Cyc: toward programs with common sense.

by Douglas B. Lenat, Ramanathan Guha, Karen Pittman, Dexter Pratt and Mary Shepherd

Cyc, a massive project to create a knowledge base spanning all human consensus knowledge, is discussed. The project will require the development of a new logic language for expressing knowledge before sets of procedures can be created and the knowledge base itself built. Cyc programmers developed CycL, a unique representation language and inference engine. Inferencing in Cyc at the heuristic level involves a variety of 'logically superfluous' mechanisms that make procedures more efficient, as well as highly specialized inference rules. The dependency analysis procedure used in the knowledge base is also described. Objects and properties can be either 'spatially' or 'temporally' intrinsic. Several application programs making use of Cyc are currently under development.

CYC: TOWARD PROGRAMS WITH COMMON SENSE

Motivation: The

Brittleness Bottleneck

For three decades, Artificial Intelligence researchers have grappled with issues like control of search in problem solving, organization of memory, logic for the representation of knowledge, perception, and so on, driven by tasks ranging from infants exploring their "worlds" through experts performing some intricate reasoning task. Despite many types of successes, today's AI software--and in many ways non-AI software as well--has reached a kind of bottleneck which is limiting its competence and usability. This article begins with a discussion of the nature of that bottleneck, and then describes the Cyc project: a serious attempt, begun in late 1984, to overcome this limitation.

The Path to Artificial

Intelligence

One of the principle tenets of AI, is that one can explicitly, declaratively represent a body of knowledge, in a way that renders it usable by a number of programs. This is in principle no different from the way in which a book "encodes" knowledge in tables and English text on the assumption that a wide audience will be able to use that data in myriad, possibly unanticipated ways. Since programs do not (yet) read and understand natural language, the encodings we use must be rather different, in particular much less ambiguous.

So achieving an AI comprises three tasks:

i) Develop a language (actually a logic) for expressing knowledge. Since we would like to allow many different programs to use this knowledge, this "representation language" needs a declarative semantics.

ii) Develop a set of procedures for manipulating (i.e., using) knowledge. Some of these will of necessity be heuristic, some will be strategic or meta-level, some will be aimed at truth maintenance and default reasoning, some will be inductive rather than deductive, and so on.

iii) Build the knowledge base(s). For example, encode knowledge in the language developed in i) above, so that one (person or machine) can apply to it the reasoning mechanisms of ii).

AI has largely concentrated on i) and ii). This is unfortunate, since it is task iii) that grounds the whole enterprise in reality. McCarthy [21] was the first to point out the importance of being able to represent knowledge in a program and initiated the task of devising representations for assorted topics such as time, agenthood, etc. Feigenbaum was one of the first to actually build programs which depended upon a significant amount of knowledge as their primary source of power. Later dubbed "expert systems," these programs showed that impressive levels of performance could be attained by taking iii)--knowledge--even half-seriously. For example, reasonable performance in narrow task domains may be achieved with modest-sized knowledge bases (KBs) (10 (2} to 10 (3) domain-specific assertions or rules.)

The Source of Software

Brittleness

There is indeed a strong local maximum of cost-effectiveness: by investing one or two person-years of effort, one can end up with a powerful expert system. The trouble is that this is just a local maximum. Knowing an infinitesimal fraction as much as the human expert, the program has only the veneer of intelligence. Let us illustrate what this means.

Cyc: toward programs with common sense.

Programs often use names of concepts such as predicates, variables, etc., that are meaningful to humans examining the code; however, only a shadow of that rich meaning is accessible to the program itself. For example, there might be some rules that conclude assertions of the form laysEggsInWater(x), and other rules triggered off of that predicate, but that is only a fragment of what a human can read into "laysEggsInWater." Suppose an expert system has the following four rules:

IF frog(x), THEN amphibian(x) IF amphibian(x), THEN laysEggsInWater(x) IF laysEggsInWater(x), THEN livesNearLotsOf(x, Water) IF livesNearLotsOf(x, Water), THEN ~livesInDesert(x)

Given the assertion frog(Freda), those rules could be used to conclude that various facts are true about Freda: amphibian(Freda), laysEggsInWater(Freda), ~livesInDessert(Freda),

etc. et the program would

not "know" how to answer questions like: Does Freda lay eggs? Is Freda sometimes in water?

Humans can draw not only those direct conclusions from laysEggsInWater(Freda), but can also answer slightly more complex queries which require a modicum of "outside" knowledge: Does Freda live on the sun? Was Freda born live or from an egg? Is Freda a person? Is Freda larger or smaller than a bacterium? Is Freda larger or smaller than the Pacific Ocean? Or even: How is Freda's egg-laying like Poe's story-writing?

Thus, much of the "I" in these "AI" programs is in the eye--and "I"--of the beholder. Carefully selecting just the fragments of relevant knowledge leads to adequate but brittle performance: when confronted by some unanticipated situation, the program is likely to reach the wrong conclusion. It is all too easy to find examples of such brittle behavior: a skin disease diagnosis system is told about a rusty old car, and concludes it has measles; a car loan authorization system approves a loan from someone whose "years at the same job" exceeds the applicant's age; a digitalis dosage system does not complain when someone accidentally types a patient's age and weight in reverse order (even though this 49-pound, 102-year-old patient was taken to the hospital by his mother); and so on.

This, then, is the bottleneck of which we spoke earlier: brittle response to unexpected situations. It is a characterization of software today: it is the quality that separates it from human cognition. The programs' limitations are both masked and exacerbated by the misleading sophistication of their templates for English outputs, by the blind confidence their users place in them, and by their being labelled with pretentious generalizations of their functionality (e.g., a "medical diagnosis expert system" today does just a narrow slice of differential diagnosis.)

Overcoming Brittleness

People are not brittle. Why? Because they have many possible ways to resolve a novel situation when it arises: asking someone for advice (this may include reading some written material), referring to increasingly general knowledge (eventually "first principles" or even "common sense"), comparing to a similar but unrelated situation. But each one of these paths to flexibility is closed to today's programs: they do not really understand natural language very well, they do not have general knowledge from which to draw conclusions, and they do not have far-flung knowledge to use for comparisons. Not only are programs vastly more narrow than we humans are, they are not equipped to dynamically grapple with a situation when it exceeds their current limitations. As we stated earlier, even the so-called expert systems, though they are the first attempt at iii), are only a "half-serious" attempt.

A serious attempt at iii) would entail building a vast knowledge base, one that is 10 (4) to 10 (5) larger than today's typical expert system, which would contain general facts and heuristics and contain a wide sample of specific facts and heuristics for analogizing as well. This KB would be distinguished by its breadth even more than by its size. Such a KB would have to span human consensus reality knowledge: the facts and concepts that you and I know and which we each assume the other knows. Moreover, this would include beliefs, knowledge of others' (often grouped by culture, age group, or historical era) limited awareness of what we know, various ways of representing things, knowledge of which approximations (micro-theories) are reasonable in various contexts, and so on.

Late in 1984, we began the first serious attempt at iii), and the bulk of this article describes that effort. We have made significant progress since then, and anticipate a kind of crossover (from primarily manual knowledge entry to primarily automatic entry via natural language understanding (NLU) later this decade. As the next article in this issue, "Knowledge and Natural Language Processing" and [2] explain in detail, one cannot expect to shortcut the building of the large KB today--let alone five years ago--by NLU, because open-ended NLU itself requires such a KB--for semantic disambiguation of word senses, resolving anaphora, inducing the meaning of ellipses, and so on.

Cyc: toward programs with common sense.

Interestingly, our work is spurring progress in i) and ii): in i), because the utility of various representation language features can best be judged by using the language; in ii), because what is wanted is not the most efficient inference procedure overall, but rather those that are used most often, so it is good to perform activity ii) in the context of a large "task-independent" test-bed KB.

Overview of the Cyc Project

Although our project Cyc emphasizes iii), building an immense KB requires that we also cover i) and ii). Namely, the KB must be built in some representation language, hence we have to include some of activity i). We also have to worry about some of ii) because the KB is going to be vastly smaller than its deductive closure, (i.e., in order to answer most queries, it will have to do some sophisticated inference). The next three paragraphs--and, in much more detail, the next three sections of this article--discuss our approach to i), ii), and iii): our representation language (CycL), our inference engine (actually many little ones), and our ontology.

Representation Language. We developed our representation language incrementally as we progressed with task iii). Each time we encountered something that needed saying but was awkward or impossible to represent, we augmented the language to handle it. Every year or two we paused and smoothed out the inevitable recent "patchwork." The latest language has been stable for 18 months; that is, its paradigm and "core" have remained stable, even though we have added many relatively minor extensions to it. To summarize it in a sentence, it is a frame-based language embedded in a more expressive predicate calculus framework along with features for representing defaults, for reification (allowing one to talk about propositions in the KB) and for reflection (allowing one to talk about the act of working on some problem.)

