form, such as a semantic network. As can be seen from the second goal of the project -- facilitating knowledge acquisition -- this process should get easier and faster as the system grows and can itself begin to help with its own growth. Nonetheless, we are still talking about a knowledge base building effort which is two orders of magnitude beyond anything ever attempted. How is it that we intend to do this? What sorts of resources will it require? This section, and section 4, respectively address those two questions.

3.1. Philosophy

Our working hypothesis is that AI has for many years understood enough about representation and inference to tackle this project, but no one has sat down and done it. It is at once an extremely humble and an extremely brash hypothesis. To take the edge of humility off it, we remark that only by pulling together the latest (and slightly beyond the latest) human interface tools, Lisp machines, ideas of enforced semantics, and funding for a decade-long effort, could a project of this scale be attempted. To take the edge of brashness off the hypothesis, we remark that we may fail in some ways (e.g., our current ideas about how to reason by analogy may not "scale up"), but we expect that in the process we will learn some powerful new lessons about building large intelligent programs. We will return to some of these issues in section 4, when we discuss quantitatively the feasibility of the CYC project, what milestones we foresee along the way, and what alternatives we have at each decision point.

Our plan is simple: we will represent one or two articles of each type, from a one-volume desk encyclopedia, and then bring on board a large number of data enterers to add the final part (the last 99%) by using copy&edit (e.g., to enter the knowledge about Britannia-Metal, the enterer would copy the closest existing article already represented, say Pewter, and then edit that structure to reflect the differences between the two alloys.) We expect the system itself to play an increasingly useful role in speeding up that process, by making intelligent substitutions (as sketched in section 1.2), and automatically adding knowledge by applying common sense (as sketched in section 2.1.)

The problem of keeping all the knowledge base builders consistent with each other will be handled by enforced semantics. Implicitly, this occurs as people copy existing structures rather than try to come up with ways to organize things on their own. Explicitly, we have our language state the semantic constraints. E.g., every rule's "IfMaybeRelevant" slot is supposed to be faster than, and

logically implied by, its "IfTrulyRelevant" slot; this can be checked dynamically, and if someone is misusing those slots then the "proper" use can be explained to them. Another semantic constraint is that each "Example" of a unit must satisfy the unit's "Definition". A third such constraint is that a unit's "Definition" must be implied by the "Definition" of every entry listed on the unit's "Generalizations" slot. Hundreds, eventually thousands, of such explicit semantic constraints work together, much as rules in an expert system, to advise the knowledge enterer when -- and how -- s/he is misusing the "commonly accepted meaning" of the various slots.

A final point to make is that this effort interacts symbiotically with MCC's Data Base efforts (and on down into Parallel Processing) and with MCC's Human Interface efforts. The size of the knowledge base makes it necessary (within 1-2 years) to strongly augment Lisp machines with a system capable of doing "smart" Get and Put operations on immense data bases. As for HI, we are already working together to provide the kind of interface we need. We expect that most of the knowledge entry will be done by zooming around "knowledge space", pointing at objects and substructures to be tied together. Although this is just a graphical extension of current-day text-based copy&edit, we have already found text editing, and conventional graph editing tools, insufficient for easily extending the growing CYC knowledge base.

Section 4 provides quantitative estimates of the work involved in this, and translates those into timetables of milestones.

3.2. Methodology

The critical point: we are using an initial set of 400 mutually-distinct articles not primarily for the specific facts they contain, but rather for their "spanning the space" of knowledge. Our methodology should make this clear:

1. Take an article (a typical member of one of the four hundred classes of articles.)
2. Represent the stated information in our representation language. (We have written a frame-based language similar to RLL; we describe it in more detail later.) Part of this step includes disambiguating what the writer actually meant.
3. For each unit/slot/value "fact", move it up to the most general unit (i.e., frame) to which it's valid. If that unit isn't already in the system, create it now.
4. For each fact F, write down the additional facts about the world that are needed to understand F. Clearly, these are common-sense facts that the writer of the article presumed the reader already knew. Now repeat steps 2 and 3 on this new set of facts. (This step may entail creating new units.)
5. For each adjacent pair of sentences in the article, write down -- and encode into our language -- the additional facts about the world that are needed to understand what transpired "between" the two sentences. (This step may entail creating new units.)

6. As we progress we are constantly exercising our representation language, and must occasionally add new capabilities to it. The language should "settle down" as we apply it to each type of article (and, thereby, to each "type" of common sense knowledge, although we do not as yet have a taxonomy of that knowledge.)

At this stage (two or three years from now) we expect to have a skeletal knowledge base completed, incorporating the most general thousand concepts that will ever appear in our knowledge base, plus a smattering of each type of specific concept. We will also have a representation language adequate for representing both the factual knowledge in the articles examined, and also the common sense knowledge implied by (assumed by the writers of) those articles.

7. Employ 2-4 dozen knowledge enterers to encode the final 99% of the knowledge base. Each enterer will take an article, locate the already-represented similar article(s), and perform a machine-assisted "copy & edit" procedure, to produce a machine understandable version of the new article.

This is the analogue of the source of power tapped by the TI NLMenu system: giving users choices rather than having them type in things "from scratch", and it is one way of implicitly enforcing a common semantics (e.g., what the "extreme-examples" slot means) shared by all knowledge enterers -- namely, the new articles inherit the semantics of the parent articles automatically. We also plan to use explicitly enforced semantics, specifications of relationships that must be true about a slot or set of slots. Some of these might be static, universal statements, e.g., "extreme-examples are a subset of examples". Some of these constraints might only be dynamically checkable, e.g., "the IfPotentiallyrelevant slot should run a lot faster, on the average, than the IfTrulyrelevant slot, of the same rule." In both cases, CYC will be able to recognize when its semantics are being violated, and can advise (complain to) the offending knowledge enterer.

8. As this long phase proceeds (1988-93), continually test out the system by hooking it to -- or building within it -- various particular AI programs: expert systems, natural language understanders, and problem solvers. Other sources of knowledge (such as childrens' stories, Tell Me Why books, newspapers, fictional short stories) will be examined, and hopefully a sample of each type will be entered, by us, during this period.

