AI Magazine Volume 11 Number 3 (1990) (© AAAI)

Cyc: A Midterm Report[1]

R. V. Guha and Douglas B. Lenat

Computers need common sense—not only to prevent brittleness characteristic of most expert systems but also for semantic disambiguation as illustrated here. The Cyc knowledge base aims to resolve such ambiguities by capturing "consensus reality."

A few of the "proverbidioms" illustrated here are: "self-made man," "you are what you eat," "rat race," "time flies," and "people who live in glass houses shouldn't throw stones." *(Artist, © T. E. Breitenbach.)*

After explicating the need for a large commonsense knowledge base spanning human consensus knowledge, we report on many of the lessons learned over the first five years of attempting its construction. We have come a long way in terms of methodology, representation language, techniques for efficient inferencing, the ontology of the knowledge base, and the environment and infrastructure in which the knowledge base is being built. We describe the evolution of Cyc and its current state and close with a look at our plans and expectations for the coming five years, including an argument for how and why the project might conclude at the end of this time.

The majority of work in knowledge representation has dealt with the technicalities of relating predicate calculus to other formalisms and with the details of various schemes for default reasoning. There has almost been an aversion to addressing the problems that arise in actually representing large bodies of knowledge with content. However, deep, important issues must be addressed if we are to ever have a large intelligent knowledge-based program: What ontological categories would make up an adequate set for carving up the universe? How are they related? What are the important facts and heuristics most humans today know about solid objects? And so on. In short, we must bite the bullet.

We don't believe there is any shortcut to being intelligent, any yet-to-be-discovered Maxwell's equations of thought, any AI Risc architecture that will yield vast amounts of problem-solving power. Although issues such as architecture are important, no powerful formalism can obviate the need for a lot of knowledge.

By knowledge, we don't just mean dry, almanac-like or highly domain-specific facts. Rather, most of what we need to know to get by in the real world is prescientific (knowledge that is too commonsensical to be included in reference books; for example, animals live for a single solid interval of time, nothing can be in two places at once, animals don't like pain), dynamic (scripts and rules of thumb for solving problems) and metaknowledge (how to fill in gaps in the knowledge base, how to keep it organized, how to monitor and switch among problem-solving methods, and so on).

Perhaps the hardest truth to face, one that AI has been trying to wriggle out of for 34 years, is that there is probably no elegant, effortless way to obtain this immense knowledge base. Rather, the bulk of the effort must (at least initially) be manual entry of assertion after assertion.

Half a decade ago, we introduced (Lenat, Prakash, and Shepherd 1986) our research plans for Cyc, a decade-long, two person-century effort we had recently begun at MCC to manually construct such a knowledge base. We have come a long way in this time, and this article presents some of the lessons learned and a description of where we are and briefly discusses our plans for the coming five years. We chose to focus on technical issues in representation, inference, and ontology rather than infrastructure issues such as user interfaces, the training of knowledge enterers, or existing collaborations and applications of Cyc.

. . . an aversion to addressing the problems that arise in actually representing large bodies of knowledge with content.

The Evolution of the Cyc Methodology

For two decades, AI research has been polarized into neats and scruffies (roughly corresponding to theoretical versus experimental approaches). After an initial strongly scruffy approach, we seem to have settled on a middle ground that combines the insights and power of each.

On the one hand, we realized that a number of mistakes made in the project's initial years would have been avoided by a more formal approach (especially in regard to the construction of the representation language). We also realized that philosophy had a lot to contribute, especially when it came to deciding on issues of ontology (Quine 1969).

On the other hand, however, there are a number of areas where we found the empirical approach more fruitful. The areas are typically still open research issues for the formalists or have not even been addressed by them, for example, codifying the most fundamental types of goals that people have.

Therefore, our approach is to largely carry out empirical research and be driven by looking at lots of examples but to keep this work supported on a strong theoretical foundation. Further, we have been driven to adopt a kind of tool-kit orientation: Assemble a collection of partial solutions to the various difficult problems Cyc has to handle, and add new tools as required. That is, for a number of problems (time, causality, inference, user interface, and so on), there aren't any known general-purpose, simple, efficient solutions, but we can make do with a set of modules that enable us to easily handle the most common cases.

The bulk of the effort is currently devoted to identifying, formalizing, and entering microtheories of various topics (such as shopping, containers, emotions). We follow a process that begins with a statement, in English, of the microtheory. On the way to our goal, an axiomatization of the microtheory, we identify and make precise those Cyc concepts necessary to state the knowledge in axiomatic form. To test that the topic has been adequately covered, stories that deal with the topic are represented in Cyc; we then pose questions that any reader ought to be able to answer after having read the story.

One of the unfortunate myths about Cyc is that its aim is to be a sort of electronic encyclopedia. We hope that this article lays this misconception to rest. If anything, Cyc is the complement of an encyclopedia. The aim is that one day Cyc ought to contain enough commonsense knowledge to support natural language understanding capabilities that enable it to read through and assimilate any encyclopedia article, that is, to be able to answer the sorts of questions that you or I could after having just read any article, questions that neither you nor I nor Cyc could be expected to answer beforehand.

Our hope and expectation is that around the mid-1990s, we can transition more and more from manual entry of assertions to (semi-) automated entry by reading online texts; the role of humans in the project would transition from the brain surgeons to tutors, answering Cyc's questions about the difficult sentences and passages. This radical change is what it means for Cyc to have a decade-long projected lifespan.

The Evolution of the Representation Language

CycL is the language in which the Cyc knowledge base is encoded. In 1984, our representation was little more than frames. Although a significant fraction of knowledge can be conveniently handled using just frames, this approach soon proved awkward or downright inadequate for expressing various assertions we wanted to make: disjunctions, inequalities, existentially quantified statements, metalevel propositions about sentences, and so on. At least occasionally, therefore, we required a framework of greater expressive power. We were thus led to embed the frame system in a predicate calculus–like constraint language.

Moreover, there were a number of downright ad hoc aspects of the 1984 frame language, such as how inheritance worked. We kept modifying and tweaking such mechanisms, and often, this method forced us to go back and redo parts of the knowledge base so that they corresponded to the new way the inference engine worked. As the size of the knowledge base increased, this process became intolerable. We came to realize that having a clean semantics for the knowledge base was vital, declaratively expressing the meaning of inheritance, TheSetOf, default rules, automatic classification, and so on, so that we wouldn't have to change the knowledge base when we altered the implementation of one of the mechanisms.

As late as 1987, the only inferencing in Cyc was done using these few mechanisms: inheritance along instances (IS-A) links, rigid toCompute definitions of one slot in terms of others plus the running of demons (opaque lumps of Lisp code) and expert system–like production rules. The results were inefficiencies (because of the overuse of the most general mechanisms), abstraction breaking (often resorting to raw Lisp code escapes), and inadequacies (for example, given a rule "If *A* Then *B*," and ¬*B*, Cyc couldn't conclude ¬*A*.)

For efficiency's sake, we developed dozens of specialized inference procedures, with special truth maintenance system–related (TMS-related) bookkeeping facilities for each (Doyle 1987). Then, to recoup usability, we developed a mechanical translator so that one can now input general predicate calculus–like assertions, and Cyc can convert them from this epistemological level into the form required by these efficient heuristic-level, special-purpose mechanisms (see The Current State of the Representation Language).

Originally, Cyc handled defaults in an ad hoc and frequently inadequate way. In the last two years, we have moved to a powerful and principled way of handling them. As we discuss in the section Epistemological Level and Default Reasoning, Cyc constructs and compares arguments for and against a proposition, using explicit rules to decide when an argument is invalid or when one argument is to be preferred over another.

Early on, we allowed each assertion in the knowledge base to have a numeric certainty factor (cf), but this approach led to its own set of increasingly severe difficulties. For example, one knowledge enterer might assert A and assert B and assign them cfs of 95 and 94 (on a 0–100 scale). There wasn't statistical data to support these numbers; the knowledge enterer just meant to express that both A and B were likely, and A was slightly more likely than B. The problem is that some other knowledge enterer might assert C and assert

D, and, respectively, give them cfs of 94.4 and 94.6. Certainly, neither person intended to tell Cyc that A is more certain than C or that D is more certain than B, but this is exactly what they were doing. These problems led us to go back to a simple nonnumeric scheme in which A, B, C, and D would all simply be true (nonmonotonically, that is, true by default), and in addition, there would be two explicit assertions: (moreLikelyThan A B) and (moreLikelyThan C D).

We conclude this section by listing our desiderata for the representation language in which to build our large knowledge base. The next section goes on to describe the current state of CycL, the language that strives toward these criteria. Our point here is that this list bears little resemblance to what we expected the language to be like five years ago:

First, the language should have a clear (and hopefully simple) semantics. The semantics should be declarative for two reasons: to facilitate communication with, and use by, many different problem solvers, be they human or machine and, as mentioned previously, to prevent having to discard or redo the knowledge base when the inference mechanisms change.

Second, it should provide certain inferential capabilities (including verifying conjectures, finding bindings for variables that make some statement true, planning) and provide them efficiently. Why? Only by actually building applications can the knowledge base be truly exercised and, ultimately, validated.

Third, it should provide some scheme for expressing and reasoning with default knowledge. Almost all the knowledge in Cyc is defeasible. Only about 5 percent is monotonic, and of this amount, only about 1 percent is definitional (such as the function cousins being defined as cousins(x) = children(siblings(parents(x))).

Fourth, it should have the expressiveness of all first-order predicate calculus (FOPC) with equality (Moore 1986). It should also be able to handle (nested) propositional attitudes (McCarthy 1986a) (such as beliefs, goals, purpose, dreads). It should have facilities for operations such as reification (McCarthy 1986b) and reflection (Weyhrauch 1986). Each assertion P (fact, rule, and so on) can have various metalevel assertions about it that need to be made; for example, who asserted this and when, what arguments support P and what arguments counter it, what arguments does P participate in, what assertions are analogous to P, and in what ways and to which models (levels of granularity or other clumpings of mutually consistent assertions) does P belong?

Two of the wish list entries are at odds . . . having a clean and simple semantics and providing speedy inference.

Given this wish list, we could summarize the major changes in CycL over the past five years as follows: We started with a frame language whose emphasis was more on issues such as the data structures used, indexing, and interface. We have since realized the importance of a declarative semantics for the language, the need for expressive power, and the importance of making a clear distinction between what knowledge the knowledge base contains and how the knowledge base is implemented.

The Current State of Cyc's Representation Language

The last several paragraphs considered some of the issues that heavily influenced the design of this language. This section presents the result of our striving to satisfy this wish list.[2]

Epistemological Level and Default Reasoning

Two of the wish list entries are at odds with each other: having a clean and simple semantics and providing speedy inference. To improve inferencing abilities, we want to include special-purpose representations and inference routines, procedural attachments, and so on. However, these make it harder to provide a simple semantics. Also, although it is reasonable to expect the semantics of CycL to remain unchanged, it is likely that new constructs are going to be incrementally added to improve CycL's inferencing. The addition of such special-purpose constructs (templates for classification, inheritance, special-purpose inference mechanisms, and so on) is likely to prove bothersome to other programs that use Cyc, for example, programs that were written before the new constructs even existed and, hence, couldn't take advantage of them.

Therefore, we would like users of Cyc (both humans and application programs) to interact with the system at an epistemological level and not a heuristic level. These terms and the distinction between them are used here in the sense of McCarthy and Hayes (1987).

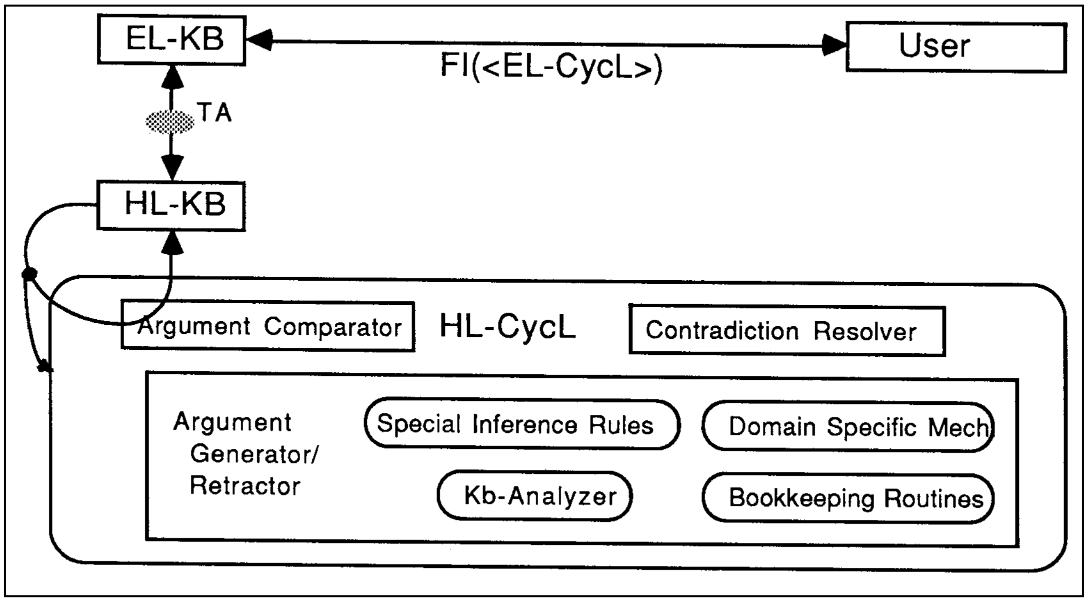

Figure 1. Cyc's Knowledge Base Is Redundantly Expressed at Both an Epistemological Level and a Heuristic Level.