Inference Engine. The same "incremental engineering approach" was taken to building the inference engine. As we identified frequently used classes of inferences that could be used more efficiently, we introduced special mechanisms for this purpose. Traditional computer science has identified many problems having varying levels of complexity and has, devised special data structures and algorithms for solving them. AI on the other hand, has largely opted for single, very general mechanisms (e.g., resolution) for doing problem solving. We have adopted the former paradigm and are applying it to cover the kinds of problems that a system such as Cyc tries to solve. This approach is quite similar to that advocated in [4].

Ontology of the KB. As for the KB, we alternate bottom-up growth with top-down design. The bulk of the effort is currently devoted to identifying, formalizing and entering "microtheories" of various topics (e.g., money, buying and shopping, containers, etc.) We follow a process that begins with a statement, in English, of the theory. On the way to our goal, an axiomatization of the theory, we identify and make precise those Cyc concepts necessary to state the knowledge in axiomatic form. To test whether the topic has been adequately covered, stories dealing with the topic are represented in Cyc, then questions any human ought to be able to answer after reading the story are posed to Cyc.

There are currently between one and two million assertions in our KB, many of which are general rules, classifications, constraints, and so on; only a fraction (at present) are specific facts dealing with particular objects and events (e.g., famous people and battles.)

More significantly, we feel we have found "solutions" for various representation thorns that we might have become caught on: time, space, belief, hypotheticals and counterfactuals, interval-based quantities, substances, composite tangible and intangible entities, etc. By "solution" we mean the following: a set of partial solutions which work adequately in the moderately common cases, and which work very well in the very common cases. For example, a significant amount of work on the problem of representing aspects of agents such as their beliefs, goals, etc., in AI and philosophy, has focused on trying to reduce the total number of these propositional attitudes to the barest minimum and in trying to handle rather esoteric problems such as the Prisoners Dilemma [28]. We, on the other hand, have been quite promiscuous about inventing new propositional attitudes and have concentrated on more mundane issues such as predicting what the driver of a car on an American road probably intends when he turns on his left turn signal.

What do we hope to get from our efforts? Here are three possible levels of success, in decreasing order of optimism. It is interesting to note that we put the chance of the better result at less than 5 percent in 1984, but--due to our clipping of representation thorns, and stabilizing of the representation language, inference engine suite, and high-level ontology--we now place it as high as 60 percent:

* Good: While not directly built upon and widely used, the Cyc research does provide some insight into issues involved in task iii). Perhaps it gives us an indication as to whether the symbolic paradigm is flawed and, if so, how. It also might yield a rich repertoire of "how to represent it" heuristics, and might at least motivate future research issues in tasks i) and ii).

Cyc: toward programs with common sense.

* Better: Cyc's KB is used by the next generation of AI research programs, and its size and breadth help make them more than theoretical exercises. No one doing research in symbolic AI in 1999 wants to be without a copy of Cyc, any more than today's researchers want to be without EVAL and ASSOC. Eventually, it empowers the first full-fledged natural language understanding systems, non-brittle expert systems, and machine learning systems.

* Best: Cyc, or something similar, serves as the foundation for the first true artificial intelligent agent. Application programs routinely tie into it, in effect letting it look over their shoulder. No one in the early twenty-first century even considers buying a machine without common sense, any more than anyone today even considers buying a PC that cannot run spreadsheets, word processing, and networking software.

CycL--The Cyc

Representation

Language

CycL is the language in which the Cyc KB is encoded. Let us first consider some of the issues that heavily influenced the design of CycL. As we mentioned earlier, it is our aim that the Cyc KB be usable by many different problem solvers. This implies that the CycL should have a clear (and hopefully simple) semantics. We also want CycL to provide certain inferential capabilities and these should not be intolerably inefficient. Since most of our commonsense knowledge is in the form of defaults, CycL should provide some scheme for dealing with such knowledge. We would like to have all the expressiveness of first-order predicate calculus with equality, and we would also like a means for handling propositional attitudes (such as beliefs, goals, dreads, etc.) [26]. And finally we would also like to provide some facilities for operations such as reification, reflection, and so on. This, in short, is the "wish list" for CycL.

Epistemological Level and Default

Reasoning

Two of the "wish-list" entries seem to be at odds with each other: having a clean and simple semantics, yet providing speedy inference. To improve inferencing abilities, we want to include special-purpose representations and inference routines, procedural attachments, etc. But these make it harder to provide a simple semantics. Also, while 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 new special-purpose constructs is likely to prove bothersome to other programs that use Cyc--for instance, programs which were written before the new constructs even existed and hence could not take advantage of them.

We therefore would like users of Cyc (either humans or application programs) to interact with the system at an epistemological level and not at a heuristic level. These terms, and the distinction between them, are used in the sense as used by McCarthy and Hayes in [23]. These observations lead to the conclusion that the KB should be constructed at two levels, the Epistemological Level (EL) and the Heuristic Level (HL)--and this is exactly what we have done. The Cyc KB exists at these two levels and an external program (or human user) can interact with CycL at either of these levels.

The Epistemological Level (EL) uses a language that is essentially first-order predicate calculus (with a slightly different syntax) with augmentations for reification [20] "i.e., having a name for propositions, and being able to make statements about other statements) and reflection [32] (e.g., being able to refer to the facts supporting the system's beliefs in another fact in axioms). The EL is meant for giving an account of the KB in a form that has a simple semantics and is easy to use to communicate.

The Heuristic Level (HL), by contrast, uses a variety of special-purpose reporesentations and procedures for speedy inference. The HL is used whenever any inference needs to be done. It is best to think of the HL just as an optimization; i.e., to consider only the EL as "real," as containing all the knowledge.

CycL has a facility called Tell-Ask (TA) for translating sentences from the Epistemoligical Level into the most appropriate representations in the Jeuristic Level and vice versa. One can therefore type Epistemological Level expressions (i.e., in something like first-order predicate calculus) to TA, and they are converted into whichever Heuristic Level representation is most efficient (inverse, transfers Through, automatic classification, inheritance, etc.).

The actual First-Order Predicate Calculus (FOPC)-like logic used by the Epistemological LEvel is called the "Cyc constraint language" (CL). In addition to the expressiveness provided by this, CycL also allows sentences and function terms to be reified into objects. (1) The Constraint Language also allows some amount of reflection of the problem solver into the language. It also uses a number of modals (e.g., beliefs and desires) to talk about the propositional attitudes of agents.

Some of the assertions in Cyc's KB are monotonic (i.e.,

Cyc: toward programs with common sense.

the addition of new facts cannot cause them to be retracted). But most (over 90 percent) are non-monotonic: they are currently held default beliefs which can quite possibly turn out to be invalidated. Very little that we believe about the world is certain and absolute; that is true not only for heuristics (and conclusions derived using them), but also for most common-sense "facts" about the world. They often turn out to be simplifications ("Lincoln was a good President"), approximations ("The earth goes around the sun in an ellipse"), or, more rarely, just plain wrong (e.g., over half of American high school students believe that if you drop a wrench on the moon, it will just hang in mid-air there!). The monotonic (absolutely true) assertions are usually those that are definitional (e.g., it is absolutely certain that tall people are people) or provable (which usually means mathematical facts, such as "squares of odd numbers are odd").

Unlike many AI programs, most of whose default reasoning facilities are woven into the logic they use, Cyc uses only minimal support from the logic for doing its default reasoning, with most of the knowledge associated with default reasoning being represented as axioms in the KB [12].

The only non-monotonic constructs used are equivalent to the Closed World Assumption and the Unique Names Assumption. The Closed World Assumption is used only to provide the language with non-monotonicity, and the default reasoning abilities are designed using this and the notion of arguments. The syntactic structure of defaults is following that suggested in [22]. Thus, the statement "birds usually fly" is represented as follows: (2)

[Mathematical Expression Omitted]

To derive conclusions from this, we use the concept of arguments, so we have an argumentation axiom (instead of the circumscription axiom.)

An argument for a proposition P is similar to a proof for P, but is non-monotonic. For example, later information never invalidates a proof, once one is found, but might very well invalidate an argument. The essential differences between a proof andan argument are that, unlike in proofs, the sentences in an argument can be assumptions [25] and that arguments are first-class objects which can be referred to in axioms. The assumptions that can be made are sentences of the form * [ab.sub.i](...).

We then write more axioms that allow us to conclude P (or * P), given a set of arguments for and against P--- i.e., axioms which conclude that some argument is valid, or conclude that one argument is stronger than another.

Here is the Argumentation Axiom; it says to believe in a proposition P:

i) if there is an argument for it,

ii) the argument is not known to be invalid, and

iii) there is no preferred argument for * P (except perhaps some which are known to be invalid): (3)

[Mathematical Expressions Omitted]

A closed-world assumption is made for the predicates argumentFor and invalidArg. This axiom uses the truth-predicate True and in order to avoid the possibility of paradoxes we allow the truth-predicate to be partial. (i.e., (True('p) V True(' * p)) is not a theorem).

The salient aspect of this approach to doing default reasoning is that most of the "work" is done using axioms in the language and not "wired in" to the logic. The real core of the default reasoning is a set of additional axioms. The axioms in one group specify when an argument is invalid: if one of the assumptions made by an argument is false, then the argument is invalid. The axioms in the other group specify when one argument is preferred to another: In the former group, if one of the assumptions made by an argument is false, then the argument in invalid. In the second group, causal arguments are preferred over reductio ad absurdum arguments. This provides greater flexibility and control, and makes it easier to fix things if inadequacies are detected (i.e., adding/removing axioms from the KB is strongly preferable to changing the logic, especially when a massive KB already exists and assumes a certain logic).