Systematic entry of that level of knowledge is beyond the 8-10 year scope of this project, and would hopefully occur during the mid- and late- 1990's, if the project is a success. That would also be the time when a large fraction of existing expert systems could "hook in", and during which the CYC knowledge base would be adapted into products: knowledge utilities akin to electric utilities, advanced entertainment and art tools, the next generation of CAI systems, and so on. As more and more expert systems tie into CYC, we will be able to exploit analogy more and more effectively, to add knowledge and especially to begin to solve problems by analogizing to other specific domains.

3.3. An Example of the CYC Methodology

We have looked at scores of articles in several encyclopedias by now, but here is the very first article we looked at and applied our methodology to:

coke, hard, gray, porous fuel substance with a high carbon content. It is the residue left when bituminous coal is heated in the absence of air. Coke is used in extracting metals from ores in the blast furnace.

— Concise Columbia Encyclopedia, 1983, p. 180

3.3.1. Step 1: Choosing a representative article.

Frankly, the coke article was chosen at random. Almost any article could have been used as the initial article, as it was bound to be a representative of some class of articles. Later analyses of articles provided the approximate figure of four hundred for the number of types of articles. The intent is that two articles are of the same type iff we could reasonably expect a knowledge base enterer to add the second article relatively easily using copy&edit from the first one. Thus, Lions and Tigers are of the same type, but no two of these are of the same type: Lions, Robots, AnimateObjects, Cheese, Cheesemaking. Our notion of "type" is a vague measure, to be sure, but we have not needed to revise it much even after several months of further work.

3.3.2. Step 2: Encode its explicitly stated factual information.

This resulted in the following units being sketched in:

Coke

basicKindOf: SolidFuelSubstances
 color: Gray
 ingredients: (# -- Carbon relAbundance High)
 hardness: High
 porosity: High
 usedAsIngredientIn: Smelting
 createdAsResidueIn: Coking
 myCreator: Lenat
 myCreationDate: "2-Feb-85 9:57:10"

The first item to note is the way we represented that fact that coke has a high carbon content. The # notation is akin to LOOPS' active value notation: anywhere that a single value might have existed -- in this case, "Carbon" -- the system recognizes a special construct of the form:

(# gf pf v p1 v1 p2 v2 p3 v3... pn vn).

"gf" is a pointer to a special function that tells how to get the value, dynamically, "pf" tells how to put (store) this value; if either is absent (as in the case of the ingredients slot of the Coke unit), the default function is used. "v" is a cached value to return if time is tight, or only a brief answer is desired (i.e., give me a one-word answer to the query "What's an ingredient of coke?") The ensuing p_i/v_i pairs are properties -- and their respective values -- that give additional descriptive information

about this value. In the present case, it isn't coke that's relatively abundant, it's the amount of carbon as an ingredient of coke. We could have employed a slot called carbonContent, and just given it the value High, but that slot would hardly ever be used. Choosing which slots "ought to exist" is usually a matter of envisioning how useful that slot will be to other units, in the future; if it won't be useful, then consider what generalization of the slot *would* be widely useful. In this case, ingredients is a slight generalization of carbonContent, and will be useful for a large fraction of units in our system. An alternative approach, which we are not at all averse to (nor is our language) is to define the carbonContent kind of slot, even though it won't be used much, but define it to "compile" -- i.e., inherit -- down into an ingredients slot which we also define; in other words, "carbonContent: x" would be equivalent to "ingredients: (# - - Carbon relAbundance x)".

We subsequently found out more about coke's composition, and developed a vocabulary of connectives that let us describe most mixtures: InDecreasingAbundance, InIncreasingAbundance, InNoParticularOrder, Absolute%, InApproximatelyEqualAmounts. Using these connectives, we could concisely express the fact that coke is 98% carbon, and the remaining 2% is mostly metals, with small amounts of organic compounds and other miscellaneous trace elements. Here is the way the Coke unit looked at this stage:

Coke

```

basicKindOf: SolidFuelSubstances
color: Gray
typeOfChemComposition: Mixture
ingredients:
  (Absolute% (Carbon 98)
    (InDecreasingAbundance
      Metals
      (InNoParticularOrder
        OrganicCompounds
        TraceElements))))
hardness: High
porosity: High
usedAsIngredientIn: Smelting
createdAsResidueIn: Coking
myCreator: Lenat
myCreationDate: "2-Feb-85 9:57:10"
lastEditor: Shepherd
lastEditedDate: "3-Feb-85 16:20:35"
```

The second item to note is that we separated off the phrase about heating bituminous coal in the absence of air, into a separate, named unit, namely the one for the process called Coking. We similarly separated off the extraction of metals from ores, in a blast furnace, as the process called Smelting. We then sketched in those two units:

Coking

```

elementOf: ChemicalProcesses IndustrialProcesses
inputs: BituminousCoal
operators: Heating
residues: Coke
operatorConstraints: (none-present air) (greater-than temperature 550)
action: apply operators to inputs under conditions
myCreator: Lenat
myCreationDate: "2-Feb-85 14:32:44"
```

Smelting

elementOf: ChemicalProcesses IndustrialProcesses
 inputs: MetalOre Coke
 operators: Heating
 residues: Metal
 locus: BlastFurnaces
 action: apply operators to inputs under conditions
 applicationOfProcess: ReductionOfMetalOxide
 myCreator: Lenat
 myCreationDate: "2-Feb-85 14:41:02"

By now we had dozens of additional units to add, namely one for each kind of slot (e.g., "applicationOfProcess", "hardness"), and one for each concept named inside a value so far (e.g., "Heating", "OrganicCompounds").

The third item to note is that hardness and porosity are relative attributes; coke is hard compared to coal, say, but not compared to diamonds or steel. In other words, coke is *relatively* hard, compared with (most) other solid fuel substances. This is an example of the disambiguation process we included as part of "step 2": figuring out what the writer of the article actually meant to say, precisely. At this stage, we replaced hardness and porosity by two new kinds of slots, relHardness and relPorosity. Their values couldn't just be "High"; we had to use the # mechanism to say what set of values they were high relative to. Including the information that these "High" values are moderately certain (since they came from the Columbia encyclopedia), the Coke unit now looked like this:

Coke

basicKindOf: SolidFuelSubstances
 color: Gray
 typeOfChemComposition: Mixture
 ingredients:
 (Absolute% (Carbon 98)
 (InDecreasingAbundance
 Metals
 (InNoParticularOrder
 OrganicCompounds
 TraceElements)))
 relHardness:
 (# -- High
 relativeToTheSet SolidFuelSubstances
 variance Low
 certainty
 (# -- ModeratelyHigh
 justification ColumbiaEncyclopedia))
 relPorosity:
 (# -- High
 relativeToTheSet SolidFuelSubstances
 variance Low
 certainty
 (# -- ModeratelyHigh
 justification ColumbiaEncyclopedia))
 usedAsIngredientIn: Smelting
 createdAsResidueIn: Coking
 myCreator: Lenat
 myCreationDate: "2-Feb-85 9:57:10"
 lastEditor: Lenat
 lastEditedDate: "3-Feb-85 18:22:56"

3.3.3. Step 3: Move information to more general units.

First, we list three criteria for deciding that a piece of information might merit being moved from its current position to one on a more general unit. Next, we show some examples of when and how we did this for Coke (and other units).