A user (human or application program) will usually communicate with Cyc at the epistemological level, and their utterances are translated by the tell-ask (TA) interface into heuristic-level propositions. The heuristic level consists of several modules for generating and comparing arguments for and against a given proposition.

These observations lead to the conclusion that the knowledge base should be constructed at two levels, which is exactly what we have done. The Cyc knowledge base itself, therefore, exists redundantly at the epistemological level and at the heuristic level, and an external program (or human user) can interact with CycL at either of these levels. The epistemological level uses a language that is essentially FOPC, with a slightly different syntax and augmentations for reification and set construction.[3] It gives an account of the knowledge base in a form that has a simple semantics and, therefore, is easy to communicate in. In contrast, the heuristic level uses a variety of special-purpose representations and procedures for speedy inference. The heuristic level is the language of choice whenever any inference needs to be done.

There is a facility for automatically translating sentences from the epistemological level into the most appropriate representations in the heuristic level, and vice versa; it is called the tell-ask (TA) interface (Derthick 1990) (figure 1). Therefore, one can type epistemological-level expressions (that is, in some form like FOPC) to TA, and they are converted into the most efficient heuristic-level representation.

Both the epistemological level and the heuristic level are real parts of CycL. The epistemological level, in particular, is not just some abstract conceptual level. For example, one could eliminate the entire heuristic level, replace it with a general-purpose problem solver, and Cyc would merely slow down (that is, the Cyc knowledge base would remain unchanged, the same conclusions would be reachable, the same problems solvable, and so on, but at [presumably] a much slower speed). One way in which some of our collaborators choose to use Cyc is at arms' length, accepting only epistemological-level expressions from the Cyc knowledge base and using their own inference engines on them.

The epistemological level is based on a language called the Cyc constraint language, which, as we remarked earlier, is essentially FOPC with equality, with augmentations for defaults and reification. It also allows some amount of reflection of the problem solver into the language and allows for predicates such as justifies that relates a sentence Q to other sentences P_1, P_2, . . . that a problem solver used to deduce it. The Cyc constraint language also uses a number of modals, including beliefs, desires, and goals, whose semantics are defined by using the ability to reify propositions.

The only nonmonotonic constructs used are the closed-world assumption (CWA) and the unique names assumption. CWA, in particular, is used to provide the language with nonmonotonicity, and the default reasoning abilities were designed using CWA and the notion of arguments. The next subsection gives a brief description of this, that is, how default reasoning is done in Cyc; details can be found in Guha (1990b).

The Cyc knowledge base itself . . . exists redundantly at the epistemological level and at the heuristic level . . .

Defaults

The syntactic structure of defaults is identical to that of defaults in circumscription (McCarthy 1987a). Thus, we represent the statement "birds usually fly" as follows:[4]

$isa(x\ Bird) \wedge \neg ab1(x) \supset flies(x)$.

Intuitively, we want to weaken the monotonic statement "all birds fly" to get the default "birds usually fly," and we use the *ab1* predicate to do so. To derive conclusions from this axiom, we use the concept of arguments and have an argumentation axiom instead of the circumscription axiom. The intuition behind this approach is as follows:

We first axiomatize the concept of an argument as an extension of the concept of a proof. In addition to the sentences that might appear in a proof, an argument might also contain a class of sentences that have been labeled as assumptions (for example, sentences of the form $\neg ab1(Tweety)$ are assumptions). We use the ability to reify sentences to do this labeling.

If there is an argument for a sentence, we would like to believe in the truth of the sentence. However, because arguments are a weaker notion than proofs, it is possible to come up with invalid arguments; that is, those that try to make assumptions that are known to be false. It is okay for an argument to make assumptions, but if we know that an assumption made by an argument is false, then we ought to know enough to ignore this argument. We explicitly restrict our attention to arguments that are not yet known to be invalid. The details of how arguments are generated in practice, and so on, are discussed in the section The Heuristic Level: Efficient Inferencing in Cyc.

It is also possible to come up with arguments for both P and ¬P (and, indeed, to have multiple distinct arguments for each). Combining arguments is tricky and is handled as follows: Given a set of arguments for P and for ¬P, Cyc compares these arguments and decides what to add to the knowledge base—P, ¬P , or neither. We do this comparison by making arguments first-class objects in the CycL language and using axioms in the Cyc knowledge base that capture our intuitions of various aspects of default reasoning. The argumentation axiom, which captures the gist of this approach, is as follows:

$$(\forall a)\ (argumentFor(`p, a) \wedge \neg invalidArg(a) \wedge \neg(\exists a1)\ (argumentFor(`\neg p, a1) \wedge \neg preferred(a,a1) \wedge \neg invalidArg(a1))) \supset True(`p) .$$

Additional axioms are used to specify when an argument is invalid (for example, $(True(`\neg p) \wedge inArgument(`p,a)) \supset invalidArg(a))$ and axioms that can conclude that one argument is preferred over another (a few of these are given later). These axioms form the core of the real default reasoning.

It is usually not possible to prove that a better counterargument to some proposition does not exist or prove that some argument is not invalid. To deal with this problem, CWA is made for the predicates *argumentFor* and *invalidArg*, and this closed-world assumption is what provides the required nonmonotonicity to the language. This axiom uses the truth predicate *True*, and to avoid the possibility of paradoxes, we (following Tarski) allow this predicate to differ from the actual truth value of the sentence for certain sentences (those that are likely to be paradoxical). In other words, the following is not an axiom schema in our knowledge base:

$True(`p) \Leftrightarrow p$.

The preference axioms and the axioms that define the predicates *argumentFor* and *invalidArg* are not part of any metatheory; they are regular axioms of the knowledge base. The salient aspect of our approach to doing default reasoning is that most of the work is done using axioms explicit in the knowledge base and not wired into the logic. This approach provides us greater flexibility and control and makes it easier to fix problems if inadequacies are detected. Adding and removing axioms from the knowledge base is strongly preferable to changing the logic, especially when a massive knowledge base already exists and assumes a certain logic.

We currently use a number of different criteria for comparing arguments: prefer inferences with shorter inferential distance (Touretzky 1987), prefer causal explanations (Lifschitz 1987), prefer constructive arguments over nonconstructive ones, and so on. We are also trying to construct a taxonomy of standard types of arguments that capture different dialectic patterns. For example, there is the concept of a narration (one kind of argument) that corresponds to giving a sequence of events that lead to some assertion (from some starting point) as an argument for why this assertion might be true.

What aspects of our default reasoning scheme need more work? Surprisingly, we would not include the logical formalism used. However, we would include issues such as what are reasonable preferences and dialectic patterns. Currently, we are actively engaged in this research (Guha 1990b).

Microtheories and the Representation Language

One of the facilities available in CycL is the ability to say that a set of sentences constitutes a theory. For example, one set of sentences represents a naive theory of physics (NTP). Here is a typical NTP sentence, which says that if something is not supported, then it falls.[5] (Of course, this heuristic is not always true; it fails for balloons, objects in outer space, and so on. This theory should not be used when dealing with such objects.)

$ist((\forall x\ \neg supported(x) \supset falls(x)), NTP)$.

These theories, such as NTP, are referred to as microtheories (µTs). They are first-class objects in the CycL language and are usually given descriptions related to their scope, when they should be used, and so on. This facility is extremely useful in forming theories and abstractions at different levels of granularity. In addition to avoiding the inconsistencies that often accompany the careless merging of different abstractions, microtheories give us a convenient formalism for using multiple representations, reasoning about when to use which one, and representing within a context (the microtheory could correspond to something like a discourse context). They also enable us to use contextual information to simplify the statement of axioms.

The description of the assumptions made by the microtheory—what is left implicit, and so on—that is associated with it form its *context* (McCarthy 1987b). A distinction is made between the *propositions* that constitute the microtheory (that is, the deductive closure of the axioms in the microtheory) and the *context* associated with it. The context is a rich (McCarthy and Hayes 1987) object and usually carries information that can be used to universalize (Quine 1969) the contextual representation of the microtheory. The addition of a fact to a microtheory always changes the theory associated with it but need not always change the context of the microtheory.

Statements about the world (as opposed to statements about a particular microtheory) have to be within some microtheory or other. One distinguished microtheory corresponds to the most expressive microtheory (MEM). In this article, as a notational convention unless otherwise mentioned, if no microtheory is explicitly associated with a statement, the microtheory in which this statement is true is taken to be MEM. Most of the discussion of the ontology in this article is of that in MEM, although the last subsection of this section does discuss the role of microtheories in the ontology.

Here, we end our discussion of the epistemological level and proceed to a description of some of the techniques used at the heuristic level to speed up inference.

The Heuristic Level: Efficient Inferencing in Cyc

The heuristic level is meant for doing inferencing. As opposed to the epistemological level, where we tried to avoid superfluous constructs, the heuristic level incorporates a host of logically superfluous mechanisms for improving efficiency.

The Functional Interface to CycL. The user can interact with Cyc at the epistemological level (figure 1) using a set of functions called the *functional interface*. The functionality of each of these is implemented at the heuristic level. The functions are Assert, Unassert, Deny, Justify, Ask, and Bundle.

Assert: (Σ x KB $\rightarrow$ KB): Given a sentence σ and a knowledge base, after Assert(σ, KB), we get a new (modified) knowledge base in which σ is an axiom.

Unassert: (Σ x KB $\rightarrow$ KB): This function is the "undo" of Assert. After Unassert(σ, KB), σ is not an axiom (although it might still follow from axioms in the knowledge base).

Deny: (Σ x KB $\rightarrow$ KB): This operation is similar to but stronger than Unassert. It tries to produce a knowledge base in which the sentence σ is neither an axiom nor a theorem. Deny is stronger than Unassert in the case where σ is currently a theorem in the knowledge base—follows from axioms Asserted into the knowledge base—but not an axiom. In such a case, Unassert would do nothing (that is, σ would still be a theorem afterwards). It is possible that after Deny, neither σ nor $\neg\sigma$ is a theorem or axiom, which contrasts with Assert($\neg\sigma$, KB), after which $\neg\sigma$ would be an axiom. Thus, Unasserting σ is weaker than Denying it, which, in turn, is weaker than Asserting its negation.

Justify: (Σ x KB → sentences): If sentence σ is true in the knowledge base, then Justify(σ, KB) should return a minimal subset of the knowledge base from which σ can be derived.

Ask: (Σ x KB → truth-value bindings): Ask is used to test the truth value of a statement and, if the expression contains some free variables, to find which variable bindings make it true. An optional argument (UNWANTED-BINDINGS) turns Ask into a generator; that is, each repeated call yields new sets of bindings.

Bundle: (sequence of functional interface statements): This facility performs a series of calls to the previous five functional interface functions as one atomic macro-operation. This operation is of great pragmatic benefit for two reasons: (1) The first few operations might violate some integrity constraints, and later ones might satisfy them again. (2) The bundling allows the heuristic level to be smart about which assertions it has to undo. For example, suppose we change an axiom from "Southerners speak with a drawl" to "Southerners over age 2 speak with a drawl." We make this change by issuing an Unassert and an Assert. Each of these two operations would cause about n modifications (assuming an average lifespan of 72 years and a uniform age distribution in the population of n Southerners). However, if we Bundle the two operations, then only $n/35$ modifications are done rather than $2 \cdot n$.

The concept of a functional interface, with functions such as Assert and Ask, has been around in computer science and AI for some time (Brachman, Fikes, and Levesque 1986). We tailored it somewhat for our purposes and increased its pragmatic usefulness by adding some new constructs (such as Bundle and Justify) and teasing apart old ones (such as Unassert(σ, KB) versus Deny(σ, KB) versus Assert(¬σ, KB)).

Default Reasoning Modules (The Structure of the Heuristic Level). Most of the gain in processing speed at the heuristic level comes about because of the way we implement Ask. (Much of the complexity at the heuristic level arises from the need to do Deny and Unassert properly, but this is another issue [Pratt and Guha 1990].) The structure of the heuristic level is based on default reasoning. As we mentioned previously, defaults are implemented in CycL by way of an argumentation axiom. The argumentation axiom is just like any other axiom at the epistemological level. However, because it's used often, at the heuristic level, there are some special procedures for incorporating it so that certain special cases can be run efficiently. Here, then, are the four modules that make up the structure of the heuristic level:

Argument Generator: Given a sentence σ, this module tries to generate an argument for it. Recall from the section on default reasoning that an argument is similar to a proof that can include assumptions.

Argument Comparator: Given a set $\{A_i\}$ of arguments for and against a sentence σ, this module decides on a truth value for σ by using knowledge base axioms to check each A_i for invalidity and decide which noninvalid A_is are preferred over others.

Argument Retractor: When the truth value of a sentence σ changes, this module ensures that truth values of other sentences that depend on σ are also updated. Not surprisingly, the argument generator module is, in practice, tightly integrated with this module.

Contradiction Resolver: This module is responsible for detecting and resolving contradictions. For example, suppose we have an axiom $p \supset q$ involving no abnormality literals, and p is true, yet q is found to be false (because of arguments for $\neg q$). This module would notice such a contradiction and attempt to resolve it, for example, by retracting some weak (default) assertion that was used in the argument for p.