This concludes the discussion of the Epistemological Level. A short description of some of the techniques used at the Heuristic Level to speed up inference follows.

The Heuristic LEvel:

Inferencing in Cyc

The Heuristic Level (HL) is meant for doing inferencing. As opposed to the Epistemological LEvel, where we tried to avoid superfluous constructs, the HL incorporates a host of "logically" superfluous mechanisms for improving efficiency.

Most of the novelty and power of Cyc stems from its rich, broad knowledge base; so why all this treatment of reasoning? Even though most commonsense reasoning is shallow, "shallow" still means one or two deductions away from what is already there in the KB. For instance, you

Cyc: toward programs with common sense.

have to make the following decisions: what to cook for dinner tonight; whether a wrench released on the moon will hang there or fall to the lunar surface; why someone just laughed; whether X is likely to already be acquainted with Y; etc. Most of the answerable queries are not preconceived. Their answers are not worth pre-computing and caching because they are numerous and, individually, each very unlikely ever to be asked (e.g., "Did Aristotle know about the Space Shuttle?" "Did Jefferson's right thumb have a thumbnail?"). Cyc can answer those questions correctly, giving "right" argumnts for its answers, and the ability to answer those questions is part of what it means to have common sense...yet it would be wildly cost-inefficient to try to store, let alone calculate, the answers to each such question ahead of time. The number of potentially useful short deductions from our current KB is in the trillions; so it is important to be able to quickly identify a small subset of sentences relevant to one's current problem, and it is important to be able to efficiently reason using those sentences.

The functionality of the Heuristic Level is defined in terms of a Functional Interface which consists of the following six operations which the HL must implement.

a) Tell: ([Sigma] x KB [arrow right] KB). "Tell" is used to assert statements. Given a sentence [sigma] and a KB, after Tell ([sigma], KB) we get a new (modified) KB' in which [sigma] is an axiom. Regardless of other arguments (multi-step "proofs") of [sigma], KB' would contain a new argument for it, of the form "Primitively, because the user told me so." [sigma] can be any well-formed formula of the EL language.

b) Unassert: ([Sigma] x KB [arrow right] KB). Given a sentence [sigma] and a KB, we get a KB' in which [sigma] is not an axiom. Nothing can be said about the truth-value of [sigma] in the resulting KB. For example, [sigma] might still be True (it still might be derivable from other axioms in KB'), it might be False ([Gamma] [sigma] might be derivable from axioms in KB'), or its truth-value might be unknown (neither [sigma] nor [Gamma] [sigma] are supported by arguments in KB'). Unassert is the direct "undo" of Assert. Note that Tell([Gamma] [sigma], KB) is quite different from Unassert([sigma], KB); the Tell would result in a KB' in which [sigma] was false, as justified by an explicit new axiom [Gamma] [sigma].

c) Deny: ([Sigma] x KB [arrow right] KB). Given a sentence [sigma] and a KB, after Deny~[sigma], KB) we get a KB' in which [sigma] is no longer true. It is common for neither [sigma] nor [Gamma] [sigma] to be true in KB' (i.e., if there are no other arguments for [Gamma] [sigma]). In other words, this squelches all positive arguments for [sigma], and does not affect negative arguments (arguments for [Gamma] [sigma]) in any way. Note that this is not the same thing as Unassert([sigma], KB), in which case [sigma] might still be true; and it is not the same as Tell([Gamma] [sigma], KB) in which case [sigma] would have to be false.

d) Justify: ([Sigma] x KB [right arrow] sentences). Justify is used to obtain the argument for a given proposition. If sentence [sigma] were true in KB, then Justify ([sigma], KB) would return a subset of the KB from which [sigma] can be derived. (Actually, Justify returns a somewhat more complicated value, which specifies the various pro and con arguments about [sigma], and how they combine to produce the "net" truth-value of [sigma] in KB.)

e) Ask: ([Sigma] x KB [right arrow] truth-value/bindings). "Ask" is used to test the truth value of a statement, and to find which free-variable bindings make an expression true. Given any constraint language (CL) expression [sigma] (which may contain free variables) and a KB, the value of Ask([sigma], KB) is either the bindings for the free variables in [sigma], or a truth-value. An optional argument turns Ask into a generator; i.e., each repeated call yeilds a single distinct binding list.

f) Bundle: (sequence of Functional Interface statements). This is a facility which performs a series of calls to the previous five FI functions as one atomic macro operation. This is of great pragmatic benefit, in two ways:

i) The operations may violate some integrity constraints and satisfy them again. For example, changing the domain of the predicate likes from Person to Animal requires one Assert and one Unassert. No matter in which order they are performed, after performing the first operation, there will be a violation nto integrity constraint that says that each predicate has precisely one recorded domain.

ii) The bundling allows the HL to be "smart" about which assertions it has to undo. For example, changing an inheritance rule from "Southerners speak with a drawl" to "Southerners over age 2 speak with a drawl" will result in n/35 retractions if they are Bundled together (assuming an average lifespan of 72 years, a uniform population distribution, etc.), rather than 2*n, if they are not.

The concept of a Functional Interface, with functions such as Tell and Ask, has existed in Computer Science and AI for some time [5]. We have tailored it for our purposes, and increased its pragmatic usefulness by adding some new constructs (such as Bundle and justify) and by teasing apart old ones (such as Unassert(p, KB) versus Deny(p, KB) versus Tell(*p, KB)).

Cyc: toward programs with common sense.

Default Reasoning Modules

Most of the gain in speed of processing at the heuristic Level comes about because of the way we implement Ask. (Much of the complexity at the Heuristic Level is due to the need to do Deny properly.)

Since most of the reasoning done is related to defaults, we first describe how this is implemented. (4) The structure of the heuristic Level is based around default reasoning and consists of these four modules:

* Argument Generator: Given a sentence, this module tries to generate an argument for it.

* Argument Comparator: Given a set of arguments for and against a sentence P, this module decides on a truth-value for P by comparing these statements. It then adds this sentece to the KB, with that "net" truth-value. Current truth-values include: monotonically true; true by default; unknown; false by default; and monotonically false.

* Conclusion Retractor: When the truth-value of a sentence x changes, this module ensures that truth-values of other sentences that depend on x are also updated. Not surprisingly, the module for the generation of arguments is, in practice, ver tightly integrated with this module.

* Contradiction Resolver: This module is responsible for detecting and resolving contradictions.

Though the epistemological Level has only two truth-values (true and false), the Heuristic Level uses 5 of them (true, default true, unknown, default false and false) to label sentences in the KB [11]. "True/false" sentences are those that are "monotonically" true (i.e., the addition of new facts cannot cause them to be retracted). "Default true/false" sentences do not have this property. "Unknown" is used for sentences for which there are unresolved conflicting arguments. Deductions that require making assumptions are only default true (or false) while those that do not require any assumptions are monotonically true.

Given a sentence P, Ask first tries to find arguments for it. If it can, it then tries to find arguments against it. These are then checked for possible invalidity, compared, and the final truth-value is decided on this basis. If there are unresolvable (incommensurable) arguments for and against P, then P is labelled as Unknown.

Since the Heuristic Level has these five truth-values, the Tell-Ask translator is able to covert axioms from the Epistemological Level into sentences at the Heuristic Level that do not contain any "ab literals" (assuming that no axiom has more than one negated ab literal). This makes the default reasoning both easier to encode and faster.

A number of the axioms (i.e., assertions that have been manually entered into the system) at the Epistemological Level are of the form ([~ab.sub.i(...)] * <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.

It should be noted that since comparing two arguments could involve using axioms in the KB, the Argument Comparator (or any of the other modules) can recursively use Ask or any of the other interface functions.

Speeding up the Argument-Generator

Module

The bulk of Cyc's time spent inferencing is used by the Argument-Generator module. a number of techniques have been introduced to make the Argument-Generator (and Conclusion-Retroactor) modules work more efficiently. These techniques fall into three categories:

* highly specialized interference rules,

* domain-specific inference modlues, and

* dependency analysis of the KB.

Highly Specialized Inference Rules

There are a number of groups of axioms whose syntactic structure can be captured using schemas that do not have any sentential variables. Each of these schemas is made into a rule of inference.

For instance, many rules we entered had the form (*x, y, z) s1(x, y) [and] s2(y, z) [right arrow] s1(x, z). For example, (*x, y, z) owns(x, y) [and] physicalParts(y, z) [right arrow] owns(x, z). If you own a car, and one of its parts is a certain steering wheel, then you also own that steering wheel. We introduced a new inference template, transfersThrough, so that one could express that rule simply as transfersThrough(owns, physicalParts). There are many other transfersThrough "rules" in Cyc, e.g., transfersThrough (lastName, father), so that lastName (MichaelDouglas, Douglas) and father (MichaelDouglas, KirkDouglas) imply lastName(KrkDouglas, Douglas). Another example of the use of this special-purpose inference schema is transferThrough(causes, agentOf); for instance, if X caused something to happen, while X was acting as an agent of Y, then we can consider Y to have

Cyc: toward programs with common sense.

caused it as well.