The most obvious signal is when information appears to be duplicated time after time, unit after unit. Even in the few snippets of units we've listed in section 3.3.2, it's clear that there are duplications (e.g., the action slot of both Coking and Smelting; the relPorosity and relHardness slots of Coke). These are clear signals that something more general is going on here.

A second signal of this kind is the use of deep constructs; e.g., in the ingredients slot of Coke, the deepest entry (namely, that some organic compounds and traces of miscellaneous elements exist in the mixture) is probably true for a large collection of mixtures used in the real world -- perhaps it ought to be a default ingredient for Mixtures; i.e., if you (or CYC) had to guess, those trace impurities are probably present in most mixtures, or else it would be worth noting that exception explicitly. The other deepest fact, namely the one about the source of the knowledge being the encyclopedia, and it being moderately highly certain,

The third signal that some information ought to get inherited from somewhere (i.e., ought to now be moved from its present position to a more general unit) is that the English language version of the fact is much terser than its current encoding. For instance, saying that "Coke is hard and porous" takes up a large fraction of the symbols used in encoding the Coke unit, namely the two huge slots relHardness and relPorosity.

Much of the information in the relHardness slot of Coke actually is information about the relationship relHardness itself, hence can be pushed from the Coke unit to the relHardness unit:

relHardness

```
subSlotOf: physicalProperties
makesSenseFor: Solids
rangeOfValues: High/LowScale
elementOf: PhysicalPropertySlots RelativePropertySlots
myCreator: Lenat
myCreationDate: "2-Feb-85 19:32:02"
lastEditor: Prakash
lastEditedDate: "8-June-85 12:29:00"
inherit:
  (instances
    (members
      relativeToTheSet (basicKindOf $unit)
      variance Low
      certainty
        (# - - ModeratelyHigh
          justification ColumbiaEncyclopedia)))
```

This last value says that any instance of a relHardness slot (i.e., in an actual unit, such as Coke) should inherit some information. Namely each entry therein (each "member" of the slot, such as "High") should get some added properties and values, namely that that entry is a relative term, relative to the set of which the unit (e.g., Coke) is a basic kind (in Coke's case, SolidFuelSubstances). Also inherited to each of those entries is the information about the variance being low, the certainty being

moderately high, and the justification for that certainty being the source book used. The relHardness slot of Coke can now just have the single simple entry "High"!

We could do the same for relPorosity, but in fact its inherit slot would be identical to relHardness' inherit slot. As we discussed above, that's a clear sign that we should push this information to a more general level. What do relPorosity and relHardness have in common that makes this coincidence occur? They are both RelativePropertySlots, and that's really where this default information applies. Every element of RelativePropertySlots -- such as the two rel... slots we've been discussing -- ought to inherit information of the form (instances (members relativeToTheSet (basicKindOf \$unit) variance Low certainty (# - - ModeratelyHigh justification ColumbiaEncyclopedia))), just as the unit relHardness used to have. So now relHardness has no inherit slot at all, nor does relPorosity; instead, the unit RelativePropertySlots exists and has one:

RelativePropertySlots

```

basicKindOf: Slots
elements: relPorosity relHardness
elementOf: KindsOfSlots
myCreator: Lenat
myCreationDate: "2-Feb-85 20:42:31"
inherit:
  (elements
    (instances
      (members
        relativeToTheSet (basicKindOf $unit)
        variance Low
        certainty
          (# - - ModeratelyHigh
            justification ColumbiaEncyclopedia))))

```

Actually, if we assume that all slots' values will have moderately high certainty, with that same justification, then we can add an inherit slot to the very general unit Slots, and simplify the inherit slot of RelativePropertySlots to this value:

```

(elements
  (instances
    (members
      relativeToTheSet (basicKindOf $unit)
      variance Low)))

```

Here is part of what the Slots unit looked like at this stage:

Slots

```

basicKindOf: Functions
basic kinds: HistorySlots PhysicalPropertySlots RelativePropertySlots ...
elementOf: KindsOfSlots
myCreator: Shepherd
myCreationDate: "1-Feb-85 8:03:19"
inherit:
  (elements
    (instances
      (members
        certainty
          (# - - ModeratelyHigh
            justification ColumbiaEncyclopedia))))

```

In other words, the Coke unit now keeps one single symbol ("High") of its originally complicated relPorosity and relHardness slots; the units called relPorosity and relHardness don't actually keep any of that information; RelativePropertySlots keeps about half of it (the information to go to the unit's basicKindOf slot to find out what set the comparison is relative to); and Slots itself gets the remaining half (information about the default certainty and its justification.)

The same kind of generalizing is done for each piece of knowledge that is signalled to be moved. For instance, glancing up at the Smelting unit, the action slot is fully inherited from ChemProcesses (and that is eventually 90% moved farther on, to Processes); the values of its operators and residues slots are fully inherited from ReductionOfMetalOxide, as is one of the values of its elementOf slot. Thus the Smelting unit now is shortened to:

Smelting

elementOf: IndustrialProcesses
 inputs: MetalOre Coke
 locus: BlastFurnaces
 applicationOfProcess: ReductionOfMetalOxide
 myCreator: Lenat
 myCreationDate: "2-Feb-85 14:41:02"

This activity of generalization and abstraction not only reduces the redundancy of the knowledge base, it drives the conceptualization -- the defining and naming -- of new, general concepts (such as RelativePropertySlots and SolidFuelSubstances). It is at this *intermediate* level of generality that we expect convergence and power, as CYC develops, more than down at the level of particular specific facts such as those about Smelting.