Given a sentence σ, Ask first tries to find arguments for it. If it can, it then tries to find arguments against it. These arguments are then compared (which can invoke preference axioms from the knowledge base and recursively involve calls on Ask), and the final truth value is decided based on this comparison. Note that the comparison process could fail (the arguments could be incomparable), which results in neither σ nor ¬σ being concluded.

The representation of default axioms at the heuristic level is different from that at the epistemological level. First, all the default axioms sharing the same *ab* literal are grouped into classes. For instance, the set of all rules whose antecedents (left-hand sides) contain ... $\wedge \neg ab_{37}(\ldots)$ . . . are grouped into one class. Second, each class is labeled with the *ab* literal (in this case, ab_{37}). Third, therefore, we can strip all the *ab* literals from all the default axioms. Fourth, although the argument generator can ignore these labels, which increases its efficiency, the argument comparator can still make use of them. However, certain precautions have to be taken against the knowledge base appearing inconsistent to the argument generator (because of some default not holding.)

Although the epistemological level has only two truth values (true and false), the heuristic level uses five truth values (true, default true, unknown, default false, and false) to label sentences in the knowledge base. True-false sentences are those that are monotonically true; that is, the addition of new facts can't cause them to be retracted (although even monotonic axioms can later be explicitly Unasserted, of course). Default true-false sentences don't have this property. Unknown is used for sentences for which there are unresolved conflicting arguments (Ginsberg 1987). The presence of five heuristic-level truth values helps in making the TMS-related bookkeeping activities more efficient.

A number of the statements at the epistemological level are of the form (*¬ab-literal ⊃ ground-formula*). These are called *local defaults* (and simply translate to the ground formula with a truth value of default true at the heuristic level), and the heuristic level provides special support to handle these efficiently.

Speeding Up the Argument Generator Module. The bulk of Cyc's time spent inferencing is done by the argument generator module. A number of techniques have been introduced to make the argument generator (and conclusion retractor) modules work more efficiently. These techniques fall into three categories: highly specialized inference rules, domain-specific inference modules, and dependency analysis of the knowledge base.

Highly Specialized Inference Rules. In the knowledge base, a number of axioms are instances of the following schema:

(s1 x y) ∧ (s2 x z) ⊃ (s1 z x) .

Here is one such axiom:

(owner x y) ∧ (parts x z) ⊃ (owner z x) .

If the owner of JoesCar is Joe, and one of the parts of JoesCar is JoesCarsSteeringWheel, then it's reasonable to conclude that the owner of JoesCarsSteeringWheel is also Joe. Here is another example of an axiom that fits the same schema:

(lastName x y) ∧ (sonOf x z) ⊃ (lastName z y) .

A new rule of inference, called transfersThrough, is used at the heuristic level to exploit the common syntactic structure of axioms matching this schema. Thus, the previous two axioms would be represented at the heuristic level as follows:

(transfersThrough owns parts)
(transfersThrough lastName sonOf) .

Currently, there are numerous such schemas, each of which has been made into a rule of inference, each schema corresponding to a category of axioms with isomorphic syntactic structure. Two other schemas are inheritance and automatic classification.

Associated with each such inference schema are specialized procedures for speeding up this sort of inferencing. For example, a certain amount of compilation can be done that drastically cuts down on conjunct ordering and unification at run time. Also, the stack used by Lisp itself can be used instead of using binding lists. Many of these savings are similar to those obtained by the Warren Machine–like compilation of Prolog programs (Warren 1983). In particular, each schema has specialized procedures for recognizing instances of the schema (for translating from the epistemological level to the heuristic level), applying instances of the schema to derive conclusions, and performing bookkeeping functions such as storing justifications.

We have also built a facility to help a user add new inference rule schemas. That is, one specifies a new schema, and Cyc automatically generates all the code needed to efficiently implement it—the types of specialized procedures previously itemized. Currently, this code generator can only handle schemas with no sentential variables, but even with this restriction, the code-generating facility is extremely helpful.

Domain-Specific Inference Modules. The previous technique—introducing specialized inference mechanisms—was based purely on the syntactic structure of the axioms and had nothing to do with the domain the axioms dealt with. However, at times, we can exploit some special properties of a set of domain-specific axioms or the domain-specific use of a set of general axioms. Some examples of such axiom clusters that Cyc currently optimizes in this way are those related to temporal reasoning and quantity arithmetic. In the case of temporal reasoning, for example, many tasks can be formulated as simple graph-search problems over the graph whose nodes are time points and whose arcs are primitive temporal relations (*before*, *after*, and *simultaneousWith*.)

In other words, a certain set of axioms about time can be compiled into a graph-search procedure. It should be noted that although at the heuristic level, nothing more than the program might be representing them, these axioms do still explicitly exist at the epistemological level.

Dependency Analysis of the Knowledge Base. Over the years, AI researchers have developed a number of stand-alone modules (for exam-

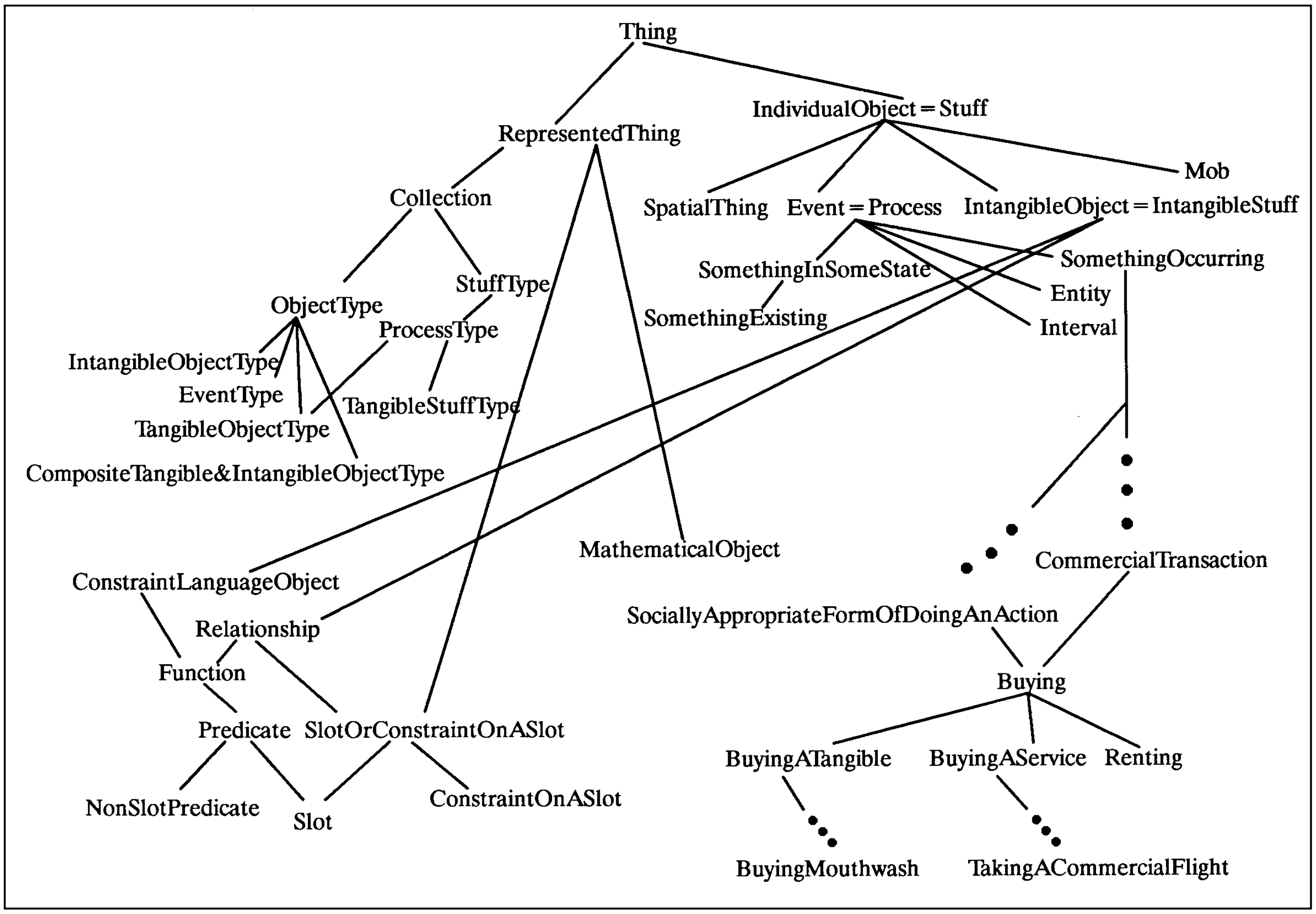

Figure 2. Some of Cyc's Collections.
(a) For millenia, ontologists have derived power through judicious choice of categories and predicates. Here, were see a few of Cyc's 5000 collections, related by genl and spec (superset and subset) links. Of course, because such arcs only depict one of Cyc's 4000 slots, many concepts that are closely related in the knowledge base (for example, ProcessType and Process) aren't connected in the diagram. (b) Although we make use of taxonomic classification, most of the power in the knowledge base derives from the particular axioms (facts, heuristics, and so on) such as the five depicted here (drawn from naive theories of buying, consuming, and so on). Axioms typically cut across category boundaries; for example, a single assertion about mouthwash might involve human emotions, cognitive-processing limitations, buying, liquids, hygiene, and containers. (c) Here is the way we enter the first two of these axioms into Cyc. Free variables (such as buyer, agent) are assumed to be universally quantified. We previously specified a context, a particular microtheory, in which these are full-fledged CycL epistemological-level axioms. Cyc can then lift them, adding clauses to decontextualize them as necessary. (For example, owning an object is only a necessary precondition to consuming it in the context of law-abiding, calm, responsible parties in a partially capitalistic economic setting, and so on). In each case, the tell-ask interface converts these clauses into efficient CycL heuristic-leel expressions

ple, TMSs) that could be used with any problem solver. To make them problem solver independent, their operation was usually made independent of the structure of the knowledge base on which the problem solver was operating. However, we have found that it is possible to obtain significant improvements in problem-solving efficiency by using an analysis of the structure of the knowledge base axioms.

For example, consider the problem of detecting purely self-justifying sequences of statements (for example, A justifies B, which justifies C, . . ., which justifies B, and then A is retracted, but the rest remain). Avoiding such situations is a traditional source of space-time complexity in TMSs. However, a dependency analysis of the knowledge base axioms could reveal the circumstances in which there could possibly be circular justifications and the set of sentences that might be involved in the circular justification. Having this information can vastly reduce the time required to search for such circularities. For example, it turns out that in Cyc's current knowledge base, (1) only a small fraction of the 4000 kinds of slots could possibly participate in circular lines of reasoning and (2) such garbage-collectable chains are precisely characterizable and, hence, easily recognizable when they occur. These two knowledge base–specific properties make the problem of detecting them computationally feasible in practice, even though this detection is complex and expensive in the worst case. Although these modules are now making strong assumptions about the structure of the representation used by the problem solver, the resultant improvements in efficiency are worth it (Guha and Lenat 1989a).

In addition to the specialized inference procedures, Cyc has a general-purpose inference mechanism that is capable of a much larger (but incomplete) category of inferences. This general inference engine is similar to a unit preference resolution theorem prover, and with this inference mechanism, CycL is complete if attention is restricted to a horn subset of the knowledge base.

When dealing with mechanisms other than the domain-specific inference mechanisms, a depth-first iterative deepening procedure is used for the search. Resource-limited reasoning is implemented by using indexical functions that specify various bounds on resources (such as the time available, the cutoff depth for search).

In addition, part of these inference mechanisms is represented in Cyc, which allows Cyc to use an agenda to perform a best first search, using various heuristics to control the search strategy. However, to date, the performance of the iterative deepening strategy alone has been good enough that this metalevel (agenda) mechanism is rarely used. Here, we end the discussion of CycL and proceed to our discussion of the contents of the knowledge base.

The Current State of Cyc's Ontology

The ontology of Cyc primarily includes categories and things in the world; it also includes reified assertions and internal machine objects such as strings, numbers, lists, and program code (figure 2).[6]

The next subsection introduces some of the basic concepts and distinctions used; the latter subsections discuss representation issues such as time, events, agents, and causality. One caveat: This discussion is only meant to give the flavor of the kind of things that are in the Cyc knowledge base; it is by no means a comprehensive overview of what is there, nor does it adequately capture the breadth of the current knowledge base. Further details on the ontology can be found in Lenat and Guha (1990).

Recall that the epistemological level is meant for communicating the contents of Cyc independent of the inferencing hacks that are used for efficiency down at the heuristic level. Hence, most of our discussion of the ontology of Cyc's knowledge base will be at the epistemological level, not the heuristic level.

We also mentioned that the knowledge base is organized in different microtheories along with information about when to use which microtheory, the information that is implicit in each microtheory, and so on. Different microtheories can use different abstractions of time, actions, space, objects that are known about, and so on. In this section, most of our discussion is in the context of the most expressive microtheory (MEM). MEM's expressiveness is a good measure of the expressiveness of the Cyc ontology as a whole. Therefore, the assertions we make have no explicitly mentioned microtheory; it should be assumed to be MEM. In the subsection Microtheories and the Ontology, we discuss the need for, and the impact of, having multiple microtheories.