Associated with each inference schema--such as transfersThrough or inherits--are specialized procedures for speeding up that sort of interferencing. For example, a certain amount of compilation can be done that cuts down drastically on unification at runtime. 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 Warren-Machine-like [31] compilations of Prolog programs. In particular, each schema has specialized procedures for:

* Recognizing instances of the schema. For example, noticing when a constraint language sentence can be transformed into an instance of that schema. If the user Tells the system (*u, v, w) owns (u, v) [and] physicalParts(v, w) [right arrow] owns(u, w), that trivially matches the general transferThrough(s1,s2) template (*x, y, z) s1(x, y) [and] s2(y,z) [right arrow] s1(x, z), so the Tell-Ask translator converts that into transfersThrough(owns, physicalParts).

* Storing justifications. Each inference mechanism is responsible for detecting whem an argument it proposed becomes invalid, and (at that time) retracting the argument. Truth Maintenance Systems perform two tasks: providing this kind of bookkeeping, and maintaining consistency. Though typically tightly interwoven, in Cyc we see that these are kept clearly separated. The bookkeeping is inference mechanism-specific (the data structure used to represent the argument could be dependent on the inference mechanism that proposed it) while the consistency maintenance task (discussed in detail later, in the subsection on Denials) is inference mechanism-independent.

* Applying the schema. For example, suppose we assert these three senteces: transfersThrough(owns, physicalParts) owns(Guha, toyota0093) and physicalParts(toyota0093, WheelRR009382015)

Then a specialized procedure associated with transfersThrough would detect the need to "fire the rule" (if it were forward-propagated, or if it were backward-propagated and someone asked whether Guha owned WheelRR009382015).

We have also built a facility to help a user add new inference rule schemas [16]. For example, one specifies a schema, and Cyc automatically generates the code needed to "implement" this schema as an inference rule--the types of specialized procedures itemized above. This facility can only handle schemas not involving sentential variables.

Domain-Specific Inference Modules

The first category of specialized mechanism was based purely on the syntactic structure of the axioms, and had nothing to do with the domain with which the axioms dealt.

There are times when one can exploit some special properties of a set of domain-specific axioms, and/or domain-specific use of a set of general axioms--notably, information about "most frequently seen cases."

Some examples of such axiom clusters that Cyc currently optmizes in this way are those related to temporal reasoning [1], quantity arithmetic [33], equality, etc.

It should be noted that while there may be nothing more than the program representing these axioms at the Heuristic Level, these axioms do exist declaratively, explicitly at the Epistemological Level.

Dependency Analysis of the KB

In the past, AI has developed a number of standalone modules (e.g., Truth Maintenance Systems [9] that can be used with any problem solver.) In order to make them problem solver-independent, their operation was usually made independent of the contents of the KB the problem solver operated upon.

However, we have found that it is possible to obtain significant improvements (in efficiency) by using an analysis of the structure of the axioms in the KB.

For example, a dependency analysis of the axioms of the KB could reveal the circumstances in which there could possibly be circular justifications and identify the only sentences that may 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, in Cyc's KB, that only a handful of the four-thousand kinds of slots can even possibly participate in circular lines of reasoning and such "garbage collectable" chains are usually rather short; these two KB-specific properties make the problem of detecting them computationally quite feasable, in practice, even though it requires a rather expensive procedure in theory.)

Though on the one hand 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.

Dealing with Multiple Specialized

Inference Engines

Cyc: toward programs with common sense.

The two most critical issues that crop up in the presence of dozens of such specialized merchanisms are:

* When should a particular inference scheme be used? To determine which mechanism to use when, we associate with each predicate the set of features that may be used to deduce atomic formulae in which that predicate appears. Cyc also has a general-purpose inference mechanism that (though inefficient) is capable of a much larger (but sitll incomplete) category of inferences. This general inference engine is very similar to a unit preference resolution theorem prover.

* How does one integrate the operation of the different mechanisms? Each inference mechanism is expected to provide a set functions (in addition to one for deducing some class of sentences) for providing the argument justifying an inference, providing a list of instances of the inference rule, etc. Given these facilities, integrating these inference features is straightforward.

Each inference module can itself call any of the interface functions. (Ask, Deny, Justify, and Tell). For example, the mechanism for implementing the previously mentioned transferThrough schema calls Ask to verify the truth (or to find bindings satisfying) a particular sentence. When dealing with mechanisms other than domain-specific inference mechanism, a depth-first iterative deepening procedure [27] is used for the search. Resource-limited reasoning [3] is implemented by the indexical function resources-available, which specifies a cut-off depth, elapsed real time, or other resource bounds for the search. (The usual cut-off depth used is about 25).

In addition, parts of these inference mechanisms are represented in Cyc, and this reflection allows one to use an agenda to perform a best-first search using various heuristics to control the search strategy. The performance of the iterative deepening strategy has been good, however, that this meta-level [7] mechanism is rarely used.

Specifying Control Information For

Individual Assertions

A number of pieces of control information can be associated with each assertion (sentence) P. Some of these include:

i) Should the conclusions of the sentence P (the positive assertions, if any) be propagated in the forward direction?

ii) If backward-propagated, at what inference "level of effort" should this rule P be run?

iii) should the sentence P b treated as an integrity constraint?

Some of the motivation behind providing such annotations for axioms is to develop a set of cliched meta-level (proof theoretic) sentences about what actions were preferred, so that problem solvers could be written to exploit these directly. That effort is still under way.

Denials

The interface function Deny, though useful, is by far the most tricky one to implement properly. As we remarked earlier, note that Deny ([Delta],KB) is not the same (~[Delta], KB); the former will usually result in a KB in which [Delta] is unknown. For example, we might want to say that children of teachers typically go to college. But we might want to Deny that for children of gym teachers. This is not to say that we would guess that they very likely do not go college, just that we do not want to bet one way or the other. Of course, there might be other agruments as to why those people (as a general rule) do or do not matriculate, and in any particular person's case there might be other conclusive agruments for and/or against the assertion that they attend college.

Though Deny can in principle be implemented by a combination of Tells and Unasserts, in practice we have found it useful to define a new operation corresponding to the functionality described below.

If we write belief (Cyc, [Delta]) to say that [delta] is in the theory corresponding to the KB, then Deny ([Delta] KB) is equivalent to asserting ~ belief (Cyc, [Delta]). We can also specify the meaning of deny without resorting to belief as follows.

i) If the sentece [Delta] had been asserted by Tell and is "monotonically true" and there is no other way to derive it (i.e. it is an axiom and not a theorem), the Deny just deletes it fom the KB. So, in this case, Deny([delta], KB) reduces to Unassert([delta], KB).

ii) If the assertion [delta] follows from others in the KB or is a "local defaults," and is labelled "default true" (or "default false"), then we get the following two classes of denials.

-- Blanket Denial. This corresponds to introducing an axion that invalidates any argument for [Delta] (i.e., agrumentFor ([Delta, a) * invalidArg (a)). A less dogmatic version of this kind of denial is also available where only those arguments that are present at the time of the denial are asserted to be invalid.

-- Constructive Denial. One or more of the assumptions

Cyc: toward programs with common sense.

(i.e., formulae of the form ~ [ab.sup.i](...)) is chosen and asserted to be false in order to "defeat" existing arguments for [Delta]. Control over which assumption gets "retracted" can be exerted by using the predicate moreLikelyThan. If moreLikelyThan ('p1, 'p2) is true, then if a choice between p1 versus p2 needs to be made, p2 is chosen as the likely one to retract.

iii) If the assertion [Delta] follows from others and has been labelled "monotonically true," then attempting to deny it causes an error to be signalled. It is then adjudicated by the asserter, who has the option of retracting or reducing the truth-value (from monotonic to default true) of various assertions from the KB (which led to [Delta] being asserted as monotonically true), or (much more common if this is a "top-level" user operation) simply abortin the attempt to Deny [Delta], at least for the time being.

We conclude the discussion of CycL and proceed to discuss the contents of the KB. Further details of the CycL language may be found in [8, 12-14, 16, 19].

The Cyc Ontology

Recall that the EL (Epistemological Level) is meant for communicating the contents of Cyc independent of the "inferencing hacks" which are used for efficiency down at the HL (Heuristic Level). Hence, most of the discussion of the ontology of Cyc's KB in this article will be at the EL, not HL.

We begin by introducing some of the basic concepts and distinctions used, and later proceed to "representation issues" such as time, events, agent, causality, etc. This discussion is meant only to give a flavor for the kind of things that are present in the Cyc Kb and is not a comprehensive overview of what it encompasses.

Some Basic Concepts and

Distinctions

The ontology of Cyc is organized around the concept of categoris. We shall also refer to these as classes or collections. though we shall frequently use set-theoretic notions to talk about collections, these collections are more akin to what Quine termed Natural Kinds [29] than they are to mathematical sets. This shall become apparent later as we start ascribing various intentional properties to collections. The collections are organized in a generalization/specialization heirarchy (not a tree since each collection may have more than one direct generalization). the generalizations and specializations of a collection (that is, its supersets and subsets) will often be referred to as its genls and specs. Elements or members of a category are usually referred to as its instances.