As another illustration of this process, consider the units below. We began with Irrigation, which grew quite large in size. We then moved more and more information upward, to increasingly general concepts: WaterConduction, FluidConduction, Conduction, TransportationProcesses, and so on. Indeed, almost all the Irrigation information was moved all the way up to TransportationProcesses; a little was left behind along the way (e.g., the unit WaterConduction retains the knowledge that the transportee is water). Irrigation now only needs to remark on the restricted set of transport vehicles (pipes and ditches), the restricted set of energy sources used (pumps gravity capillary action), and what processes irrigation is used in (namely, growing). It is also worth noticing that once this general knowledge of transportation processes was codified, it was quite fast to add the units for the concepts of the rain cycle (i.e., EarthWaterEvaporation, RainCloudsForming&Moving, and Raining) because each of them is an instance of TransportationProcesses, having its own particular values for destination and departure points, energy source, etc.

Irrigation

elementOf (CollectionsOfTransportProcesses)
 energySource (Pumps Gravity CapillaryAction)
 supersets (CommercialWaterSupplying WaterConduction)
 transportVehicle (Pipes Ditches)
 usedIn (GrowingProcesses)
 myElementOf (Units)
 myCreator (Lenat)
 myCreationdate ("15-Apr-85 16:55:16")
 myLastEditDate ("17-Jun-85 09:56:09")
 myLastEditor (Lenat)

TransportationProcesses

elementOf (CollectionsOfTransportProcesses)
 subsets (Conduction TransportationOfDiscreteObjects)
 achieves ((Deprive (a Actors with location = departurePoint) Of transportee)
 (Supply (a Actors with location = destinationPoint) With transportee))
 action (InSequence (InAnyOrder (Prepare transportee)
 (P Prepare departureReceptacle))
 (InAnyOrder (Insert transportee Into departureReceptacle)
 (P Prepare transportVehicle))
 (InAnyOrder (Transfer transportee Into transportVehicle)
 (P Prepare destinationReceptacle)
 (E Enable energySource))
 [Apply energySource To transportVehicle
 (Result: (Change (Location transportee) To
 (Location destinationReceptacle)
 (Transfer transportee Into destinationReceptacle))
 actorConstraints
 ((elementOf departureTime Times)
 (elementOf arrivalTime Times)
 (EarlierThan departureTime arrivalTime))
 actors (transportee transportVehicle departurePoint energySource
 departureReceptacle destinationReceptacle
 departureTime destinationPoint arrivalTime)
 constraints
 ((Equal (Location destinationReceptacle)
 destinationPoint)
 (Equal (Location departureReceptacle)
 departurePoint)
 (Can (Contain transportVehicle transportee))
 (Can (Contain departureReceptacle transportee))
 (Can (Contain destinationReceptacle transportee))
 (Can (Power energySource transportVehicle
 From departurePoint To destinationPoint))
 (Can (Survive transportee action))
 (Interval (action TransportationProcesses)
 departureTime arrivalTime))
 genlProcesses (Transferring)
 newSlots (departurePoint departureReceptacle destinationPoint
 destinationReceptacle energySource transportVehicle transportee)
 supersets (MechanicalProcesses Transferring)
 myElementOf (Units)
 myCreator (Lenat)
 myCreationdate ("15-Apr-85 15:50:38")
 myLastEditDate ("17-Jun-85 09:41:12")
 myLastEditor (Lenat)

Conduction

elementOf (CollectionsOfTransportProcesses)
 subsets (FluidConduction ElectricalConduction SignalConduction)
 supersets (TransportationProcesses)
 myElementOf (Units)
 myCreator (Lenat)
 myCreationdate ("15-Apr-85 15:58:15")

TransportationOfDiscreteObjects

elementOf (CollectionsOfTransportProcesses)
 subsets (Shipment Travel)
 supersets (TransportationProcesses)
 myElementOf (Units)

myCreator (Lenat)
myCreationdate ("15-Apr-85 15:54:08")

FluidConduction

elementOf (CollectionsOfTransportProcesses)
subsets (GasConduction SolidConduction LiquidConduction)
supersets (Conduction)
myElementOf (Units)
myCreator (Lenat)
myCreationdate ("15-Apr-85 16:00:42")

LiquidConduction

elementOf (CollectionsOfTransportProcesses)
subsets (WaterConduction)
supersets (FluidConduction)
myElementOf (Units)
myCreator (Prakash)
myCreationdate ("22-Apr-85 10:29:36")

WaterConduction

elementOf (CollectionsOfTransportProcesses)
subsets (CityWaterPumping EarthWaterEvaporation Irrigation
RainCloudsForming&Moving RainCycle
Raining RuralWellPumping WaterDrainage)
realizes (WaterSupplying)
supersets (LiquidConduction)
transportVehicle (Pipes Ditches Air Riverbeds WatertightContainers)
transportee (Water)
myElementOf (Units)
myCreator (Lenat)
myCreationdate ("15-Apr-85 16:02:34")
myLastEditDate ("17-Jun-85 09:56:07")
myLastEditor (Lenat)

RainCycle

elementOf (CollectionsOfTransportProcesses)
subsets (EarthWaterEvaporation RainCloudsForming&Moving Raining)
action (Cyclically (action EarthWaterEvaporation)
(action RainCloudsForming&Moving)
(action Raining))
supersets (NaturalWaterSupplying WaterConduction)
myElementOf (Units)
myCreator (Lenat)
myCreationdate ("16-Apr-85 14:12:37")

Raining

elementOf (CollectionsOfTransportProcesses)
departurePoint (Stratosphere)
departureReceptacle (RainClouds)
destinationPoint (EarthsSurface)
destinationReceptacle (EarthsSurface)
energySource (Gravity)
supersets (NaturalWaterSupplying RainCycle WaterConduction)
transportVehicle (None)
myElementOf (Units)
myCreator (Lenat)
myCreationdate ("16-Apr-85 14:06:34")
myLastEditDate ("10-Jun-85 14:21:23")
myLastEditor (Lenat)

EarthWaterEvaporation

elementOf (CollectionsOfTransportProcesses)
 departurePoint (EarthsSurface)
 departureReceptacle (EarthsSurface)
 destinationPoint (RainClouds)
 destinationReceptacle (Stratosphere)
 energySource (Evaporation)
 supersets (RainCycle WaterConduction)
 transportVehicle (None)
 myElementOf (Units)
 myCreator (Lenat)
 myCreationdate ("16-Apr-85 14:22:02")