Some Basic Concepts and Distinctions

The ontology of Cyc is organized around the concept of *categories*. We also refer to them as classes or collections. The categories are organized in a generalization-specialization hierarchy (a directed graph, not a tree because each category can have several direct generalizations) (figure 2).

The Cyc predicates relating a category to its immediate supersets and subsets are, respectively, genls and specs. The instances of a category are its elements or members; the inverse of this relation is instanceOf. It should be noted that unlike many frame systems, a strong distinction is made in Cyc between the two relations instances (elements) and specs (subsets). The relation between Person and MarvinMinsky is different from the relation between Person and ComputerScientist. Another example is genls(Texan, American) and instanceOf(Lenat, Texan). Although we frequently use set-theoretic notions to talk about collections, these collections are more akin to what Quine (1969) termed natural kinds than they are to mathematical sets. This distinction becomes apparent later when we start ascribing various intentional properties to collections.

Because the generalization-specialization hierarchy is important, we start off by discussing some of its highest nodes, why they are in certain unintuitive genls-specs relations to other nodes, and so on.

The universal set is called Thing. One of its partitionings is into the two sets InternalMachineThing and RepresentedThing. Instances of InternalMachineThing include the number 5 and the string "DOG," that is, things for which the representation is provided by the Lisp substrate on which CycL is written. Instances of RepresentedThing are things such as Chair for which only a representation is provided by CycL. This distinction is useful when deciding whether to use model attachments (Weyhrauch 1986).

Another partition of Thing is into IndividualObject and Collection. IndividualObjects are elements such as Fred, TheWhiteHouse, and TheFourthOfJuly1990, that is, the non-sets. They can have parts but not instances. Instances of Collection include Thing (the set of all things), Chair (the set of all chairs), Texan (the set of all Texans), Dining (the set of all dining events), and so on.

Predicates are all strongly typed, and a single category from the Cyc hierarchy has to be specified as the type for each argument.[7] This was a conscious design decision and has tremendous heuristic power as the knowledge base is built. Namely, when knowledge enterers have an urge to define a new kind of slot, they must either select or define the domain of the slot. Usually, the slot is separately worth existing only if the domain is; so, our single category constraint frequently gives the knowledge enterers a well-needed doublecheck on what they were about to do.

It should be noted that predicates such as age and weight can't legally be applied to collections (such as Chair). To rephrase, because Chair is a set, an abstract mathematical entity, it can't have a weight or an age (it can, of course, have many other slots such as cardinality). Of course, we could discuss weight (Chair905)—the weight of an element of the set Chair—but this element is different.

In addition to collections of individuals, we also have collections of collections. For example, PersonType is a set whose elements include Person, ComputerScientist, and Texan, which themselves are collections whose elements include Lenat, Bledsoe, and so on. The hierarchy folds into itself at this level; that is, we don't have collections of collections of collections, and Collection is an instanceOf itself.[8]

The predicates themselves are first-class objects in the language and can be used as arguments to other predicates (this is a second-order–like construct that can easily be first ordered). Although some of our editing tools (and internal data structures) gather into frames the set of assertions that are binary predicates sharing a common first argument, this distinction is merely a heuristic-level (and user interface) one; there is nothing special about binary versus other arity predicates at the epistemological level.

We are now ready to discuss some of the representation issues. First, we discuss the distinction between Substances and IndividualObjects; then we proceed to how we represent events, causality, intelligent agents, and so on.

Substances versus Individual Objects

If you take a piece of wood, and smash it into 10 pieces, each of them is still a (albeit smaller) piece of wood. However, if you do the same to a table, each piece is not a (smaller) table. Substances are usually referred to in linguistics and philosophy as *mass nouns*; some of them are obvious (sand, peanut butter, air) and some less so (time, walking). We view the concept PeanutButter as the collection of all pieces of peanut butter.

Every individual is made of some substance or other. If we don't have a single type of substance of which this individual is composed, we can define a new one (BertrandRusselStuff? ugh!); use a more general substance (AnimalMatter); or even fall back on the most general kind of substance, Substance.

Conversely, every piece of any substance—say, this particular piece of peanut butter over here—is clearly an individual. This leads to some interesting relations between substances and individuals:

Because every individual is a piece of some substance, IndividualObject $\subset$ Substance. However, any particular piece of any substance is an individual, and because the category corresponding to a type of substance is nothing but the set of its pieces, Substance $\subset$ IndividualObject.

Thus, rather surprisingly, the two properties are extensionally equivalent, which explains the IndividualObject = Substance node in figure 2. We still choose to distinguish between them because they have different intensional descriptions. The different substances (Wood, PeanutButter, Air, and so on) are all instances of the collection SubstanceType, and the collections of individuals (Table, Person, Number, and so on) are instances of ObjectType. We shortly describe how this difference in intensional description is made use of.

Certain properties are intrinsic (that is, if an individual has them, slices of this individual also have them, at least as a default), and other properties are extrinsic (that is, the parts don't inherit this property from the whole). The notion of intrinsicness is closely related to that of substances: Consider a par-

The predicates themselves are first-class objects in the language and can be used as arguments to other predicates . . .

We can cut something up spatially . . . but we can also cut it up temporally.

ticular table made entirely of wood—Table103. It inherits various default properties from Wood, which is the kind of substance it's an instance of (properties such as density, flash point), and it inherits other properties from Table, which is the kind of individual object it's an instance of (properties such as number of legs, cost, size). The former properties are intrinsic; the latter are extrinsic. This is no coincidence! We have noticed that an object (typically) inherits its intrinsic properties from whichever instances of SubstanceType it's an instance of, and it inherits extrinsic properties from whichever instances of ObjectType it's an instance of.

Thus, we now have a way of predicting for most predicates P whether it will be intrinsic. Namely, see whether P's domain is an instance of SubstanceType or ObjectType. Now we see the importance of having collections of collections; we could actually dispense with the concepts Substance and IndividualObject (because they are coextensional), but we can't do without SubstanceType and ObjectType.

For each type of substance, the granule predicate associates are the building blocks out of which the substance is built; for example, granule(Wood, PlantCell), granule(Sand, GrainOfSand), and granule(ThrongOfPeople, Person). Considering a portion of a substance smaller than a mob of its granules is risky; many of the default properties are violated as one approaches this level of granularity. Strictly speaking, we could conceptually carve a throng of people into all their heads and the rest, and neither alone would be an instance of ThrongOfPeople. Thus, we place a further restriction and say that the substancehood principle applies not only to portions much bigger than a granule but also only to portions that are just a contiguous mob of granules. As one drops below the grain size of a substance, one finds individual objects (which, in turn, are made out of substances). For example, MilitaryHardware is a kind of substance (that is, an instance of SubstanceType); its granules include guns and tanks, which are individual objects. Each instance of Gun is, in turn, made of some substance such as Iron. The granule of Iron is IronAtom, which is an ObjectType, and so on. This alternating individual-substance-individual-. . . continues at higher and lower (particle physics) levels.

Processes, Events, and Persistent Objects

To this point, we have used the terms piece and cutting up in a loose fashion. Actually, these terms can be used in two senses—spatially and temporally—and we now examine them both. This leads to more general issues about events that occur over some time interval and objects that exist over some time interval.

We can cut something up spatially (as we did with the piece of peanut butter), but we can also cut it up temporally. For example, consider a process such as walking. In Cyc's ontology, we have the collection Walking, which is the set of all walking events. Consider one of its instances, a particular event in which, say, you walk to the corner mailbox and back home. Imagine a videotape of this event, and now consider some contiguous one-minute segment of the tape, say, the third minute of it. If anyone watched just this minute, they would report that this minute was itself a (albeit shorter) walking event, that is, another instance of Walking.

In the last subsection, we noted that the class Wood has an interesting property: When a member of the class (for example, WoodenTable001) is carved into several pieces, each one is still an instance of Wood. We then said that Wood is a type of substance (that is, instanceOf(Wood, SubstanceType)), and we could use such substance-like categorization to decide on intrinsicness of properties. Here, we are seeing an analogous phenomenon: A class Walking, which is a set of events, has the property that when a member of the class is temporally carved into pieces, each one is still an instance of Walking. We say that Walking is a type of temporal substance, a type of process. That is, Walking is an instance of ProcessType and a spec of Process. This analogy turns out to be more than superficial. Indeed, ProcessType is a spec of SubstanceType; so, Walking is an instance of SubstanceType. SubstanceType is partitioned into TangibleSubstanceType and ProcessType. Wood, for example, is an instance of TangibleSubstanceType.

Similarly, ObjectType is currently partitioned into TangibleObjectType and EventType. Although Walking is a type of process, WalkingToTheMailboxAndBack is not. If you imagine the third minute of the 10-minute WalkingToTheMailboxAndBack event, it is still an instance of Walking, but a stranger watching just this minute would not say that it was an instance of someone walking to a mailbox and back home—neither your home nor the mailbox might be visible anywhere on the videotape during this minute! The relationship here between Walking and WalkingToTheMailboxAndBack is indeed the same as the one between Wood and Table. Table is an instance of TangibleObjectType, and

WalkingToTheMailboxAndBack is an instance of EventType.

Earlier we saw that, surprisingly, Substance and IndividualObject were coextensional; with a similar argument, it turns out that Process and Event are coextensional. This is why ProcessType and EventType are actually the more useful collections to have explicitly represented rather than Process and Event.

There are now two types of intrinsicness as well: A property can be spatiallyIntrinsic or temporallyIntrinsic. If you imagine the particular event in which you walked to the mailbox and back home, it is an instance of Walking (from which it inherits default values for rate of speed, step size, amount of attention required, and so on) and an instance of WalkingToTheMailboxAndBack (from which it inherits default values for destination, duration, and so on). Sure enough, this third minute of the videotape would agree with the entire video on properties such as rate of speed but would differ radically on properties such as duration. rateOfMotion is temporallyIntrinsic, just as density is spatiallyIntrinsic.

Consider Table001, a particular table, an instance of the category Table. It persists for a lifetime, an interval of time before and after which it doesn't exist. Consider a temporal slice of Table001, such as the decade it was owned by Fred. This slice is also an instance of Table. This fact is interesting because it means that the category Table is an instance of ProcessType, and Table001 is an instance of Event! The process going on is existing.

Not surprisingly, a number of categories exist in our ontology whose instances are space-time chunks that exhibit sufficiently persistent properties that it makes sense to associate a notion of identity with these objects. The category of such things is called SomethingExisting, and this category is an instance of ProcessType. Because all physical objects (which have any persistent identity) exhibit this property, TangibleObject is an instance of ProcessType; so, anything that is spatially substance-like is also temporally substance-like, although the converse is not true.

This view of concepts such as Lenat or Table001 (that is, as Events) is interesting. We view these objects as space-time chunks, and we call the temporal pieces of these—for example, LenatDuring1990, Table001WhileBeingEatenOn—*subAbstractions* of the larger piece. SubAbstractions can, of course, have further subAbstractions. Each maximal subAbstraction (for example, Lenat or Table001) is called an Entity; that is, entities cannot have superAbstractions. Being space-time chunks, these subAbstractions have temporal properties such as duration (the duration of Lenat is his lifespan), startingTime and endingTime.[9] This subabstraction scheme shares similarities with the histories approach of Hayes (1985).

Not all objects that have temporal extents need exhibit enough persistence to warrant a persistent identity. Consider Roger dining at a restaurant one evening. We might consider a system consisting of Roger, the waitress, the table, cutlery, food, and so on, interesting enough to create an explicit object RogerDiningEvent2317 to represent this system. However, this object has no temporally persistent properties, has poorly defined boundaries, and is of little interest after Roger walks out (except perhaps as an example in an AI article). Such objects are instances of SomethingOccurring, which is another important instance of EventType; these objects correspond to names that usually go by the names of actions, scripts, or processes, and they are rich objects (McCarthy and Hayes 1987).

The parts of an instance X of SomethingOccurring are referred to as its actors. These parts include all the persistent objects (that is, instances of SomethingExisting) that are involved in X. The actors of RogerDiningEvent2317 include the waitress, the food, the table, Roger, the bill, and so on. There are useful specializations of actors, such as performer, objectActedUpon, and instrumentInAction. The various actors in an event usually undergo some change either during or after the event occurs (that is, the subAbstractions of the actors during or just after the event are different from the subAbstractions just before).

No fixed set of slots appears in an action such as RogerDiningEvent2317. The ones that appear could depend on what happened there and how much Cyc knows about it; and in general, more details can always be added later. RogerDiningEvent2317 can later be arbitrarily elaborated along any dimension, for example, by specifying P(RogerDiningEvent2317) for predicates P that were not known at the time that RogerDiningEvent2317 was created!

It should be noted, however, that no ad hoc distinction is made about what kinds of events can cause changes in the properties of instances of SomethingExisting. In fact, because some of the properties of things change simply by their existing (for example, their age), it could well be the case that the properties of something change substantially even though it was not an actor in any instance of SomethingOccurring.

Any instance of Event can have temporal

. . . by associating temporal extents with objects as opposed to reifications of propositions, we get a certain added expressiveness.

properties (duration, endsAfterTheStartOf, and so on). We use two abstractions of time to specify these temporal properties: interval based and set based.[10] Let's first discuss the *interval-based abstraction of events*. We can define a number of relations between events using the two primitives before and simultaneousWith that can hold between the starting or ending times of these intervals (which might have discontinuities in them). Thus, for example, we define the binary temporal relation startsBeforeStartOf as

(∀x,y) (startsBeforeStartOf(x, y) ⇔ before (startingTime(x),startingTime(y))) .