Since this heirarchy is every important, we begin by discussing some of its important nodes and why they are in certain unintuitive genls/specs relations; we also discuss some of the partitions of categories (that is, dividing a category C into mutually disjoint subsets whose union is C).

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,' the string "Foo," et.--i.e., things for which the representation is provided by the Lisp substrate upon which CycL is written. Instances of RepresentedThing are things like Table, for which only a representation is provided by Cycl. This distinction is of use when deciding whether to use model attachments.

Another partition of Thing is into IndividualObject and Collection. IndividualObjects are things like Fred, TheWhiteHouse, TheFourthOfJuly1990,--i.e., the non-sets. They can have parts, but not elements (instances). Instances of Collection include Thing (the set of all things), Table (the set of all tables), 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. This was a conscious design decision, and has tremendous heuristic power as the KB is built. Namely, when a knowledge enterer has an urge to define a new kind of slot (i.e., benary relation), he or she must either select or define the domain (makesSenseFor) and range (entryIsA) of the slot. Usually, the slot is worth existing separately only if the domain is, and frequently gives the knowledge enterer a well-needed doublecheck on what he was about to do.

It should be noted that predicates such as age(x) and weight(x) cannot legally be applied to collections (such as Table). To rephrase: since Table is a set, a mathematical entity, it cannot have a weight or an age (it can of course have many other slots such as cardinality), that is, the domain of weight does not include collections such as Table. Of course we could discuss weight (Table905)--the weight of an element of the set Table--but that is quite different. Table is indeed a subset (spec) of IndividualObject, it is just not an element of (istanceOf) IndividualObject.

In addition to collections of individuals, we also have collections of collections. For example, PersonType is a set whose elements include Person, ComputerScientist, Texan, etc., which themselves are collections whose

Cyc: toward programs with common sense.

elements include Lenat, for example. The hierarchy folds into itself at this level and we do not have collections of collections of collections. (5)

It should be noted that unlike many frame systems, a distinction is made between the relations, instances (elements) and specs (subsets). So the relation between ComputerScientist and Fred (instances) is very different from that between Person and ComputerScientist (specs).

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 be easily first orderized). Although some of our editing tools (and internal data structures) gather together into "frames" the set of assertions that are binary predicates sharing a common first argument, that is merely a Heuristic Level (and user interface) distinction--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 Individuals [18], and then proceed to how we represent objects with temporal aspects to them.

Substances and Processes vs.

Individuals and Events

If you take a piece of wood, and smash it into ten pieces, each piece is still a (albeit smaller) piece of wood. But if you do the same for a table, each piece is not a (smaller) table. Substances are usually referred to in English as mass nouns; some of them are obvious (sand, air, peanut butter) 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 the other. If we do not have a single type of substance of which that individual is composed, we can define a new one (Bertrand-RusselStuff? ugh!), use a more general substance (AnimalMatter), or even fall back on the most general kind of substance of all, Substance.

Conversely, every piece of any substance--say this particular piece of peanut butter over here--is an individual. This gives us some interesting relations between substances and individuals.

Since every individual is a piece of some substance, Individual Object * Substance. On the other hand, any particular piece of any substance is an individual and, since the category corresponding to a type of substance is nothing but the set of its pieces, Substance * IndividualObject.

So, rather surprisingly, the two sets are extensionally equivalent. We still choose to distinguish between them since they have different intensional descriptions. More specifically, one of the differences in their intensional descriptions is as follows. The different substances (such as plastic, peanut butter, air, etc.) are all instances of the collection SubstanceType while the collections of individuals (Table, Person, Number) are instances of ObjectType. We shortly describe how this difference in intensional description is used.

There are certain properties that are intrinsic in that if an individual has them, parts of individuals also have them (at least as a default), while there are other properties that are extrinsic (i.e., parts of individuals do not have this property even if the individual does.) The notion of intrinsicness is closely related to that of substances in the following way. Consider a particular table made entirely of wood--Table103. It inherits various default properties from Wood, the kind of substance it is an instance of (properties such as density, flash point, etc.) and it inherits other properties from Table, the kind of individual object it is an instance of (properties such as number of legs, cost, size, etc.) The former properties are intrinsic, the latter are extrinsic. This is no coincidence! An object X (typically) inherits its intrinsic properties from whichever instances of SubstanceType X is an instance of, and X inherits extrinsic properties from whichever instances of ObjectType it is an instance of.

So we now have a way of predicting, for any known predicate, whether or not it will be intrinsic: determine whether its domain (makesSenseFor) is an instance of SubstanceType or ObjectType. This explains our earlier remark about how vital collections of collections are--we could actually dispense with the concepts Substance and IndividualObject (since they are coextensional), but we cannot do without SubstanceType and ObjectType.

Strictly speaking, it is always possible to carve up a substance so that the resulting parts are not instances of what the whole was an instance of. For example, one could take a glob of peanut butter and separate out all the peanut chunks, and these alone do not form a glob of peanut butter. So there is some restriction on how we may cut up a piece of some substance for the substancehood principle to apply. We associate a granule size with each kind of substance and the substancehood principle applies only to pieces larger than the granule of that substance. This allows us to deal with strange kinds of substance like military hardware which is usually considered a substance even though it consists of items like guns which are surely

Cyc: toward programs with common sense.

not substance-like.

Events and Persistent Objects

So far we have used the terms "piece" and "cutting up" in a very loose manner. There are actually two senses in which these terms can be used--spatially and temporally--and we shall now examine them both. This examination will lead to a discussion of more general issues concerning events and objects that occur and exist over some time interval.

We can cut up something spatially (as we did with the piece of peanut butter). We can also cut it up temporally. For instance, consider the process of 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 you walk to the corner mailbox and back home again. Imagine a videotape of that event, and now consider some contiguous one-minute segment of the tape--say the third minute of it. If someone watched just that minute, he or she would report that the minute was itself a (albeit shorter) walking event, that is, an instance of Walking.

In the last subsection, we noted that the class Wood had an interesting property: when a member of the class is physically carved into pieces each piece is still an instance of Wood. We then said that Wood was a type of substance (an instance of Substance Type), and we could use such "substance-like" categorization to decide on intrinsicness of properties. Here, we are seeing an analogous phenomenon: Walking, a class of events has the property that, when a number of the class is temporally carved into pieces, each piece is still an instance of Walking. We say that Walking is a type of temporal substance--what we will call a Process (that is, Walking is an instance of ProcessType). This turns out to be more than a superficial analogy. Indeed, Walking is an instance of SubstanceType. We now divide SubstanceType into TangibleSubstanceType and ProcessType. Wood, for example, is an instance of TangibleSubstanceType.

Similarly, ObjectType is now divided into TangibleObjectType and EventType. Even though Walking is a type of process, WalkingToTheMailboxAndBack is not. If you imagine that third minute of the ten-minute WalkingToTheMailboxAndBack event, it is still an instance of Walking, but a stranger watching just that 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 anywhere visible on the tape during that 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 it was illustrated that, surprisingly, Substance and IndividualObject were coextensional; as a special case, it turns out that Process and Event are coextensional. That is why ProcessType and EventType are actually more useful collections to have explicitly represented than Process and Event.

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

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 does not exist. Consider a temporal "slice" of Table001, such as the decade it was owned by Fred. This too is an instance of Table. This is interesting since it means that the category Table is an instance of ProcessType! Actually there exist a number of categories in our ontology whose instances are space-time chunks that have temporal aspects and which exhibit sufficiently persistent properties so that it makes sense to associate a notion of identity with these objects. The category of such things is called SomethingExisting and this is an instance of ProcessType. Since all physical objects (which have any persistent identity) exhibit this property, both TangibleSubstanceType and TangibleObjectType are specs of ProcessType! Consequently, anything that is spatially substance-like is also temporally substance-like, though the converse is not true.

This is an interesting view of concepts such as Lenat or Table001. We view these objects as space-time chunks and we call the temporal pieces of these (e.g., LenatDuring1990, Table001WhileBeingEatenOn) subAbstractions of the larger piece. SubAbstractions can of course have further subAbstractions. The maximal subAbstraction (e.g., Lenat, Table001) is called an Entity. Entities cannot have superAbstractions. Being space-time chunks these subAbstractions have temporal properties such as duration (the duration of Lenat is his lifespan), startingTime, endingTime, and so on. (6)

Cyc: toward programs with common sense.

Not all objects that have temporal extents exhibit enough persistence to warrant according them a persistent identity. Consider Roger dining at a restaurant. We can consider a system consisting of Roger, the waitress, the table, cutlery, food, etc., interesting enough to create an explicit object for this system. However, this object has no temporally persistent properties and is of little interest after Roger walks out (except perhaps as an example in an article). Such objects are instances of SomethingOccurring, which is another important instance of ProcessType; they correspond to reifications of what usually goes by the name of actions, scripts, or processes. The parts of such an object are referred to as its actors (though there are useful specializations (specSlots) of actors, such as performer, objectActedUpon, instrumentInAction, etc.). The various actors in an event usually undergo some change either during or after its occurrence (i.e., the subAbstractions of the actors during or immediately following the event are different from the subAbstractions immediately preceding). 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, since some of the properties of objects change simply by their existing (e.g., age), it could well be the case that the properties of something change even though it was not an actor in any instance of SomethingOccurring.