RainCloudsForming&Moving

elementOf (CollectionsOfTransportProcesses)
 constraints ((Usually (NotEqual destinationPoint departurePoint)))
 departurePoint (Stratosphere)
 destinationPoint (Stratosphere)
 destinationReceptacle (RainClouds)
 energySource (Wind)
 supersets (RainCycle WaterConduction)
 transportVehicle (None)
 myElementOf (Units)
 myCreator (Lenat)
 myCreationdate ("16-Apr-85 14:17:04")

The figure on the following page illustrates another aspect of this generalization process. The article on coke provided information shown in line 4 of the figure. We then began to generalize that knowledge, as shown by the third line of the figure. Further generalizing involved composing of processes into a single multistep processes (line 2 in the figure). Common sense (which we will focus on in the next subsection) about *why* this was done, led us into a discussion of zeroth order economics, which resulted in the frame about commercially viable processes (line 1). As can be seen from the figure, this frame further tied in to two other frames -- Processes in general, and People -- both of which we already had represented in our system at the time. The idea is that by looking at a broad, necessarily *sparse* sampling of *specific* concepts, such as coke, we will be led to generate a *nonsparse* amount of the *general* knowledge of the world.

that we wouldn't normally call it an analogy, but there is a continuum here. The "uranium to U-235 to plutonium" process is more complicated (e.g., one can use mixtures of enriched uranium and plutonium itself to serve the intermediate role of fuel².) As another example, consider the process of spending money to do research, to come up with ideas which are then developed into new products. If the knowledge enterer has the right tools, s/he may use the knowledge the system has about "coal to coke to metal" to facilitate entry of the new scenario; conversely, if it is entered independently, the system itself may notice that analogy and use it to answer the question "what will happen if research dollars get scarcer?" Thousands of additional analogues exist even farther afield; e.g., people are trained as teachers and then go teach.

Notice that we are intentionally blurring the distinction between "two instances of a general situation" and "two situations which are analogous." Usually, once one has found a useful analogy, one names and stores away the generalization that encompasses (aspects of) the two analogues. Conversely, one often finds analogues by searching known "cousins" of a concept (i.e., concepts which are known to share some properties already with the given one), and searching for new additional commonalities. Eventually we expect to have a taxonomy of kinds of analogy and generalization processes, with heuristics specialized to each type, to guide CYC in finding them, extending them, and so on.

3.3.4. Step 4: Extract and encode the implied (common sense) knowledge.

At this stage, after representing the factual information from the Coke article, we inspected each sentence, extracting from it the common sense knowledge of the world that its writer assumed its reader would know. Examining the Coke article, here are some of its unstated facts:

A kind of fuel is a mass noun -- i.e., there isn't just one piece of coke in the world; if you split one in half, each is now a full-fledged piece of coke.

Hardness and porosity are negatively correlated; both are uncorrelated with color; all three are *form* (appearance) properties of physical objects.

Fuels have widely varying color, solidity, and other physical attributes.

Real world objects have both *form* (physical attributes) and *function*.

"Fuels" is a class of substances grouped together because of similarities in function, not form. They store energy from one source so it can be released at a different time, place, and/or rate that enables some subsequent process to take place. Typically, the fuel is consumed at that time.

Miscellaneous simple knowledge of hardness, porosity, grayness, carbon, residues, coal, heating, absence, air, extracting, metals, ores, and blast furnaces. I.e., one unit's worth of information for each of these concepts, such as what their alternatives or siblings are.

Most processes, including heating, generally starts at a certain time, and stop at a later time. The duration is rarely random: it's either fixed, or lasts until some condition is met (typically, this condition is: the goal of the process is either achieved or irrevocably failed).

After a process runs, residues may be left -- usually deposited where the original inputs to the process were located. While a process is running, dynamic outputs are generated, usually at that same locus.

Industrial processes are run either for their dynamic effect, or because their outputs (including residues) are more valuable than their inputs.

A substance is valuable if it's used in a process that is highly valuable (e.g., producing an even more valuable end product).

Often, substances are needed in purer states than they naturally occur in (or initially get produced in).

At a meta-level: an article will specify all the atypical values, pairs of negatively correlated values, and many of the high-variance attributes. For physical objects, both form and function will be specified (or defaulted), along with origin and/or history.

Two things surprised us early on, as we began to collect this sort of knowledge:

First, much of it involves an understanding of economics and, more generally, human motivation and values. For instance, the article doesn't say *why* we'd bother to produce coke out of coal, and metal out of ore and coke. It assumes the reader understands that such processes must be economically viable, hence the end products must be much more valuable than the inputs. Also, there is probably no known method that produces those end products more cost effectively (where, perhaps, other "costs" than ergs are involved: threat to life and limb, pollution, need to centralize rather than distribute the loci, uncertainty of success despite a high expected return on investment, etc.)

The second surprising thing is that most of the unstated knowledge is declarative -- not procedural -- information about other concepts (e.g., fuels, industrial processes). The article would refer to a concept, such as fuel, and assume that the reader understood "the basics" of that concept. After representing SolidFuelSubstances and FuelSubstances in our knowledge base -- putting in just the basic form and function and origin information -- we sat down again to see what additional common sense knowledge about fuel was required by the Coke article, and found that everything we could think of really wasn't particular to Coke, it got pushed up to take its place as declarative (form or function) knowledge about some more general concepts (SolidFuelSubstances, FuelSubstances, Solids, Coking, Smelting, IndustrialProcesses, Processes, etc.)

In other words, the common sense you need to understand X is often just the explicitly stated form and function knowledge about concepts mentioned in the various values of slots of X (notably in X's basicKindOf and other types of generalization slots). For instance, much *common sense* knowledge about Coke turned out to be *factual* knowledge about Fuels; much of the the *common sense* knowledge required to understand Fuels turned out to be the *factual* knowledge about even more general concepts: physical objects, storage and conversion of energy -- plus, as was mentioned in the last paragraph, knowledge of economics and human motivation.

3.3.5. Step 5: Extract and encode the intersentential knowledge.

Another surprise occurred when we entered step 5 of our loop, looking for the unstated knowledge that let the reader cross the gap from one sentence to the next. In most short articles, there was almost no such knowledge required! The articles were simply a stream of facts, loosely clumped into sentences. For instance, in the Coke article, the three sentences could come in any order, with no degradation in the article's content or comprehensibility. Only in the long articles (several paragraphs in length), and in historic accounts of a person or region, did we find such "missing link" common sense knowledge. In the latter case, typically what's going on "between" sentences is that

various actors in the scenario are reacting to what just happened (matching their perceptions to their own goals, and then deciding what to do next.) For instance: "Atomic bombs were dropped on Hiroshima and Nagasaki. Japan surrendered. U.S. forces occupied Japan."