Why do we need a second abstraction of time? The interval-based abstraction of time makes it awkward to say things such as "people don't eat and sleep at the same time" because we are not dealing with a single convex interval. In such cases, it is easier to abstract the times when x is eating and the times when x is sleeping as sets of points. Then, based on this set-based abstraction, we use set-theoretic relations such as intersects, subset, and disjoint to state axioms such as "people don't eat and sleep at the same time." In this case, the sentence would just be an assertion that for each person, the piece of time they're eating and the piece of time they're sleeping, viewed as sets of points, have an empty intersection.

It is interesting to note that by associating temporal extents with objects as opposed to reifications of propositions, we get a certain added expressiveness. For example, it is easy to express statements of the form "Fred when he was 39 liked his house as it had been 20 years earlier" in this formalism, and it is difficult to do so with formalisms that associate time with propositions (or their reifications). However, there is a high cost associated with this expressiveness. Given n entities and m intervals, we can have as many as $n \cdot m$ subAbstractions (that is, have to create $O(n \cdot m)$ objects) using the set-of-points-based abstraction; using the interval-based formalism, we would only need $O(n + m)$ objects.

A vast majority of the statements we would like to make relate coTemporal objects; that is, the temporal extents of the objects related is the same. For example, FredIn1972 could not be married to EthelIn1958. Similarly, Fred lived in House001 from 1965 through 1975 means that livesIn(FredFrom1965To1975, House001From1965To1975).[11] To exploit this regularity, we use the predicate holdsDuring. Instead of creating innumerable pairs of concepts never again needed (FredIn1958, EthelIn1958; FredFrom1965To1975 and House001From1965To1975; and so on), we just assert

holdsDuring(Fred, livesIn, House001, 1965-1975)

and

holdsDuring(Fred, marriedTo, Ethel, 1958).

Similar constructs that allow temporal extents to be associated with reifications of propositions are used when dealing with predicates of arity greater than two.

It turns out to be notationally much simpler to write complex axioms (where specific instances of SomethingExisting are replaced by variables) using the subAbstraction formalism, but it is more efficient to inference using the holdsDuring predicate. The heuristic level provides special support for doing exactly this bifurcation: allowing the expressive but inefficient Subabstraction formalism for stating axioms but using the more efficient holdsDuring predicate for doing inference whenever possible.

In addition to instances of SomethingExisting, such as Fred , and instances of SomethingOccurring, such as FredBeingBorn, we also recognize changes in properties of persistent objects as first-class events. If Fred was hungry before going to the restaurant and not hungry afterwards, we can consider this change as an object. Formally, this corresponds to reifying a sentence that specifies that he was hungry at some time and not hungry at some later time into an object and making this resultant object an event (because one can associate temporal properties with it).

Temporal Projection. When one of the properties of an instance of SomethingExisting changes, it's not likely to affect all (or even many) of its other properties (McCarthy and Hayes 1987). For example, when Guha gets a haircut, it doesn't affect his address, languagesSpoken, birthDate, and so on. This fact is not surprising because a useful set of properties is useful in part because the properties are largely independent.

Associated with each ground formula is an interval of persistence. Thus, if we knew that Fred was happy at time *t0*, and the persistence interval of this time was *I1*, then given any point in time between *t0* and *t0* + *I1*, we can conclude that Fred was happy at this time point. Usually, we associate default periods of persistence with classes of propositions by using axioms, which are called *temporal projection axioms.* These axioms enable us to *project* (infer a good guess for) Fred's name and gender years in the future or past, his hairStyle months in the future or past, his mood seconds in the future or past, and so on, based on the values of these attributes at some given time.

These temporal projections are only defaults. If the evidence is contrary to these projections based on particular actions that have taken place, this contrary evidence usually overrides these projections. Also, if an event that is capable of changing this property is known to have taken place, even if the effects of the event can't be computed (possibly because of missing information), we don't project this property across such an event.

Associating specific finite periods of persistence with propositions has advantages over using a frame axiom (McCarthy 1987a) to allow for extended projection. It provides a simple scheme for accumulating the information at a finer granule (regarding the possibility of different events capable of changing this property that are likely to occur) into a coarser granule in terms of this interval. However, this association introduces the following problem: If our knowledge that Fred was happy (or a gun was loaded) at *t0* came from some source other than temporal projection, we are willing to say that until time *t0* + *I1*, he's happy (or the gun is still loaded). However, we don't want to repeat this argument and say that at time *(t0* + *I1)* + *I1* he's still happy, and the gun is still loaded. That is, we want to project only from a base time point where we had this other source of information (that is, some justification other than temporal projection) about the fact in which we are interested. Notice how we escape from this classic problem by making use of the ability to explicitly refer to justifications for facts (which we obtained using reflection) to state this dependence (Guha 1990a).

Temporal projection has more to it than just a frame axiom. We mentioned earlier the concept of a narrative as a kind of argument. These narratives, together with argument preference criteria based on notions of causality, are used to actually do the temporal projection. The details of this can be found in Guha (1990b).

Causality. Most treatments of causality (in AI) proceed by labeling some appropriate subset of occurrences of material implication as causal. We do this labeling by using a causal relation whose argument is the reification of a sentence involving a material implication. For convenience, we refer to $((p \supset q) \wedge (causal\ '(p \supset q)))$ as (Cause *p q*). Let us take a closer look at the axioms that specify the meaning of Cause:

First, suppose we assert (Cause *p q*). Then it follows that $(p \supset q)$. In other words, Cause is a strictly stronger notion than material implication.

Second, suppose (Cause *p q*), and *p* and *q* are ground sentences and true. Then *p* and *q* must refer to events (which in Cyc is anything that can have temporal attributes). That is, *p* and *q* must have at least one object constant that is an event.

Third, suppose (Cause *p q*). Then every single event referred to in *p* must startBeforeTheEndingOf every event in *q*.

Fourth, given any atomic ground sentence *q* that refers to an event, either *q* should be basic, or there should be some sentence *p* such that (Cause *p q*) is true. Intuitively, *q* being classified as basic corresponds to the notion of it being unexplainable.

Fifth, given a statement of the form (Cause *p q*), either this statement is "basic," or a sequence of sentences exists of the form (Cause *p a*), (Cause *a b*) . . . (Cause *m q*), that is, some mechanism that implements this causal relation.

Both the fourth and the fifth axioms are extremely strong statements to make, which is why the notion of basic sentences was included. No commitment is made about which occurrences of implication are to be labeled as causal. The aim of the previous formalism is to provide a facility to state and experiment with various heuristics for doing just this labeling. As with default reasoning, we chose to have axioms in the knowledge base provide the guidance and decision making rather than wiring it in in some fashion.

Actions and Concurrent Processes. Each action (that is, instance of SomethingOccurring) has a set of axioms associated with it specifying its preconditions, its postconditions, and constraints on the actors during the event. A tangible object is a structured arrangement of its physicalParts (for example, Table001 breaks down into its legs, top, and so on). In much the same way, an action is a constrained arrangement of its subEvents and

Associating specific finite periods of persistence with propositions has advantages over using a (conventional) frame axiom. . .

actors; for example, RogerDiningEvent2317 breaks down into subEvents such as ordering food, eating, and paying the bill and actors such as Roger, the waitress, and the food, with restrictions on their arrangements and properties. Both events and tangible objects, then, can have a structure slot. An entry on a structure slot is a Structure, which, in turn, lists the parts/subEvents and the constraints on them. Several entries might be on a structure slot corresponding to orthogonal decompositions. It is significant that the constraints themselves, in both the action and tangible object cases, are often similar. For example, a set of subEvents or physical parts might be linearly sequenced or partially ordered (in time or space, respectively), nested in series, and so on.

Given a physical object and its parts, it is often possible to distinguish between different classes of parts. For example, the parts of most tables can be classified into parts meant for providing support to the top, the top itself, parts for decoration, and so on. We usually associate a predicate (which is an instance of PartSlot and a specSlot of physicalParts) with each of these classes and use these predicates to relate the parts to the overall object (rather than just using the single predicate physicalParts.)

A similar approach is taken to relating the parts of an action to the action. When dealing with actions, there are two important categories of parts—the actors and the subEvents—and there are separate categories of slots that are used to relate the actors to an action (the ActorSlots) and the subEvents to the event (the SubEventSlots). The ActorSlots define the roles played by the different actors in the event (performer, victim, instrument, and so on). Given an action and a participant actor, there are three subAbstractions of the actor related to this action, namely, the subAbstraction of the actor just before, during, and after the action. In practice, we associate the entities of the actors with the action (through the ActorSlots) and then use three ternary predicates (subAbsOfActorBefore, subAbsOfActorDuring, and subAbsOfActorAfter) to specify the exact subAbstractions of the actors.

It should be noted that there are no primitive actions into which all actions are broken down. That is, the actions are not merely macros introduced for notational convenience (for use instead of more complex sequences of primitive actions). We eschew primitive actions because we want to be able to reason at different levels of abstraction, and the a priori assigning of a set of actions as primitives goes against this ability; often, one can provide only descriptions and not definitions of the more complex actions in terms of their subEvents. In such cases, the more complex actions are not merely for notational convenience but are an epistemological necessity.

One of the problems that arises with predicting the effects of actions on the participating actors is the possibility of concurrent events. Our solution for this problem is as follows: Cyc gathers all the co-temporal events E_i (cutting up events if required) that affect a particular property and collects them into a single new event E. Cyc then computes the net effect of E on this property from the subEvents of E (suppressing the direct updating of the property by the subEvents). It should be noted that the agglomeration is necessarily a nonmonotonic process because a closed-world assumption has to be made while collecting the E_i.

Another thorny issue is how to handle, for example, the weight of (the subAbstraction) of FredSmith on a particular day because it obviously fluctuates slightly from moment to moment. More generally, when dealing with subAbstractions of a reasonable duration, it becomes hard to specify values for most temporally intrinsic numeric attributes because of the (often slight) changes in the value of the attribute over the period of the subAbstraction. To overcome this problem, we use a class of terms corresponding to intervals in the quantity space (of the attribute). These intervals can be named (for example, around 180 pounds) and in the extreme can also be open in one direction. A calculus for performing simple mathematical operations with these intervals (provided by CycL) makes it relatively easy to use both qualitative and quantitative specifications for attributes, switch between them, and so on. Another use for these interval-based quantity terms is to specify defaults for numeric attributes (for example, height, weight) for categories that exhibit some but not too much variation in the value of these attributes (for example, the weight of newborn humans).

Interval-based quantity slots are also useful for dealing with quantities for which no acceptable measurable scale or measuring instruments have yet (or perhaps ever will) to be defined: happiness, alertness, level of frustration, attractiveness, and so on. Despite the lack of absolute units of measure, mileposts for these attribute values can be defined, and partial orders and even crude calculi can be developed.

Composite Objects and Agents

In addition to purely physical objects (such as tables and rocks), objects such as books and people exist with which we would like to associate an intangible aspect such as a message or a mind (which also would have a temporal aspect). Given such a composite tangible-intangible object, we can separate the purely tangible parts from the purely intangible and separately represent both of them explicitly as well as represent the composite. The purely intangible parts are instances of IntangibleObject, the purely physical parts are instances of TangibleObject, and the composition is an instance of CompositeTangibleIntangibleObject.

The most important subset of CompositeTangibleIntangibleObject is Agent—the set of intelligent agents—and this subsection considers some aspects of representing agents. Agents include people (and most macroscopic animals), computer programs, corporations, and other things with intelligence, purpose, and (frequently) rationality.[12] Let us first consider why we want this distinction between the physical and nonphysical aspects of agents. Consider representing the Frankenstein monster at some point in time. We would like to be able to say that his body was *n* years old, his mind was *m* years old, and the composition was *k* years old. Rather than introduce new predicates such as ageOfMyMind, ageOfMyBody, amountOfTimeSinceMyMindAndBodyWereJoined, and so on, we would like to use the single existing predicate age; besides being simpler and cleaner, this predicate also lets us fully use the axioms involving age that are already available.

To deal with the previous example, we need to be able to explicitly talk about the physical and mental extents of a composite. Having made this distinction (and relating these using the predicates physicalExtent and mentalExtent), we associate weight not with the mental part of the Frankenstein monster nor with the composite part but only with the physical part; similarly, IQ is associated only with the mental part. Finally, age makes sense for all three aspects and has a different value for all three.

This scheme has the advantage of separating the physical aspects of composites from their mental aspects and allows us to talk about aspects that might apply to both with different values (age, interestingness, likedBy, and so on). However, in most cases, there is not a single predicate that can be used and has a different value for two or more of these units (physical extent, mental extent, composite). We would like to make use of this regularity.

In other words, we don't mind having three separate concepts for Frankenstein's monster—he was rather unusual after all—but we shouldn't need to have three separate concepts for every composite if nothing conflicts from one to the other. We added the categories PartiallyTangible (a spec of SomethingExisting and a genl of TangibleObject) and PartiallyIntangible (a spec of SomethingExisting and a genl of IntangibleObject); so now, CompositeTangibleIntangibleObject is a spec of both of these new Partially . . . collections. IQ, for example, now makes sense for PartiallyIntangibleObjects, and weight makes sense for PartiallyTangibleObjects, and so on. Thus, we can state IQ, weight, and all the other properties right on the composite unit.