Any instance of Event can have temporal properties (duration, endsAfterTheStartOf, etc.) We use two abstractions of time to specify these temporal properties: interval-based and set-based [1], [24].

Let us 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 and/or ending times of these (possibly concave) intervals. For example, we define the binary temporal relation startsBeforeStartOf by stating the following assertion to Cyc: (*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 like "people do not eat and sleep at the same time" since 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, disjoint, etc., to state axioms like the one above. In this case, the sentence would just be an assertion that two intervals--viewed as a set of points--have 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 35 liked his house as it had been 20 years earlier" in this formalism, while it is difficult to do so with formalisms that associate time with propositions (or their reifications). There is, however a high cost associated with this. Given n entities and m intervals we can have up to n.m subAbstractions (O(n.m) objects) while using the other formalism we need only O(n+m) objects.

A vast majority of the statements we would like to make relate co-Temporal objects, and we would like to exploit this. We do so by having a predicate holdsDuring. So instead of having to create two concepts never again needed, and asserting livesIn-(FredFrom1965To1975, House1From 1965To1975), we can now just assert holdsDuring(Fred, lives, House1, 1965-1975). It turns out to be notationally much simpler to write complex axioms (where specific instances of SomethingExisting are replaced by variables) using the subAbstractions formalism, but it is more efficient to do inference using the holdsDuring predicate.

In addition to persistent objects such as Fred and nonpersistent objects such as FredGoingToWendysForDinnerOnJuly4th1990, 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 afterward, we can consider this change as an object. Formally this corresponds to reifying a sentence that specifies he was hungry at some time and not hungry at some later time into an object, and making this resultant object an event (since one can associate temporal properties with it).

Temporal Projection

When one of the properties of an instance of SomethingExisting changes, it is not likely to affect all (or even many) of its other properties [23]. For example, when Guha gets a haircut, it does not affect his address, languages-Spoken, birthDate, etc. This is not surprising, since a useful set of properties is useful partly because they are largely independent.

Associated with each ground formula are intervals of persistence. So if we knew that a gun was loaded at time t0 and the persistence interval of this was I1, then, given any point in time between t0 and t0+I1, we can conclude that the gun was loaded at that time point. Usually we associate default periods of persistence with classes of propositions by using axioms, which are called Temporal Projection Axioms. These enable us to project (infer a

Cyc: toward programs with common sense.

good guess for) Fred's name and gender years in the future or past, his hair style months in the future or past, his mood seconds in the future or past, etc., based on the values of those attributes at any given time.

These temporal projections are only defaults. If there is evidence contrary to these projections based on particular actions that have taken place, this contrary evidence usually overrides these projections.

Associating specific finite periods of persistence with propositions is much better than using a frame axiom [22] to allow for extended projection, but introduces the following problem. If our knowledge that the gun was loaded at t0 was derived from a source other than temporal projection, we are willing to say that up until time t0+I1 it is loaded. However, we do not want to carry on and say that at time (t0+I1)+I1 it is still loaded. That is, we want to project only from a base time point where we had that "other" source of information (i.e., a 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 refer to justifications for facts (which we obtained using reflection) to state this dependence.

Causality

Most treatments of causality (in AI) proceed by labelling some appropriate subset of occurrences of material implication as causal. We do this by using a relation causal whose argument is the reification of a sentence involving a material implication. For convenience, we shall refer to ((p * q) [and] causal'(p * q))) as (causes p q). Let us take a closer look at the axioms that specify the meaning of causes.

So suppose that we assert (causes p q). Then:

a) We have an inference rule that allows us to conclude that p implies (material implication) q from the above statement. Hence causes is a strictly stronger notion than material implication. That is, if p causes q, then p * q.

b) If p and q are ground sentences and true, they must refer to events (which in Cyc is anything which can have temporal attributes). That is, p and q must have at least one object constant that is an event. More importantly, every single event referred to in p must startBeforeTheEndingOf every event in q.

c) Given any atomic ground sentence q that refers to an event, either q should be "basic" or there should be some sentence p so that (causes p q) is true. Intuitively, q being classified as basic corresponds to the notion of it being "unexplainable."

d) Given a statement of the form (causes p q), either this is basic or there exists a sequence of sentences of the form (LogCause p a), (LogCause a b) ... (causes m q), i.e., some "mechanism" that implements this causal relation.

Both c) and d) are extremely strong statements to make, which is why the notion of "basic sentences" has been included. It would be nice to have a stronger definition of causality that makes sentences such as (causes False p) false and we are working on this. No commitment is made as to which occurrences of implication are to be labelled as causal. The aim of the above formalism is to provide a facility to state and experiment with various heuristics for accomplishing that purpose.

Actions and Concurrent

Processes

Each action (i.e., instance of SomethingOccurring) has associated with it a set of axioms specifying the preconditions for the action, the postconditions of the action, and other axioms specifying the constraints on the actors during the event. Each action also may (and usually does) have a set of subEvents, the composition of which is the overall event. This decomposition of an event into subEvents (which are also actions) is identical to the decomposition of an instance of SomethingExisting into its parts. In other words, the breaking down of a table into physical parts such as its legs, top, etc. is similar to the breaking down of having a meal at a restaurant into ordering food, eating, paying the bill, etc.

The structure of a physical object is defined by the constraints on its parts, and the structure of an event is defined by constraints on its subEvents.

Just as there may be orthogonal ways of breaking down a physical object, there may be orthogonal ways of breaking down an action into subEvents.

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, etc. We usually associate a predicate (which is an instance of Part Slot) with each of these classes and use these to relate the parts to the overall object (rather than using a single predicate such as parts or physicalParts).

A similar approach is taken to relating the parts of an action to the action. When dealing with actions there are

Cyc: toward programs with common sense.

two important categories of parts--two specSlots of parts, namely actors and subEvents--and there are separate categories of slots that are used to relate the actors to the particular action (the ActorSlots) and to relate the subEvents to the particular event (the SubEventSlots). The actor slots define the "roles" played by the different actors in the event (performer, victim, instrument...) Given an action and a participant actor, there are three subAbstractions of the actor related to that 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. This approach is motivated by two reasons: we wish to be able to reason at different levels of abstraction and a priori assigning of a set of actions as primitives goes against this; often one might be able to provide only descriptions and not definitions of the more complex actions in terms of their sub-Events. 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 [10, 23, 24]. A solution for this is obtained by collecting all overlapping events Ei (cutting up events if required) that affect a particular property into a single event E and computing the net effect of E on the property from the subEvents (suppressing the direct "updating" of the property by the subEvents). The basic idea is to agglomerate the various concurrent processes that affect some property into a single process which has no concurrent process that affects that property. The effects of this process are computed from those of the subEvents and the net change in the value of that property is that specified by this agglomeration process. It should be noted that the resulting agglomeration is necessarily a nonmonotonic process since a closed-world assumption has to be made while collecting the set of processes that affect our property.

When dealing with subAbstractions of reasonable durations, it becomes very difficult to specify values for most temporally instrinsic numeric attributes because of the (often slight) changes in the value of the attribute over the period of the subAbstraction. To overcome this difficulty, we introduced a new class of terms corresponding to intervals in the quantity space (of the attribute). These intervals may be named (e.g., "around 180 pounds") and explicitly represented as Cyc units e.g., #%Around-180lbs). The intervals may be open (unbounded) 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, etc. [33]. Another use for these interval-based quantity terms is to specify defaults for numeric attributes (e.g., height, weight, etc.) for categories which exhibit some but not too much variation in the value of these attributes (e.g., Fred's weight during September of 1990).

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) be defined: happiness, alertness, level of frustration, attractiveness, etc. Despite the lack of absolute units of measure, reified "mileposts" for these attributes' values can be defined, and partial orders and even crude calculi developed.

Composite Objects and

Agents

In addition to purely physical objects (such as tables and rocks) there exist objects like books and people with whom 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 out the purely tangible and the purely intangible parts, and represent both of them separately and explicitly as well as representing 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 CompositeTangibleObject is Agent--the set of intelligent agents--and this subsection considers some aspects of representing agents. But 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,..., we would much rather use the existing predicate age; besides being simpler and cleaner, this also lets us fully utilize the

Cyc: toward programs with common sense.

already-available axioms involving age.

To do this, we need to be able to explicitly talk about the physical and mental extents of a composite. Having done this (via 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; and age makes sense for all three aspects--and has a different value for all three.

This scheme gives 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,...) However, in most cases, there is no predicate that can be used for both the physical and mental extents that has different values and we would like to make use of this regularity.