We have found that compilations of goals into hierarchies of scripts (similar to Schank and Kolodner's MOPS) handles both the shallow, fast inferences, and -- when needed -- the reasoning from deeper and deeper knowledge. Coke does not provide a good example of this; instead, consider these sentences (brought to our attention by John McCarthy): "Napoleon died on St. Helena. Wellington was saddened." The intersentential knowledge includes:

People react to events if they're conscious of them. In particular, they must be alive, and witness or hear of the event.

Wellington must have outlived Napoleon.

Wellington must have heard of Napoleon's death (he was in England, not St. Helena, at the time, and people can't be in two places at once, so he didn't witness it first hand.)

It must be obvious that Wellington would hear of Napoleon's death, or else that would have been stated explicitly. What is the knowledge used to make it unsurprising to a human reader? Below are some of the necessary facts:

The "doings" of famous people are widely publicized.

"Doings" include births, deaths, weddings, divorces, victories, defeats,...

Such doings are even more likely to be known by other important contemporaries (in intercommunicating cultures.)

Such doings are even more likely to be known by "interested parties", those who have some close relationship to the famous person.

"Close relationships" of this kind include family members, allies (coworkers, team members, etc.), enemies (rivals, victims), and so on.

Wellington and Napoleon were not only important contemporaries, but also bitter enemies.

England and France had regular communications passing between them at the time of Napoleon's death.

A slightly deeper analysis rests on the fact that there was an initial *witnessing* event (in which one or more people saw Napoleon die, or saw direct evidence of it, such as his corpse); following this, there was a chain of *communicating* events, wherein each person told the next one; finally, that chain stretched to Wellington. It is likely that one or more links in the chain were not strictly verbal, but involved the written reporting of the event in newspapers or letters, and the subsequent reading of that story by the next person in the chain. This explains the *possibility*, but not the *likelihood* of Wellington hearing the news; e.g., if it had been Napoleon's barber who died on St. Helena that day, we wouldn't expect Wellington to have heard of that event. In other words, *why* is it true that "the doings of famous people are widely publicized, especially to other important, involved contemporaries"?

The witnessing event needs no further justification -- perception is more or less "turned on" for human beings all the time. But why would the chain of communicating events take place? Why is it likely that that spreading activation network touched Wellington? To answer that, we had to dig deeper into human motivation.

Clearly the people in the chain were satisfying some goals or subgoals of their own by passing the news along. Mostly, this appears to be curiosity (on the part of the listeners) and self-aggrandizement (on the part of the speakers). Wellington's close relationship with Napoleon would have made him even more eager to hear of the news, and Wellington's high position in England promised yet higher rewards for whomsoever should bring it to him, so someone may even have gone out of their way to have been the first to tell Wellington.

We went one step deeper, analyzing why people are curious, why they try to make themselves seem important, etc. This entailed organizing (and entering into our system) a top-level tree of human goals. The top level ended up to be species' preservation and self preservation and emotional well-being. Below that came self improvement, self aggrandizement, sating desires, and avoiding unhappiness ("dreads"). Beneath that (as subgoals) came curiosity, ecstasy (including physical pleasure, vengeance, etc.), love (including friendship, family ties, and romantic love), and planning (to decrease the future costs of satisfying goals). The mental dreads came out to be unavenged wrongs, fear itself (!), mental inadequacy (inability to meet a goal), and increasing dependency (such as drug addiction). The physical dreads came out to be death (and -- less extreme -- wasting of time), pain, and physical inadequacy. Pain included the absence of a needed resource, hence included thirst, suffocation, starvation, and addiction withdrawal. There were some independent partitionings of motivation-space as well, such as conscious vs. instinctual, physical vs. mental, short vs. long term, and so on. These became the names of slots that each kind of motivation unit possesses.

Incidentally, it became clear during our codification of human motivation that most of our society's institutions formed -- and are sustained -- because they serve to spread out the cost of satisfying more or less common sub-sub-...goals that people have. For instance, diet clinics and newspapers both serve self-improvement types of goals for their customers; their employees are of course partly (at least) motivated by earning money (self-improvement, self preservation, rivalry, emotional satisfaction), and money itself as an institution increases security and saves time.

The point here is that the superficial knowledge we all use everyday is present in the system ("famous people know the doings of other famous people") and is as deep as it needs to go most of the time, but also present is the deeper knowledge from which we could, if we wanted, explain things in much more detail. Whenever one level of depth of knowledge fails, the program can -- albeitly more slowly -- reason at the next deeper level of knowledge. This is very much in the spirit of the whole CYC effort.

3.3.6. Step 6: Incrementally add to the representation language.

The current representation language began as a reimplementaion of Lenat & Greiner's RLL; the reader is referred to their article describing this frame-based tool in the proceedings of the 1980 AAAI conference. One important feature of this is that the language itself is (or at least should be) described within its own representation; that is, as a set of units. That should facilitate translating the system to other "underlying languages", as only the most fundamental primitives (plus display- and other machine-dependent functions) need be translated. Also, the system should be able to apply its own knowledge and skills -- e.g., question-answering and analogizing -- to itself. We believe this is

important if it is to monitor its own runtime behavior, and explicitly enforce semantics on its users (such as the `IfPotentially/IfTrulyRelevant` slot case, discussed earlier.)

From time to time, additional features have had to be added, as different types of articles present new pieces of knowledge which could only be handled awkwardly at the time.

One example of this is the information about relative slots, certainty, justification, etc., in the Coke case. That caused us to examine other current representation languages, and then implement a feature similar to LOOPS' active values, namely the `(# get put val p1 v1 p2 v2 p3 v3...)` construct.

Another example was when we came up to irrigation (and transportation), and had a sequence of steps to describe. That caused us to adopt a hierarchical script-like mechanism, combining features partially implemented in KRL (Winograd & Bobrow) and CYRUS (Kolodner & Schank).

We expect that this kind of augmentation of the representation language will taper off in time, that it will converge to a stable language, and that goal is one reason we're going out of our way to represent one example of each kind of article there is, rather than focusing on a particular area in depth, first, and then only later discovering holes in our representation language.