If we happen to be representing an exception, such as the monster, in which some property has a different value for the physical or mental extent, then we can create the appropriate instances of TangibleObject and IntangibleObject, just as we did previously. As a default, we also inherit the properties that talk about physical (mental) properties to the physical (mental) extents. This framework gives both the expressiveness of the separation of physical and mental parts and the efficiency of not doing this separation unless it is required.

A full description of the various issues related to agenthood that have been and are being considered in Cyc would require more space than is available here, so we just mention a few of them. One of our recent technical reports (Guha and Lenat 1989b) deals exclusively with this topic.

Agents can be collective (such as organizations and institutions) or individual (such as people). Each agent can have one or more attitudes toward any given proposition; some of the propositional attitudes currently used in Cyc include beliefs, goals, dreads, purpose, and desires. We now consider some issues related to these propositional attitudes.

A primitive notion of awareness is incorporated as follows: Each agent has a set of terms and predicates s/he is awareOf. An agent cannot have any attitude toward a sentence that involves terms s/he is not aware of. This restriction is introduced to keep us from, for example, talking about Aristotle's beliefs about the Space Shuttle.

Attributing our own beliefs to other agents (with whom we might never have directly communicated) is something frequently done. Sometimes this attributing is good (for example, when the traffic light in front of you turns green, you assume that the drivers on the cross-street share your beliefs about what this green light means), and sometimes

We eschew primitive actions because we want to be able to reason at different levels of abstraction . . .

it's bad (for example, cross-cultural mirror imaging has led to innumerable political disasters). A class of axioms called the *belief projection axioms* enables Cyc to efficiently do this sort of mirror imaging and explicitly record separate beliefs when they are known. These belief projection rules themselves are moderately interesting because they describe what it means to be a public figure, what it means to be commonsense knowledge, and so on. CycL provides special support to efficiently handle these rules at the heuristic level.

Agents can be in control of (the truth value of) propositions, which means that the controlling agent can perform the requisite actions that determine the proposition's truth value. For example, a robber holding a gun is in control of whether the gun fires and at whom. The truth value chosen by the controlling agent is assumed to be based on his/her/its desires.

This notion of agents controlling propositions is sometimes an expedient way of computing the truth value of certain propositions. If an agent is in control of a proposition P, and s/he desires P, then we can assume that P is true (modulo limited resources, conflicting goals, and so on).

The concept of control gives us an abstraction layer that allows us to skip the details of the agent planning to make P true, executing this plan, monitoring it, repairing it, and so on. For example, a teacher assigns a book report to a student and, later, needing a copy of this book, asks the student where it can be obtained. The teacher didn't have to worry about the plan the student made to get the book or read it, the details of the execution of this plan, and so on, to believe that (by the day before the book report was due) the student would have somehow obtained a copy of the book.

Just knowing that you control when you go home from work and that you want to sleep at home tonight is enough to have me call you first at home at midnight if I have to reach you at this time; that is, I don't have to worry about the plan you made to get home, the details of the execution, and so on, to believe that (by midnight, at least) you had made it home.

Agents can participate in events (actually in instances of SomethingOccurring (see the section Processes, Events, and Persistent Objects and also figure 2) in one of two modes: voluntarily or involuntarily. If an agent A participates in an event E voluntarily, s/he usually has a purpose P; P is generally one of his(her) goals, which, in turn, are the subset of his(her) desires that s/he has decided to try and achieve. Moreover, A believes that P will become true because of this event E. The concept of purpose allows us to (write axioms that will) decide when an agent will participate in (or pull out of) an event.

Agents can enter into Agreements with other agents; some of the parties to an agreement might be individual agents, and some might be collective. This activity is one of the most important, frequent ones that agents carry out, and we use the remainder of this subsection to discuss it.

An *agreement* defines a set of propositions that all the participants share (although they might have different propositional attitudes toward the various clauses of the agreement!). In addition, the Agreement might also assign certain responsibilities (logically, these are also propositions) to specific participants. Agreements also usually specify certain punitive or remedial actions to be taken in case these responsibilities are not fulfilled. If the agent performing these punitive actions is a LegalSystem (such as a Government or GovernmentalAgency), then the agreement is said to be a LegalAgreement.

We distinguish between agreements in which the event that enrolled a particular agent was one in which s/he voluntarily participated and ones in which s/he didn't participate voluntarily. For agreements an agent involuntarily participates in, the constraint that s/he shares the common beliefs of the agreement is slightly relaxed.

As a default, collective agents have one or more special types of agreements associated with them, such as their charter and articles of incorporation. Often, an organization or institution will itself have (or at least act as if it has) certain desires, dreads, purposefulness, authority, and so on, that are not present in most (or perhaps even any) of the participants or members of the organization. Often, such attitudes and goals are captured in, or even caused by, the special agreements it participates in.

Microtheories and the Ontology

The formalisms discussed in the previous few sections try to provide a general and expressive framework for representing various aspects of the world. However, as our initial statements claimed, much of the power in Cyc comes from specific partial solutions, simplified models (of time, space, a domain, and so on) that work in certain common but restricted situations and not from overarching use-neutral formalisms.

The use of single abstraction, no matter how expressive, also has bad consequences from a pragmatic standpoint. For example, one would like to be able to state something like "When x buys o from y, one of the subEvents is a paying event, in which x pays y the price of o." Given an expressive abstraction of time, we might have to write a long axiom to encode this statement—one that involves numerous uses of subAbsOfActorDuring, subAbsOfActorAfter, and so on—and most of the details being represented would be irrelevant in 99 percent of the uses of the axiom.

Thus, we would like to use much simpler abstractions if the domain we are dealing with does not require more expressive abstractions. Further, the process of formulating models for domains becomes time consuming if one is going to try and develop the *right* model for each domain.

These considerations led us to introduce the concepts of microtheories and contexts (McCarthy 1987b; Guha 1990a) that we mentioned earlier. These microtheories are first-class objects, and a number of operations are possible on them. For example, given a discourse or a problem to be solved, we can decide to create and enter a new context that leaves implicit a number of arguments to predicates, makes assumptions about the domain, and so on. We can have axioms that describe which kind of microtheory should be used for which problems, which form taxonomies or other structures of microtheories, which decide when it is necessary to change the context of a microtheory (see The Evolution of the Representation Language for a discussion on the relationship between a context and a microtheory), which have defaults that lift the contents of one microtheory into a more general one, and so on.

The ontology described in earlier sections is the most detailed one and is representative of the net-expressive power of the language. In addition to the formalism for time and actions previously mentioned, we have a lattice of simpler abstractions of these available. Some of these abstractions have only discrete time or associate time with propositions, and some even assume that the whole theory they are dealing with is about a single instant of the world and, hence, completely ignore time. A particular domain theory about, say, Buying (remember that many different theories can exist for any given domain), can use the simplest abstraction of time, actions, and so on, that are adequate for this theory.

The concept of microtheories figures prominently in the approach we are taking for developing a representation for space. After many failed attempts at developing a single general abstraction for space, we decided to use a number of globally inadequate but locally adequate theories of space. For example, we are working on abstractions of space that are based on (1) simple diagram-like representations, (2) computer-aided design–like representations that build solids and surfaces out of a small number of primitives, and (3) device-level representations that primarily deal with the topology of a device by using a number of primitive components and using a small number of ports for each primitive and a small number of ways in which two primitives can be connected. Although none of these abstractions is sufficient as a general approach to representing space, for any given problem, one of these (plus a few more that we are developing) if often adequate. These various abstractions of space are organized into a hierarchy because some are just refinements of others.

The concept of microtheories figures prominently in the approach we are taking for developing a representation for space.

Contexts and microtheories are not panaceas for the problems mentioned at the beginning of this subsection. They bring their own set of problems with them. For example, we need to determine when to use which context, when a context is insufficient, when we need to enter a new context, and so on. Although it does not seem hard to come up with extremely specific heuristics that do these tasks (for example, for kinematics problems, if the velocity of the objects is less than .5c, then use Newtonian mechanics), it would be preferable to have at least a few general heuristics for these purposes (Guha 1990a). This area is one of current focus. Most of the knowledge base is currently in the theory MEM. The simpler abstractions of time mentioned earlier are available, and a few theories use these abstractions. A few general heuristics are available for solving some of the problems (such as deciding when to use which theory), and more are being developed. We expect that most of the theories of topics at a finer granule than those mentioned in this section, which we will add to Cyc, will make use of this notion of microtheories and that they will figure prominently in the overall structure and contents of the knowledge base.

The Evolution of the Cyc Ontology

Earlier, we described the 1984–1990 evolution of the Cyc methodology and the CycL representation language. The construction of the ontology began in late 1984 with an expert system–like approach. As is apparent from Lenat, Prakash, and Shepherd (1986), issues such as the representation of time, substances, agents, and causality were largely ignored in favor of highly specific issues. However, our position has considerably changed in the interim, and these issues are now regarded as significant to the ontology.

We have worked out fairly adequate solutions to some of these ontological problems and are working on solutions for others. We differ from the rest of the AI community working on such issues in that we have refrained from agonizing over the myriad subtleties of, for example, the different formalisms for representing pieces of time; instead, we have concentrated on using the formalisms we have for actually encoding information about various domains. We have also refrained from tinkering with the logic for, for example, reasoning about pieces of time; instead we have worked within the logic described earlier (see The Evolution of the Representation Language).

In general, we did not find these solutions on the first try or even the second. Therefore, we conclude this subsection with a brief list of some of the mistakes we made and the lessons we learned along the way.[13]

We began by explicitly representing the time interval over which an event occurred. Thus, for example, we would have an event such as TrafficAccident903 and a separate object TimeIntervalOfTrafficAccident903, and only the time intervals could participate in temporal relations such as before and startsAfterTheEndOf. However, again and again, there was never anything more to say about these time interval frames, that is, beyond a pointer to the event that they represented the time interval and beyond the temporal relations in which they participated. Thus, eventually we just merged the two notions—an event and the time interval of its occurrence—and now the domain and range of temporal relations is Event (the set of all events) rather than TimeInterval. In theory, TimeInterval need no longer exist; pragmatically, a few pure time intervals do exist in the sense of there being some events whose only interesting aspects are their temporal extents, for example, TheYear1990.

By examining many cases of reasoning about the creation and destruction of objects, we were led to classify TangibleObject as a subset of Event. That is, each tangible object is running a process so to speak—the process of existing. This ontology lets us easily state temporal relations between objects, such as "Fred lived long after Aristotle." We originally viewed the Substance versus IndividualObject distinction to be crucial. We were surprised to find the two concepts *coextensional* (each portion of a substance is an individual and vice versa); similarly, TangibleSubstance is coextensional with TangibleObject; and so on. The central distinction turns out to be between, for example, TangibleSubstanceType (whose instances include, for example, Wood and Water) and TangibleObjectType (whose instances include, for example, Table and Ocean). Using this latter distinction, Cyc can, for example, predict whether a property will be extrinsic or intrinsic.

During late 1986 and early 1987, we thrashed through a series of formulations for basic property inheritance, including explicitly having frames such as TypicalPerson instead of (and at one point in addition to) TheSetOfAllPeople. We can use TypicalPerson to illustrate two troubles with prototypes:

First, one often wants to inherit along slots other than allInstances. For example, each entry on Stallone's fans slot should inherit entries such as Rocky and Cobra to its moviesSeen slot. (Of course, our use of terms such as slot and entries is just shorthand. In this case, the general nonmonotonic assertion at the epistemological level is $\text{fans}(\text{Stallone}, x) \wedge \neg ab_n(x) \supset \text{moviesSeen}(x, \text{Rocky})$.)

Second, we occasionally need to distinguish a predicate (such as interestingness, createdBy, likedBy) applying to the set of all people, being inherited to the typical instance of this set, or even applying (at a metalevel) to the reified Cyc concept itself. For example, the typical person is created by their parents, the frame TypicalPerson was created by Lenat, and the set of all people has no creator (all sets exist in a Platonic universe and can be neither created nor destroyed). The typical person is interesting, the concept TypicalPerson is uninteresting, the set of all people is reasonably interesting, the concept Person representing the set of all people is slightly interesting, and so on. Eventually, we dispensed with concepts such as TypicalPerson because any assertion applying to it could just as well be stated as applying to (that is, inheriting to) allInstances of Person.

Precision is generally a good thing, or (to be more precise) the ability to be precise is generally a good thing. However, in most situations, the most cost-effective behavior is to choose a level of detail and use the corresponding microtheory instead. A case in point was our partitioning of the universe (Thing) into TangibleObject (such as Rock0093), Intangible (such as the predicate mass or the set Rock of all rocks), and CompositeTangible/IntangibleObject (such as Lenat). As we discussed in detail in the sections on composite objects and agents, Cyc frequently gets the right answer without making this distinction, even though in rare cases, it is important (for example, the typical person has the same answer for how old is their mind, their body, and the joining of the two, but we can, as in Frankenstein, separate the three concepts if we need to). In these subsections, we explained our solution (that is, our current way out of this dilemma), which involves the use of PartiallyTangible and PartiallyIntangible, and we also described our current investigations into codifying and using microtheories.