In other words, we do not mind having three separate concepts for Frankstein's monster--he was rather unusual, after all--but we should not need to have three separate concepts for every composite if there is nothing "conflicting" among them. We accomplished this by adding the categories PartiallyTangible (a spec (subset) of SomethingExisting, and a genl (superset) of TangibleObject) and PartiallyIntangible (a spec of SomethingExisting and a genl of IntangibleObject). So CompositeTangibleIntangibleObject is now a spec of both of these new Partially...collections. Having done this, we can use a single unit, say Fred, to state both mental and physical properties of Fred. IQ, now makes sense for PartiallyIntangibleObjects, weight makes sense for PartiallyTangibleObjects; and Fred is an instance of both those collections and hence can legally have both an IQ and a weight. If we happen to be representing an exception, like the Frankenstein 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 earlier.

As a default, we inherit the properties that talk about physical/mental properties to the physical/mental extents. This gives both the expressiveness of the separation of physical and mental parts and the efficiency of not doing this when it is not required.

Since a full description of the various issues related to agenthood (that have been/are being) considered in Cyc would require more space than is available here we will mention only a few of them. One of our recent technical reports [15] 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 propositional attitudes toward any given proposition. The fundamental propositional attitudes currently used are believes and desires. From these two, using time and other concepts, a variety of other modals are described and used (e.g., dreads, purposes, expects).

A primitive notion of awareness is incorporated as follows. Each agent has a set of terms and predicates of which he is aware. An agent may have an attitude only toward sentences that involve only terms of which he is aware. This restriction is introduced to keep us from doing things like talking about Aristotle's beliefs about the Space Shuttle. We now consider some issues related to these propositional attitudes.

Attributing our own beliefs to other agents (with whom we might never have directly communicated) is something done quite frequently. Sometimes this is good--(e.g., when the traffic light in front of you turns green, you assume that the drivers on the cross street share your beliefs about what that means!)--and sometimes it is bad (e.g., cross-cultural "mirror-imaging" has led to innumerable political disasters.) There is a class of axioms called the belief projection axioms (analogous to temporal projection axioms) that enable Cyc to efficiently do this sort of mirror-imaging, yet explicitly record separate beliefs when they are known. The belief projection rules themselves are moderately interesting, since they describe what it means to be a public figure, what it means to be commonsense knowledge, etc. CycL provides special support to handle these efficiently at the Heuristic Level.

Agents can be in control of (the truth of) propositions. That 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 there is an agent in control of a proposition P, and he or she desires P, then we can assume that P is true (modulo limited resources, conflicting goals, etc.).

The concept of control provides us an abstraction layer that allows us to skip the details of the agent planning to make P true, executing that plan, monitoring it, repairing it, etc. Just knowing that you control the time you go home from work, and that you want to sleep at home tonight, gives me enough information that I will call you first at home at midnight if I have to reach you then; that is, I do

Cyc: toward programs with common sense.

not have to worry about the plan you made to ge home, the details of the execution, etc., in order to believe that (by midnight, at least) you would have arrived at home.

Agents may participate in Events (actually in instances of SomethingOccurring) in one of two modes: voluntarily or involuntarily. If an agent participates in an event voluntarily, he usually has a purpose (usually a propositional that is also one of his desires) that he believes will be true as a result of that event. The concept of purpose allows us to (write axioms which 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 may be individual agents, and some may be collective. An agreement defines a set of propositions that all the participants share (though they may have quite 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 usually also specify certain punitive and/or remedial actions to be taken in the case of these responsibilities not being fulfilled. If the agent performing these "punitive" actions is a LegalSystem (such as a Government or GovernmentalAgency) then the agreement is a LegalAgreement.

We distinguish between agreements in which the event that "enrolled" a particular agent was one in which he or she voluntarily participated and ones in which he did not participate voluntarily. For agreements an agent involuntarily participates in,the constraint that he or she 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, articles of incorporation, etc. Often an organization or institution will itself have (or at least act as if it has) certain desires, dreads, purposes for its actions, authority, etc., that are not obtainable by a simple combination of those of the participants.

Conclusion

This article began by explaining the need for a large, general KB: to overcome the brittleness (in the face of unanticipated situations) that limits software today. The need for a Cyclike KB is critical not only in expert system-like applications, but also for doing semantic processing for natural language understanding, and for enabling realistic machine learning by far-flung analogizing.

We then focused on criteria for an adequate representation language, which drove us to the bifurcated architecture of having both an expressive epistemological level (EL) and an efficient heuristic level (HL). One of the Cyc project's most interesting accomplishments has been the construction of the Tell-Ask translator, which can convert back and forth between general EL (first-order predicate calculus-like) expressions and special-purpose HL template instances.

Finally, we discussed some of the unexpected aspects of the Cyc KB's organization and contents, such as the relationships between Individual-Object, Substance, Process, and Event. And we gave the flavor of some of our recent research by sketching our still very incomplete treatment of Agents and Agreements.

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

The reader may have noticed several aspects of the Cyc effort which we have not touched on in this article. while interesting in their own right, these are not our main topic for research, and in each case we have done what we felt was necessary to maximize the rate of construction of Cyc. Here is a list of a few such intentional omissions from the article:

* The Knowledge Server: This subsystem accepts everyone's KB operations, serializes them, and, if constraint violations appear, adjudicates the resulution of the conflict. In cases of no conflict, it then broadcasts the operations to everyone else. The connections today are generally thin-wire, though we expect this to change in the coming year.

* The User Interface: This collection of tools includes various textual and graphical tools for browsing, querying, and editing the KB. Some of the graphical tools are semantic-net-based; one is an Escher-esque recursive birdseye view of a museum floor plan. Some of the editing tolls are ideal for making "point mutations" and corrections, some are oriented toward sketching some brand new area and gradually making the sketch more precise.

* The Machine-Learning Module: This subsystem roams over the KB, typically at night, looking for unexpected

Cyc: toward programs with common sense.

symmetries and asymmetries. These in turn often turn out to be bugs, usually crimes of omission of one sort or another. In very rare cases today, but, more frequently we jope, in future years, these will turn out to be genuine little discoveries of useful but hitherto unentered knowledge.

* Digitized Images: Yes, often it is much easier to just grab a picture of an object and point to the part you mean, rather than trying to figure out what its name is. Cyc contains such images (from the Visual Dictionary [6]), but experienced knowledge enterers rarely use them.

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

* The Copy and Edit Mechanism: Most knowledge entry in Cyc involves finding similar knowledge and copying it, and modifying the copy. Increasingly 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. This is good since the number of assertions before reaching the NLU crossover point also appears to be larger than our 1984 estimate. These two discrepancies are not unrelated: many of the extra assertions deal with overcoming ambiguities, with being precise about a cluster of closely related concepts, and that means that Cyc can help the user copy a whole cluster or related "thin" concepts in approximately the time we expected it to take to copy one of our original "fat" concepts.

How are we to judge where we are going? How do we make sure that we do not go through these 10 years of labor, only to learn in 1994 that the assumptions upon which we based our efforts were fundamentally mistaken all along? We do this be getting others to actually use our system. In the past 18 months, as the Cyc Representation Language and Ontology stabilized, we began to encourage collaborations both with academic researchers and with industrial researchers and developers: we held workshops and panel sessions; and we have begun once again (after a purposeful several-year hiatus to focus solely on research) to write books and technical reports and journal articles, such as this one, to inform and interest the greater artificial intelligence and computer science communities.

Cyc is still too small to have more than an anecdotal chance of improving the performance of application programs using it, but the early results are promising. AT DEC, for example, John McDermott, DAvid Marques, Renata Bushko, and others have built a Cyc-based computer-sizing application. Serving as a pre-processing step for XCON [30], its job is to ask questions about a potential DEC customer and come up with a very rough computer sizing. The trouble with having standard expert systems do this task is they tend to ask too many questions, questions which can often be answered by common sense, questions for which Cyc is able to guess answers. (For example, given that toy manufacturers have stringent government safety regulations, and adult clothing manufacturers do not, which is more likely to be the proper "match" or "precedent" for this new potential customer who is a manufacturer of children's clothing? Or: 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 kind of place to decide that for a new potential customer which is a hospital the right business unit is the bed, not the room.)

Numerous other Cyc-based applications are under way at NCR, Bellcore, US West, and Apple. Academic collaborations include coupling with large engineering knowledge bases (with Ed Feigenbaum and Tom Gruber at Stanford), large data bases (with Stuart Russell and Mike Stonebraker at Berkeley), standardizing knowledge interchange formats (with Mike Genesereth at Stanford), axiomatizing human emotions (with John McCarthy at Stanford), machine learning by analogy (with Devika Subramanian at Cornell), and qualitative physics reasoning in the service of understanding children's stories (with Ken Forbus at Illinois). And of course one vital collaboration is with Elaine Rich and Jim Barnett at MCC, namely the natural language understanding project which is described in [2]. testing the system. We also thank Ed Feigenbaum, Pat Hayes, John McCarthy, John McDermott, and Marvin Minsky, who have in almost orthogonal ways helped us to think more clearly about the material presented herein. We thank Bobby Inman and Woody Bledsoe for setting up MCC, the only place in the US where a high-risk high-labor long-term project like Cyc could be done; conversely, we appreciate our shareholders' sticking with us in the five first, riskiest years. Finally, we wish to acknowledge our significant debt to the AI community in general; Cyc is built upon a rich foundation of three decades of CS and AI research results, only a small fraction of which we have explicitly cited in the References.