4. Feasibility and Milestones of the CYC Project

Section 3 illustrated qualitatively, by example, how it is that our effort will be conducted. This section provides a set of milestones and asymptotic goals for the project, and in the process partially quantifies the effort involved.

1. Asymptotic Goal: A documented, debugged representation language, adequate for representing the spectrum of material contained in -- and assumed by -- the encyclopedia. The way to measure performance for this will be to monitor how often the language needs to be augmented, as new articles are encoded. In other words, how often is the representation language the bottleneck in adding new knowledge into the system? If we fail to asymptotically approach this sort of closure, that would signal the need for a fundamental redesign of the language, perhaps abandoning the goal of enforced simplicity and naturalness in representation, in favor of a simpler, fixed language within which we'd put up with somewhat awkward statements of some kinds of knowledge. Such a contingency would be disappointing; it would slow down the overall CYC effort, but not be a critical problem.

2. Milestone: Have a version of the language running on both Xerox 1109's (in Interlisp-D) and Symbolics 3600's (in Common Lisp). This will enable very simple interfacing to other projects. The measure of performance is that a file of units could be created on one type of machine, and loaded into another (and worked with there). Once accomplished, the two versions would then be constantly updated to match the other. If we fail to reach and maintain compatibility, perhaps due to the vast differences in the user environments, editors, etc., between the two types of machines, we

will have to choose whether to restrict the language somewhat, or add resources to building up a second similar language (plus a translator for units from one language to the other), or switching our work over entirely to just one family of machines (and, hence, just one language).

3. Milestone: A suitable three-dimensional graphics interface for the CYC system -- i.e., for units represented in the language mentioned above in 1 and 2. This would be a tool for viewing the network of frames as a semantic net, and efficiently browsing through it and editing it. The measure of performance is that we will be able to enter knowledge faster using this interface than with current text-based screen editors. A common point in 1, 2 and here in 3, is that the bottleneck in knowledge entry will not lie with the language and its interface. That is, the bottleneck should not be orienting oneself in knowledge space, and using this editor to find and change the knowledge base. We want (so to speak) the bottleneck to be in deciding what the knowledge it is that ought to be entered, not in the entry process itself. That's why the language and the interface to it are being worked on first; they are "power tools" for knowledge entry. If we fail, the existing graphics metaphors may have to be expanded and extended (maybe "flying around knowledge space in a helicopter" isn't good enough); additional types of displays and interfaces may be experimented with; the text-based editors may be improved enough to eliminate the need for the graphics interface. Power tools are exactly that: tools; their successful development should be cost-effective, but no one of them is on the CYC project's "critical path".

4. Asymptotic Goal: Have the CYC system find and use analogies. The only analogies that "count" are those which hadn't already been preconceived and (albeit subconsciously) preprogrammed into CYC by us. We foresee three distinct uses of the analogies: (i) analogy useful in entering new knowledge, (ii) analogy interesting in its own right to human beings, (iii) analogy which would be able to solve some problem (cope with some situation) that would "break" an expert system. The measure of progress toward this goal is how often, and how usefully, our knowledge enterers exploit analogies while they add new knowledge; to hook performance programs up to our system and track how often analogy (rather than just general knowledge) gets them out of binds; and so on. If we fail to have CYC exhibit any of these three uses of analogy, that would force us to rethink our ideas of how analogizing gets done. We would look more carefully at real cases of problem solving and knowledge acquisition by analogy, and try to reformulate algorithms for those processes. This is a distinct possibility, by the way, as we will be the first to apply AI methods for analogical reasoning within such a large knowledge base. There are several alternative diagnoses we may consider: Analogy may not be as useful as we currently believe; this would actually reduce the need to have a broad base of specific knowledge; analogy is important and ubiquitous, but requires better interacting tools to exploit; more detailed cross-indexing of knowledge (more kinds of slots for our units) must be added; etc.

5. Milestone: One important milestone is to have represented some pieces of each kind of knowledge we intend to ever represent in the coming decade. This excludes perceptual knowledge (visual scenes, e.g.), but includes almost all encyclopedic knowledge and the common sense underlying it. The measure of performance is that one example of each "kind" of article is represented within CYC. As discussed above, there are about four hundred distinct kinds of articles, spawning about 4000 total concepts in the program at that time, about half of which will be very general (common sense) concepts. If we fail because one of the "power tools" (1,2,3, above) was delayed in being reached, that would be understandable and correctable. Unlike the other milestones, *there is no good alternative for failure here*. We could restrict the domain of knowledge over which CYC will have some grasp (e.g., only knowledge about static, concrete, real-world objects),

but it is hard to envision any such restriction that would not defeat the whole point of -- and expected sources of power of -- the CYC program. If this milestone is not within reach after several years of work, it will be time to rethink the entire project.

6. Milestone: Represent most of the common sense knowledge underlying the encyclopedia. This will of course not end until the entire encyclopedia is finished (milestone 8), but we expect at least half of the common sense to be found concurrently with milestone 5. The measure of performance is the number of such concepts represented, and the rate at which new ones are needed. The average article we add now (in mid-1985) requires dozens of new common sense units to be added; this rate should drop asymptotically over time. If we fail to converge working by ourselves on this entry process, that is an interesting new finding about common sense, but has no crucial bearing on the feasibility of this project -- the knowledge enterers would have to have and use tools for adding common sense knowledge themselves, as they went along adding factual knowledge from articles. If we fail to find the rate of spawning new common sense concepts slowing down, not only would those tools have to be built (so enterers could easily add new common sense concepts), but careful monitoring of the semantics of the common sense knowledge base would then be called for. It might turn out that new ways of organizing it (e.g., hooking it to a set of simulations of the world) would be needed, to keep it from continuing to explode in size.