The Current State of Cyc's Environment and Infrastructure

You might have noticed several aspects of the Cyc effort that we did not even touch on in this article. Although interesting in their own right, these areas are not our main topic for research, and in each case, we have done only what we felt necessary to maximize the rate of Cyc's construction. Here is a list of a few such intentional omissions:14

The Knowledge Server: This subsystem accepts everyone's knowledge base operations; serializes them; and, in case constraint violations appear, adjudicates the resolution of the conflict. In cases of no conflict, it then broadcasts the operations to everyone else. The connections today are generally thin wire, although we expect this to change in the coming year.

The User Interface: We have constructed various textual and graphic tools for browsing, querying, and editing the knowledge base. Some of the graphic tools are semantic net based; one is a sort of Escheresque recursive bird's-eye view of a museum floor plan. Some of the editing tools are ideal for making point mutations and corrections; some are oriented toward sketching some brand new (and not yet well codified) area and gradually making the sketch more precise. User interface issues are tangled with knowledge acquisition (the next two items respectively describe tools for doing knowledge acquisition manually and automatically).

The Copy and Edit Mechanism: Most knowledge entry in Cyc involves finding similar knowledge and copying it and modifying the copy. More and more over the years, Cyc has helped in this process, and as a result, knowledge entry can be done more rapidly than we had originally estimated.

Digitized Images: Yes, often it's much easier to just grab a picture of an object and point to the part you mean rather than try to figure out what its name is. Cyc contains such images (from *The Visual Dictionary* [Corbell 1987]), but experienced knowledge enterers know their way around the knowledge base and rarely use them.

Other Nonpropositional Knowledge: Some Cyc researchers are building neural nets that we can use at the earliest (preheuristic) and latest (reduction to instinct) stages of understanding some task (for example, training a net on examples of good analogies and then letting it make hunches about potentially good new analogies, hunches that the rest of Cyc can investigate and flesh out symbolically).

The Machine Learning Module: This module is a symbolic learning subsystem in addition to the previous statistical, nonpropositional one. This program roams over the knowledge base, typically at night, looking for unexpected symmetries and asymmetries. These, in turn, often turn out to be bugs, usually crimes of omission of one sort or another. In rare cases today and, we expect, more frequently in future years, these turn out to be genuine little discoveries of useful but hitherto unentered knowledge.

Present and Future Directions

This article began by explaining the need for a large, general knowledge base to overcome the brittleness (in the face of unanticipated situations) that limits software today. The need for a Cyc-like knowledge base is critical not only in expert system–like applications but also to do semantic processing for natural language understanding (disambiguation, anaphora, ellipsis) and to enable realistic machine learning and analogizing.

We then focused on criteria for an adequate representation language for this knowledge base, which drove us to the bifurcated CycL architecture, having both an expressive epistemological level and an efficient heuristic level. One of the Cyc project's most interesting

Most knowledge entry in Cyc involves finding similar knowledge and copying it and modifying the copy.

We plan to develop an ontology of arguments and formulate and encode assorted perference criteria that are useful in default reasoning . . .

accomplishments has been the construction of the TA translator, which can convert back and forth between general epistemological-level (FOPC-like) expressions and particular heuristic-level template instances. The epistemological and heuristic knowledge base levels and the TA translator are fully implemented and function as described in this article.

We discussed some of the unexpected aspects of the Cyc knowledge base's organization and contents, such as the relationships between IndividualObject, Substance, Process, and Event. We gave the flavor of some of our recent research by sketching our (still incomplete) treatment of agents and agreements. Finally, we briefly mentioned the Cyc user interface tools, knowledge server, and related issues.

Perhaps the most important theme from all these aspects of the project is that of eschewing the single general solution dream, instead assembling a set of partial solutions that work most of the time and work efficiently in the most common situations. We have seen this tenet apply to representation language design, knowledge entry methodology, control of search during inferencing, and truth maintenance as well as throughout the contents of the knowledge base. The emerging global behavior of the system should hopefully be fairly use neutral.

In the following two subsections, we discuss the directions along which we are currently working to extend both CycL and the knowledge base. In some sense, the two are extremely intertwined, in that the extensions to CycL are largely driven by the needs of the knowledge base. We then touch on other future plans and issues: new collaborations; the proliferation, standardization, and marketing of Cyc; and the expected future scenario for the demise (or at least transmutation) of the Cyc project.

Future Plans for the Representation Language

The main focus of research on extending the reasoning abilities of the CycL language is along the following lines:

Default Reasoning: We plan to develop an ontology of arguments and to formulate and encode assorted preference criteria that are useful in default reasoning (to aid in choosing one argument over another).

Microtheories: Although the epistemological level does not require any extension to deal with these explicit contexts, it is likely that certain standard kinds of microtheories are likely to benefit from the heuristic level's providing special support for this feature.

Metalevel: Although there are various facilities for stating and using metalevel information in Cyc, the metalevel reasoning that takes place in Cyc today is minimal because it has not yet been the bottleneck in building and running Cyc. We believe this situation might change as Cyc grows even larger, and the search for arguments grows. Metalevel reasoning in Cyc needs to be extended both by adding reasonable metalevel heuristics (for example, for identifying the small subset of the knowledge base that is even possibly relevant to answering a given query [Guha and Levy 1990]) and providing some ability for the system to improve its performance from past experience.

Future Plans for the Ontology and the Knowledge Base

The work being done on the Cyc knowledge base is largely in the form of the development of microtheories for topics at the level of transportation, human emotions, modern buildings, and so on. Microtheories are being developed in two major areas. The first of these is a set of commonsense theories about everyday physical phenomena (in the manner of naive physics), and the second of these is a set of theories dealing with aspects of agents (such as agents having and acting on the basis of emotions, goals, limited resources).

In addition to these areas, a significant research effort is in progress on the topic of microtheories themselves, from the perspective of both formulating axioms for solving problems (such as when which theory is to be used) and cataloging the standard kinds of simplifications that can be made to theories on developing formalisms, such as for time and space, that make it easier to state axioms (that use these formalisms).

Finally, although gross size is often negatively correlated with overall knowledge base quality and power, we should mention that Cyc's knowledge base currently contains over a million assertions, and we expect almost an order of magnitude increase in the coming five years. There are currently 30,000 units, of which approximately 4,500 are types of predicates, and approximately 5,000 are collections.

Collaborations

How are we to judge where we are going? How do we make sure that we don't go through these 10 years of labor and then find out in 1995 that we were fundamentally mistaken all along? We make sure by getting others to actually use our system.

Microtheories are being developed in two major areas. . . a set of common sense theories about everyday physical phenomena . . and . . . a set of theories dealing with aspects of agents . . .

In the past 18 months, as the Cyc representation language and ontology stabilized, we began to encourage collaborations with both academic researchers and industrial researchers and developers. We also held workshops and panel sessions. Finally, we once again (after a purposeful several-year hiatus to focus solely on research) began to write books and technical reports and articles, such as this one, to inform and interest our colleagues.

Cyc is still too small to have more than an anecdotal chance of improving the performance of the application programs using it, but the early results are promising. Cyc-based applications and research efforts are under way at Bellcore, Digital Equipment Corporation, NCR, US West, and Apple, and these efforts do not include the numerous Cyc-based research projects in academia. In the following paragraphs, we sketch a few of these efforts plus a few that are just beginning. Our purpose is to illustrate projects that are using the richness, scope, or size of the Cyc knowledge base.

At Digital, John McDermott, David Marques, Renata Bushko, and others have built a Cyc-based computer-sizing application. Serving as a preprocessing step for Xcon (Soloway, Bachant, and Jensen 1987), its job is to ask questions about a potential Digital customer and come up with a rough computer sizing (mainframes versus workstations, approximate number and size of file servers, and so on). The trouble with standard expert systems doing this task is that they ask too many questions, questions that can often be answered by common sense, questions that Cyc is able to guess answers for. (We don't mean that standard expert systems lack inferential abilities, just that they simply don't have the required commonsense knowledge from which to draw the inferences.)

For example, given that toy manufacturers have stringent government safety regulations and adult clothing manufacturers don't, which is more likely to be the proper match or precedent for a new potential customer who is a manufacturer of children's clothing? As another example, given that the basic business unit in a hotel is the room and at a car rental agency is the car, use relatively deep understanding of what goes on at each place to decide that for a hospital, which is a new potential customer, the right business unit is Bed, not Room. These examples illustrate a reliance on the scope of the Cyc knowledge base. The next example illustrates a reliance on the richness and size of one particular part of the knowledge base, namely, the part dealing with predicates (and, in particular, slots.)

One speculative but fascinating line of research by Paul Cohen, Micheal Huhns, and others (Cohen and Loiselle 1988; Huhns and Stephens 1988) involves automatically guessing whether the composition of two unary functions (binary predicates, slots) s1 o s2 is equal to any known relation s3. The guess is made as follows: They assign +, -, and 0 values to 10 attributes of each predicate (such as hierarchical, composable, temporal, near, intrinsic). They then create a combination table for each of these attributes, which says, for example, "if s1 has a + value for its hierarchical attribute, and s2 has a - value, then s3 must have a + value." Given a particular s1 and s2, their values for the 10 attributes produce a set of 10 constraints on the values of these attributes for s3, which is used to constrain the search for s3's identity.

To the extent that this line of research succeeds, it teases apart the underlying deep structure of what it means to be a predicate. However, when or if the scheme fails, such failures should lead to the refining of the combination tables and assignments or, rarely, to the defining of a new primitive attribute for predicates. In any case, the point is that the research required a platform like Cyc: (1) The size of Cyc's slot space (4000+ known types of slots and other predicates) increases the chance that the resultant 10 vector matches some existing predicate. (2) The richness of slot space (categories and other interslot relationships) helps in the following way: For each particular slot s_i, most of the 10 primitive attributes' values for s_i can

We tell Cyc that Sam is a land animal and ask for types of body parts that Sam probably has:

(assert '(#%allInstanceOf #%Sam #%LandAnimal))
(ask '(#%hasPartTypes #%Sam x)) .

The question is addressed to the epistemological level. The tell-ask interface translates this question into an appropriate heuristic-level query. The heuristic level already contains an inheritance axiom that says the land animals have legs. The inheritance procedure runs and yields a simple argument that Sam has legs. Because there is no reason for invalidating this argument and because there isn't any argument for Sam not having legs, the Argument Comparison module asserts into the heuristic-level knowledge base that Sam has legs. In turn, this assertion produces a binding list for the free variable x in the formula that we Asked, above; namely,

((x . Legs)) .

Repeated calls to Ask can be made and will result in additional bindings for *x*. We now tell the systems that as a default, Reptiles don't have legs. The tell-ask interface translates this assertion into an inheritance axiom:

(assert '(#%LogImplication (#%LogAnd (#%allInstanceOf x #%Reptile) (#%LogNot (#%abnormal x "reptile-with-leg")))
(#%LogNot (#%hasPartTypes x #%Legs)))) .

We now tell Cyc that Sam is a snake, and (in case it didn't know already) that snakes are reptiles, and again ask the system if Sam has legs.

(assert '(#%allGenls #%Snake #%Reptile))
(assert '(#%allInstanceOf #%Sam #%Snake))
(ask '(#%hasPartTypes #%Sam #%Legs)) .

Even though the system previously asserted that Sam had legs, the addition of the inheritance caused this conclusion to be marked as potentially stale. Thus, when we ask this query, the argument generator tries to search for an argument for the negation of the query and, sure enough, finds one. Now we have an argument for Sam having legs based on his being a land animal and an argument for Sam not having legs based on his being a reptile. The argument comparator chooses the more specific one and concludes (and asserts into the heuristic-level knowledge base) that Sam does not have legs.

Figure 3. An Example of Interaction between the User, the Epistemological Level, the Tell-Ask Interface, and the Heuristic Level.
The example is intentionally very simple and is meant to be clear rather than typical of the sophisticated inferencing that Cyc performs.

be inherited or otherwise inferred rather than have to go through (in our current case) 40,000+ separate episodes of pondering and entering a +, -, or 0 entry.

One vital collaboration is that with Elaine Rich and Jim Barnett at MCC, namely, the natural language understanding project described in Barnett et al. (1990). Here, the Cyc knowledge base is stressed to its limits to disambiguate word senses, resolve anaphoric and omitted references, interpret the meaning of ellipses, and so on. Only so much can be done with syntactic analysis and discourse rules; the rest must be handed to Cyc, which creates various alternate hypothetical worlds and decides how (im)plausible each one is, for example, by seeing how metonymical the speaker would have to be in each possible world for no real-world constraints to be violated.

A number of theories in AI seem rather powerful—and might be—but have not been tested on many examples. Having a knowledge base with a wide scope (not just size) on which to test these theories would be useful. One example is the work of Devika Subramanian, now at Cornell University, who is testing her reformulation ideas in Cyc.

Other ongoing research collaborations include coupling with large engineering knowledge bases (with Ed Feigenbaum and Tom Gruber at Stanford University) and large databases (with Stuart Russell and Mike Stonebraker at University of California at Berkeley), standardizing knowledge interchange formats (with Mike Genesereth at Stanford), axiomatizing human emotions (with John McCarthy at Stanford), developing machine learning (with Wei-Min Shen and Mark Derthick at MCC), and using qualitative physics and analogy in children's stories (with Ken Forbus). We are also engaged in informal collaborations with Marvin Minsky, Pat Hayes, and others.