(1) The reification of a function term is different from the value of that function term. For example, it might then be referred to in a proposition about how costly its evaluation might be, which proofs depend on knowing the value, and

Cyc: toward programs with common sense.

other meta-level assertions.

(2) The "isa" predicate corresponds to the set-membership relation; it is sometimes called ISA, is-a, AKO, element-of,. . . In Cyc's KB we happen to call this instanceOF. Also, the "ab1" predicates are short for abnormal in fashion i; so ab1 corresponds to being an exception in the sense of being a bird and not being able to fly.

(3) Note that 'p, read "quote p," refers to the sentence p, rather than to its truth-value. Normally, one is free to substitute "equals for equals" in mathematical or logical formulae, but think of the trouble you would get into with "Fred believes Mary's age is 39' if it turns out that Mary is 40. We certainly do not want to do the substitution and conclude "Fred believes 40 is 39." To prevent this sort of problem, assertions and formulae (such as "Mary's age") can be quoted in this fashion.

(4) The argumentation axiom is just like any other axiom at the Epistemological Level. However, since it is used very often, at the Heuristic Level, there are some special procedures for incorporating it.

(5) We used to, but they were never much use. Collections of collections, however--such as PersonType and SubstanceType and EventType--have proven vital.

(6) This is superficially similar to the "histories" framework [17] but is different in a very important way: there is no relation between the intersection of these histories and the frame problem.

References

[1] Allen, J. Maintaining knowledge about temporal intervals. IN Readings In Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[2] Barnett, J., Knight, K., Mani, I. and Rich, E. Knowledge and natural language processing. Tech. Rep. ACT-NL-104-90, MCC, March 1990 (Also appears in this issue of CACM).

[3] Bobrow, D.G. and Winograd, T. An overview of krl, a knowledge representation language. In Readings In Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[4] Bledsoe, W.W. Non-resolution theorem proving. In Readings in Artificial Intelligence, B.L. Webber and N.J. Nilsson, Eds., Morgan Kaufmann, Los Altos, CA, 1981.

[5] Brachman, R.J., Fikes, R.E. and Levesque, H.J. Krypton: A functional approach to knowledge representation. In Readings In Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[6] Corbell, J.-C. The Visual Dictionary. Facts on File, New York, 1987.

[7] Davis, R. and Buchanan, B.G. Meta-lelvel knowledge: Overview and applications. In Readings IN Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[8] Derthick, M. An epistemological level interface for cyc. Tech. Rep. ACT-CYC-084-90, MCC, February 1990.

[9] Doyle, J. A truth maintenance system. In Readings IN Nonmonotonic Reasoning, M. Ginsberg, Ed., Morgan Kaufmann, Los Altos, CA, 1987.

[10] Forbus, K. Qualitative physics: Past, present and future. In Exploring Artificial Intelligence, H. Shrobe, Ed., Morgan Kaufmann, Los ALtos, CA 1988.

[11] Ginsberg, M. Multivalued logic. In Readings In Nonmonotonic Reasoning, M. Ginsberg, Ed., Morgan Kaufmann, Los Altos, CA, 1987.

[12] Guha, R.V. The representation of defaults in cyc. Tech. Rep. ACT-CYC-083-90, MCC, February 1990.

[13] Guha, R.V. and Lenat, D.B. Cycl: The cyc representation language, part 2. Tech. Rep. ACT-CYC-452-89, MCC, December 1989.

[14] Guha, R.V. and Lenat, D.B. Cycl: The cyc representation language, part 3. Tech. Rep. ACT-CYC-454-89, MCC, December 1989.

[15] Guha, R.V. and Lenat, D.B. The world according to cyc, part 2: Agents and institutions. Tech. Rep. ACT-CYC-453-89, MCC, December 1989.

[16] Guha, R.V. and Lenat, D.B. Cycl: The cyc representation language, part 4. Tech. Rep., MCC, April 1990.

[17] Hayes, P.J., Naive physics 1:Ontology for liquids. IN Formal Theories of the Common Sense World, J.R. Hobbs and R.C. Moore, Eds., Ablex, Norwood, N.J., 1985.

[18] HAyes, P.J. Some problems and non-problems in representation theory. In Readings IN Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

Cyc: toward programs with common sense.

[19] Lenat, D.B. and Guha, R.V. Building Large Knowledge Bases. Addison-Wesley, Reading, Mass., 1990.

[20] McCarthy, J. First order theories of individual concepts and propositions. In Readings In Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[21] McCarthy, J. Programs with common sense. IN Readings In Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los ALtos, CA, 1986.

[22] McCarthy, J. Applications of circumscription to formalizing common sense knowledge. IN Readings In Nonmonotonic Reasoning, M. Ginsberg, Ed., Morgan Kaufmann, Los Altos, CA, 1987.

[23] McCarthy, J. and Hayes, P.J. Some philosophical problems from the standpoint of artificial intelligence. In Readings In Nonmonotonic Reasoning, M. Ginsberg, Ed., Morgan Kaufmann, Los Altos, CA, 1987.

[24] McDermott, D. A temporal logic of reasoning about process and plans. Cognitive Science, 6 (1982), 101-155.

[25] McDermott, D. and Doyle, J. Nonmonotonic logic. In Readings In Nonmonotonic Reasoning, M. Ginsberg, Ed., Morgan Kaufmann, Los Altos, CA, 1987.

[26] Moore, R.C. The role of logic in knowledge representation and commonsense reasoning. IN Readings IN Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[27] Pearl, J. and Korf, R. Search techniques. Annual Review of Comput. Sci., (1987).

[28] Poundstone, W. Labyrinths of Reason. Doubleday, 1988.

[29] Quine, W.V. Natural kinds. In Ontological Relativity and other essays. Columbia University Press, New York, 1969.

[30] Soloway, E., Bachant, J. and Jensen, K. assessing the maintainability of xcon-in-rime: Coping with the problem of a very large rule-base. IN Proceedings of AAAI-87 (1987 pp. 824-829.

[31] Warren, D.H.D. An abstract prolog instruction set. Tech. Rep. 309, SRI, Artificial Intelligence Center, Computer science and Technology Center, October 1983.

[32] Weyhrauch, R.W. Prolegmena to a theory of mechanized formal reasoning. In Readings IN Knowledge Representation, H. Levesque and R. Brachman, Eds., Morgan Kaufmann, Los Altos, CA, 1986.

[33] Williams, B. Minima: A symbolic approach to qualitative algebraic reasoning. In Proceedings of AAAI-88, 1988.

CR Categories and Subject Descriptors: C.5 [Computer Systems Organization]: Computer System Implementation; D.3.3 [Programming Languages]: Language Constructs--Control Structures, Data Types and Structures; F.4.1 [Mathematical Logical and Formal Languages]: Mathematical Logic; H.2.8 [INformation Systems]: Database Management--Database application; I.2.1 [Artificial Intelligence]: Applications and Expert Systems--NAtural language interfaces; I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving--Deduction (e.g., natural, rule-based); I.2.4 [Artificial Intelligence]: Knowledge Representations and Formalisms--Relation systems, representation languages, semantic networks

General Terms: Design, Human Factors

Additional Key Words and Phrases: Cyc, knowledge bases

DOUGLAS B. LENAT is Principal Scientist at MCC and Consulting Professor of Computer Science at Stanford University. His pioneering work in Machine Learning led him to chafe at the "brittleness bottleneck," and in 1984 he established the Cyc project. He has authored more than 50 published papers and has written and edited several books, including Knowledge Based Systems in Artificial Intelligence and Building Expert Systems.

R.V. GUHA is a mechanical engineer and computer scientist who is pursuing a Ph.D. in Computer Science at Stanford University. He has authored several papers and technical reports, and (with coauthor Doug Lenat) a recent book: Building Large Knowledge Based systems: Representation and Inference in the Cyc Project. Guha is interested in investigating the role that contexts play in everyday reasoning.

KAREN PITTMAN is a botanist who has spent the last three years adding knowledge to Cyc. Initially, she was involved in the construction of the UT Computer Science Department's Botany Knowledge Base. She has authored papers in the American Journal of Botany and in the Biotechnology and Ecology of Pollen.

DEXTER PRATT is a chemist and reformed entrepreneur (prior to joining MCC, he was president of Red Shark Software). Before that, he was a long-time employee of

Cyc: toward programs with common sense.

Lisp Machine, Inc. (LMI), performing a variety of tasks including processor design, software development, and technical management.

MARY SHEPHERD is a sociologist and engineer, and has worked on the Cyc project since its inception. She has authored articles on interface tools for browsing and editing large KBs. Prior to her involvement in Cyc, she worked at Thinking Machines, Inc. (TMI) and was assistant to the Vice-President for Information Technology at Harvard.

Authors' Present Address: All authors are members of the Cyc Project technical staff at Microelectronics and Computer Technology Corporation (MCCe, 3500 W. Balcones Center Dr., Austin, Texas 78759. Their email addresses are ai. <last name> @mcc.com (e.g., ai LEnat@mcc.com).