7. Milestone: Utilize a large cadre of lightly-trained knowledge enterers, working together on a common, consistent version of the system. This follows a decision-point to hire these workers, reached after milestones 1-5 have been met. The measure of performance for this task (cost-effective utilization of the knowledge enterers) is the rate at which they add new concepts, and the degree to which they maintain a consistent semantics for that knowledge. Our first "knowledge entry personnel" will help us test out the various tools we'll be building to speed up their task. We also must discover what kinds of knowledge enterers we need: high school students, people with Bachelors' degrees in the fields they're representing, experts in those fields, etc. Our guess is that we will need a mixture of all those types, and will quickly discover some heuristics about the kind of enterers to apply to each type of article. The problem of keeping all these people consistent will be handled by enforced semantics. *Implicitly*, this occurs as people copy existing structures rather than try to come up with ways to organize things on their own. This is the analogue of the source of power tapped by the TI NLMenu system: giving users choices rather than having them type in things "from scratch", and it is one way of implicitly enforcing a common semantics (e.g., what the "extreme-examples" slot means) shared by all knowledge enterers -- namely, the new articles inherit the semantics of the parent articles automatically. *Explicitly*, we have our language state the semantic constraints (e.g., every entry in a unit's "Examples" slot is supposed to satisfy the predicate stored in that unit's "Definition" slot. If someone is misusing those slots then the "proper" use can be explained to them.) If we fail to utilize this group of people, we will have to experiment with different techniques for selecting them, training them, keeping them oriented, managing them, etc. Continued failure at this will be fatal to the project, for we -- the small group of AI researchers involved -- could not accomplish on our own the two person-centuries of knowledge entry work required.

8. Asymptotic goal: Enter "the world's most general knowledge", down to ever more detailed levels. A preliminary milestone would be to finish encoding the contents -- and underlying common sense -- of a one-volume desk encyclopedia. Other sources of knowledge (such as childrens' stories) will be examined, and some samples will be entered. *Systematic* entry of that level of knowledge might be called for eventually, if the project is a success. The measure of performance here is probably as

simple as the amount already encoded, plus a check on the rate at which each knowledge enterer is adding knowledge. There are approximately 30,000 articles in a typical one-volume desk encyclopedia, such as [1], almost all of which are about a paragraph long. (For comparison, the Encyclopaedia Britannica has about ten times as many articles, and their mean length is about four times as long.) A conservative estimate for the data enterers' rate is one paragraph-sized unit per hour; this would make their total effort about 150 man-years. The decision to hire the cadre of knowledge enterers required to perform this task in a small number of years is a significant one, and will depend on the successful completion of milestones 1-6. An important point is that the initial few paragraphs will each spawn dozens of new units, but gradually we expect the number of new units spawned to approach one per paragraph. This is the key figure on which we base the estimate of the effort needed, so we should justify it. The argument is based on pigeon-holing: *each unit holds about a paragraph's worth of content, so the ultimate number of units one would expect to get is about the same as the number of English paragraphs.* The initial 1% of the encyclopedia articles and the initial 50% of the common sense knowledge -- the parts that we are doing ourselves -- will be coded at a much slower rate than the final parts, not only because the system will eventually help the knowledge enterers, but also because as we progress we are constantly exercising our representation language, and must occasionally add new capabilities to it. We will not take on the horde of knowledge enterers before that language has settled down, before it has been tested on each type of article and, we expect, therefore, on each "type" of common sense knowledge (although we do not as yet have a taxonomy of common sense knowledge.) If we fail to encode the entire encyclopedia, but come close, that will be fine; if we are not going to even get close by the end of our time horizon (roughly one decade), that will call for serious replanning. Perhaps we will simply need more knowledge enterers, if milestone 7 has been reached successfully. Or, if milestone 9 is within sight, perhaps accelerating work on tying our system to a natural language understanding program will be useful, so that it itself can "read" some of the remaining articles on its own. If milestone 4 is within reach, accelerating the reliance upon analogy may rapidly get us at least a "first pass" over the breadth of the encyclopedia.

9. Asymptotic goal: As this project proceeds, test out the system by hooking it to (or building within it) various particular AI programs: expert systems, natural language understanders, and problem solvers. Besides helping us test CYC, these interfacing should provide synergy, amplifying the performance program and CYC as well. The measure of performance for this would be the gain in performance in the hooked-up programs and, to a lesser extent, CYC itself. That is, we recursively fall back on their measures of performance for understanding natural language, solving difficult problems, adding knowledge to CYC, etc. If we fail to boost their performance, we may have to wait until the CYC program approaches its final size, i.e., we may not have passed the "knee of the curve" yet. Another possibility is that these performance programs would have to be written *within* CYC, rather than merely being loosely hooked up to it. To close on a more positive note, we consider finally the case that we succeed in reaching these milestones, and approaching these asymptotic goals. At that time, many more expert systems (and other performance programs) would "hook in"; meanwhile CYC would be adapted into products: knowledge utilities akin to electric utilities, advanced entertainment and art tools, autonomous design-discovery engines, and the next generation of CAI systems.

Acknowledgements

We would like to thank MCC, and our colleagues there and elsewhere, for their support and useful comments on this work. Special thanks are due to Woody Bledsoe, John Seely Brown, Ed Feigenbaum, Alan Kay, and John McCarthy.

Bibliography

- [1] *The Concise Columbia Encyclopedia*, Columbia University Press, New York, 1983.
- [2] Carbonell, Jaime, and S. Minton, *Metaphor and common-sense reasoning*, Technical report CMU-CS-83-110, Carnegie-Mellon University, March, 1983.
- [3] *Encyclopaedia Britannica*, Chicago, 1984.
- [4] Greiner, Russell, and D. Lenat, "RLL: A representation language language", *Proc. 1st AAAI Conference*, Stanford, pp. 165-9, 1980.
- [5] Greiner, Russell, *Learning by understanding analogies*, PhD thesis, Computer Science Department, Stanford University, July, 1985.
- [6] Lakoff, George, and M. Johnson, *Metaphors We Live By*, U. Chicago Press, 1980.
- [7] Lenat, Douglas, A. Borning, D. McDonald, C. Taylor, and S. Weyer, "Knoesphere: Building expert systems with encyclopedic knowledge", *Proc. 8th IJCAI*, pp. 167-169, 1983.
- [8] Lenat, Douglas, and John Seely Brown, "Why AM and Eurisko appear to work", *Artificial Intelligence* 23, pp. 269-294, 1984.
- [9] Mitchell, Tom (editor), *Proceedings of the 3rd International Machine Learning Workshop*, Rutgers University, June, 1985.
- [10] *The New American Desk Encyclopedia*, Signet, New York, 1984.
- [11] Schank, Roger, and Robert Abelson, *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, Hillsdale, N.J., 1977.
- [12] Winston, Patrick, T. Binford, B. Katz, and M. Lowry, "Learning physical descriptions from functional definitions, examples, and precedents," *Proc. AAAI-83 Conference*, Washington, D.C., pp. 433-439, August, 1983.