Conclusion

What do we hope to get from our efforts? Where do we hope to be by the end of the 1990s? Here are three possible levels of success, in increasing order of optimism:

Good: Although not directly built on and widely used, the Cyc research might still provide some insights into issues involved in building large commonsense knowledge bases. Perhaps, it would give us an indication about whether the symbolic paradigm is flawed and, if so, how. It might also yield a rich repertoire of "how to represent it" heuristics and might at least motivate research issues in building future AI systems.

Better: Cyc forms the core of a knowledge base that can be used by the next generation of AI research programs to help make them more than theoretical exercises. No one doing research in symbolic AI in the early twenty-first century would want to be without a copy of Cyc, any more than today's researchers would want to be without Eval.

Best: Cyc's knowledge base serves as the foundation for the first true AI agent. It is something on which full-fledged natural language understanding systems, expert systems, and machine learning systems can be built. No one in 2015 would dream of buying a machine without common sense, any more than anyone today would buy a personal computer that couldn't run spreadsheets, word processing programs, communications software, and so on.

Numerous projects are under way in Japan and Great Britain and in this country at Carnegie-Mellon University, USC/Information Sciences Institute, Stanford, and elsewhere that appear similar to Cyc in some ways, but each of them has traded breadth and scope for an increased chance of success at this narrowed goal (for example, several of these projects have some form of limited machine translation as their goal).

Do we have a good chance of achieving our better goal? Because of our clipping of representation thorns and stabilizing of the representation language, inference engine suite, and high-level ontology, we believe the answer is "Yes." At the least, we now expect that the body of knowledge we accumulate in this decade will continue to exist, be used, and be extended for decades to come, with much of this accretion being done semiautomatically.

We now come to the final area we address in this article: the Cyc project's expected life expectancy and mode of demise. We expect to increasingly rely on knowledge-based natural language understanding (KBNL) as the means of clarifying apparent contradictions and gaps in the knowledge base, and we expect to rely increasingly on knowledge-based machine learning (KBML) as the means of identifying these unexpected regularities and irregularities in the knowledge base. Both KBNL and KBML can gradually become more and more important ways of entering new knowledge into the knowledge base. These projects have already begun, but it's too early to look for signs of the ramp up. This particular corner is one we expect to turn in the middle part of this decade, with KBNL becoming the dominant mode of knowledge acquisition for Cyc and KBML becoming the dominant mode of policing the growing knowledge base. The role of human knowledge enterers today, manually entering assertion after assertion, is akin to teachers who must instruct by surgically manipulating brains. Turning the corner would mean that the role of humans would become much more like tutors, with a great deal of question answering going on in both directions. Much of the work in this coming decade will be driven by use as humans and application programs exercise Cyc and rely on it more and more as a substrate for intelligent behavior.

. . . the role of humans would become much more like tutors . . .

Acknowledgments

We would like to thank Mary Shepherd for her patience and help since the inception of the Cyc project. We also want to thank Mark Derthick, Karen Pittman, Dexter Pratt, David Wallace, Nick Siegel, Wanda Pratt, John Huffman, and the other members of the Cyc group, past and present, for their help in building and testing the system. We also thank Ed Feigenbaum, Pat Hayes, John McCarthy, John McDermott, Marvin Minsky, and Stuart Russell, who have in almost orthogonal ways helped us to think more clearly about the material presented herein. We also thank John McCarthy, Charles Petrie, and Russ Greiner for reading earlier drafts of this article and giving us numerous useful comments.

References

Allen, J. 1986. Maintaining Knowledge about Temporal Intervals. In *Readings in Knowledge Representation*, eds. H. Levesque and R. Brachman, 509–523. San Mateo, Calif.: Morgan Kaufmann.

Barnett, J.; Knight, K.; Mani, I.; and Rich, E. 1990. Knowledge and Natural Language Processing, Technical Report ACT-NL-104-90, MCC Corp., Austin, Tex.

Brachman, R. J.; Fikes, R. E.; and Levesque, H. J. 1986. Krypton: A Functional Approach to Knowledge Representation. In *Readings in Knowledge Representation,* eds. H. Levesque and R. Brachman, 411–431. San Mateo, Calif.: Morgan Kaufmann.

Cohen, P. R., and Loiselle, C. L. 1988. Beyond isa: Structures for Plausible Inference in Semantic Networks. In Proceedings of the Seventh National Conference on Artificial Intelligence. Menlo Park, Calif.: American Association for Artificial Intelligence.

Corbell, J. C. 1987. *The Visual Dictionary.* New York: Facts on File.

Derthick, M. 1990. An Epistemological Level Interface for Cyc, Technical Report ACT-CYC-084-90, MCC Corp., Austin, Tex.

Doyle, J. 1987. A Truth Maintenance System. In *Readings In Nonmonotonic Reasoning*, ed. M. Ginsberg, 259–279. San Mateo, Calif.: Morgan Kaufmann.

Ginsberg, M. 1987. Multivalued Logics. In *Readings in Nonmonotonic Reasoning*, ed. M. Ginsberg, 251–256. San Mateo, Calif.: Morgan Kaufmann.

Guha, R. V. 1990a. Contexts in Cyc, Technical Report ACT-CYC-129-90, MCC Corp., Austin, Tex.

Guha, R. V. 1990b. The Representation of Defaults in Cyc, Technical Report ACT-CYC-083-90, MCC Corp., Austin, Tex.

Guha, R. V., and Lenat, D. B. 1989a. Cycl: The Cyc Representation Language, Part 3, Technical Report ACT-CYC-454-89, MCC Corp., Austin, Tex.

Guha, R. V., and Lenat, D. B. 1989b. The World according to Cyc, Part 2: Agents and Institutions, Technical Report ACT-CYC-453-89, MCC Corp., Austin, Tex.

Guha, R. V., and Levy, A. 1990. A Relevance-Based Meta Level, Technical Report ACT-CYC-040-90, MCC Corp., Austin, Tex.

Hayes, P. J. 1985. Naive Physics 1: Ontology for Fiquids. In *Formal Theories of the Common Sense World* (chapter 3), eds. J. R. Hobbs and R. C. Moore. Norwood, N.J.: Ablex.

Huhns, M., and Stephens, L. 1988. Extended Composition of Relations, Technical Report ACA-AI-376-88, MCC Corp., Austin, Tex.

Lenat, D. B., and Guha, R. V. 1990. *Building Large Knowledge-Based Systems*. Reading, Mass.: Addison-Wesley.

Lenat, D. B.; Prakash, M.; and Shepherd, M. 1986. Cyc: Using Common Sense Knowledge to Overcome Brittleness and Knowledge-Acquisition Bottlenecks. *AI Magazine* 6:65–85.

Lenat, D. B.; Guha, R. V.; Pittman, K.; Pratt, D.; and Shepherd, M. 1990. Cyc: Toward Programs with Common Sense. *Communications of the ACM* 33(8).

Lifschitz, V. 1987. Formal Theories of Action. In *Readings in Nonmonotonic Reasoning,* ed. M. Ginsberg, 410–432. San Mateo, Calif.: Morgan Kaufmann.

McCarthy, J. 1987a. Applications of Circumscription to Formalizing Common Sense Knowledge. In *Readings in Nonmonotonic Reasoning*, ed. M. Ginsberg, 153–166. San Mateo, Calif.: Morgan Kaufmann.

McCarthy, J. 1987b. Generality in Artificial Intelligence. *Communications of the ACM* 30(12): 1030–1035.

McCarthy, J. 1986a. Epistemological Problems of Artificial Intelligence. In *Readings in Knowledge Representation*, eds. H. Levesque and R. Brachman, 23–31. San Mateo, Calif.: Morgan Kaufmann.

McCarthy, J. 1986b. First-Order Theories of Individual Concepts and Propositions. In *Readings in Knowledge Representation*, eds. H. Levesque and R. Brachman, 523–531. San Mateo, Calif.: Morgan Kaufmann.

McCarthy, J., and Hayes, P. J. 1987. Some Philosophical Problems from the Standpoint of AI. In *Readings in Nonmonotonic Reasoning*, ed. M. Ginsberg, 26–45. San Mateo, Calif.: Morgan Kaufmann.

Moore, R. C. 1986. The Role of Logic in Knowledge Representation and Commonsense Reasoning. In *Readings in Knowledge Representation*, eds. H. Levesque and R. Brachman, 335–343. San Mateo, Calif.: Morgan Kaufmann.

Pratt, D., and Guha, R. V. 1990. A Functional Interface for Cyc, Technical Report ACT-CYC-089-90, MCC Corp., Austin, Tex.

Quine, W. V. 1969. Natural Kinds. In *Ontological Relativity and Other Essays* (chapter 5). New York: Columbia University Press.

Soloway, E.; Bachant, J.; and Jensen, K. 1987. Assessing the Maintainability of Xcon-in-Rime: Coping with the Problem of a Very Large Rule Base. In Proceedings of the Sixth National Conference on Artificial Intelligence, 9824–9829. Menlo Park, Calif.: American Association for Artificial Intelligence.

Touretsky, D. S. 1987. Implicit Ordering of Defaults in Inheritance Systems. In *Readings in Nonmonotonic Reasoning*, ed. M. Ginsberg, 106–109. San Mateo, Calif.: Morgan Kaufmann.

Warren, D. H. D. 1983. An Abstract Prolog Instruction Set, Technical Report 309, SRI, Artificial Intelligence Center, Computer Science and Technology Center, Menlo Park, Calif.

Weyhrauch, R. W. 1986. Prolegmena to a Theory of

Mechanized Formal Reasoning. In *Readings in Knowledge Representation*, eds. H. Levesque and R. Brachman, 309–329. San Mateo, Calif.: Morgan Kaufmann.

Notes

1. A shortened version of portions of this paper appeared in *Communications of the ACM,* Vol. 33, No. 8, Copyright 1990, Association for Computing Machinery, Inc. By permission.

2. All the material presented on the language and the ontology has been implemented and is in use in the Cyc system unless otherwise mentioned. In a number of places in this article, we have omitted the axioms and other details. They have been worked out and in most cases are available in one of the references or can be obtained from the authors.

3. Because it includes facilities to form the set of objects *x* satisfying a given property *P(x)*, our language is closer to Zermelo-Frankel (ZF) than to first-order predicate calculus (FOPC).

4. The isa predicate corresponds to the set-membership relation; it is sometimes called ISA, is-a, AKO, element-of, and so on. In Cyc's knowledge base, we happen to call it *instanceOf.* Also, the ab_i predicates are short for *abnormal in fashion i*; so $\mathrm{ab_i}$ corresponds to being an exception in the sense of being a bird and not being able to fly. One Cyc axiom says that birds with broken wings are abnormal in sense $\mathrm{ab_i}$.

5. *ist(x,y)* means *x* is true in theory *y*. Moreover, although supported, falls, and so on, are fluent valued functions, in this article, we often refer to them as predicates; the intended meaning should always be clear from the context.

6. Although in most cases, we use the term ontology in the sense of the objects that can be quantified over, we occasionally use it in a looser sense of the organization and structure of the knowledge base or even the predicates and objects in the epistemological-level language.

7. The same predicate can appear with different arities and argument types in different microtheories.

8. We used to, but they were never much use. Collections of collections, however, such as PersonType, SubstanceType, and EventType, have proven vital They are analogous to object-oriented systems' metaclasses.

9. The subabstraction formalism is superficially similar to the histories framework (Hayes 1985). However, there is no relation between the intersection of these histories and the frame problem (McCarthy and Hayes 1987).

10. The interval-based abstraction was developed independently of Allen (1986).

11. There are, of course, exceptions. For example, FredIn (1972) might like EthelIn (1958) more than EthelIn (1972).

12. This use of the term agent is unrelated to the linguists' usage, as in "'Kettle' is the agent in 'The kettle whistled'."

13. Rather than go through a lengthy presentation of some of the ways in which the Cyc ontology has changed over time, we refer the reader to the previous section, The Current State of Cyc's Ontology, wherein ~~the nature and salient aspects of~~ our fairly adequate solutions should become apparent.

14. Unless otherwise noted, each of them is fully implemented and in routine daily use.

R. V. Guha is the co-leader of the Cyc project and a Ph.D. student at Stanford University.

Douglas Lenat is principal scientist at Microelectronics and Computer Technology Corporation (MCC). He was a professor in the Computer Science Departments at Carnegie-Mellon University and Stanford University prior to his move to MCC and remains a consulting professor and industrial lecturer at Stanford today. His Ph.D. thesis (Stanford, 1976) was a demonstration that certain kinds of creative discoveries in mathematics could be produced by a computer program. This work earned him the biannual International Joint Conference on Artificial Intelligence, Inc., biannual Computers and Thought award in 1977. He was also named one of America's brightest scientists under the age of 40 by *Science Digest* in December 1984. Lenat has authored and coauthored more than 50 published papers and has written and edited several books, including *Knowledge-Based Systems in Artificial Intelligence*; *Building Expert Systems*; and, most recently, *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* He is an editor for *IEEE Transactions on Knowledge and Data Engineering* and the *Machine Learning Journal* and was a cofounder of Teknowledge, Inc. His main research interest is in getting machines to discover new knowledge, an interest that has led him to work on a large commonsense knowledge base.