

Ideas for Applying Cyc

MCC Technical Report Number ACT-CYC-407-91

MICROELECTRONICS AND COMPUTER TECHNOLOGY CORPORATION

Advanced Computing Technology
Artificial Intelligence Lab

December 1991

Copyright © 1991 Microelectronics and Computer

All Rights Reserved. Shareholders of MCC may reproduce and distribute these materials for internal purposes by retaining MCC's copyright notice, proprietary legends, and markings on all complete and partial copies.

Contents

1. INTRODUCTION	2
1.1 OUTLINE OF THIS DOCUMENT	2
2. A BRIEF LIST OF THE APPLICATIONS	3
3. ADVICE SERVICES	4
4. DIRECTED MARKETING	6
4.1 POINT-OF-SALE MARKETING	7
4.2 DIRECT MAIL MARKETING	8
5. ONLINE BROKERING OF GOODS/SERVICES	9
6. DATA BASE CLEANING	12
6.1 'SANITY CHECKING' OF DATA	12
6.2 DIAGNOSING THE SOURCE OF THE ERROR	13
6.3 CORRECTING THOSE ERRORS	14
6.4 INFERRING ABSENT ENTRIES	15
6.5 NOTICING PATTERNS IN THE DATA	15
7. DATA BASE INTEGRATION	16
8. CORPORATE KNOWLEDGE ASSETS MANAGEMENT	17
9. SMART SPREADSHEETS	19
10. SMARTER INTERFACES	20
11. MACHINE TRANSLATION OF TECHNICAL DOCUMENTS	21
12. ENRICHED 'ARTIFICIAL REALITY'	23
13. CONCLUSION	27

Ideas for Applying Cyc

MCC Technical Report Number ACT-CYC-407-91

1. Introduction

The CYC project is now sufficiently far along in its development that we can foresee numerous possible applications of it, in the near term. This document lists ten such ideas for applying Cyc.

The total "terms" of the projects vary. Some are as short as 1 year away from being fielded, most are 2 - 3 years out, and a few as distant as 5 years out. Nevertheless, they share in common the following key attribute: it is appropriate to begin development on each of them *now*.

Many more 'blue-sky' applications using Cyc are possible (e.g., open-ended machine translation or natural language understanding), but we have avoided them here, due to their longer term, smaller chance for success, or the inappropriateness of beginning work on them today.

There are also many other much shorter term and lower impact applications of Cyc and CycL that have not made it to the list given here. The two major criteria used for obtaining the list of applications given here are that no major technical breakthrough should be required to make the application possible and that the application, if it succeeded should have very significant financial implications (at least on the order of a billion dollars a year.)

1.1 Outline of this document

In Section 2 we list - very briefly - the ten applications, then discuss them one by one in detail, in subsequent sections.

The Conclusion, Section 13, gathers together a set of criteria for what is and is not a good Cyc application.

Appendix A recounts the various ways in which using Cyc can provide added "power" to the applications, namely through the CycL representation language being expressive and efficient, and, more importantly, through the tremendous size, more or less complete breadth, and stable organization of its knowledge base. It is that last point, the content of Cyc, which would be most difficult for a competitor to "fast-follow."

2. A Brief List of the Applications

Ten applications are presented in this section in fairly specific, concrete form; i.e., a thumbnail "scenario of use." In later sections, when we treat each application in turn, we will suggest additional tasks in that same category, and explain how and why Cyc could be used.

(1) Advice services

Over a network (e.g., CompuServe), offer a service that helps people select which type of new car to buy.

(2) Directed marketing

Use a person's buying history to infer their hobbies, interests, occupation, physical needs and preferences, etc. From that model, decide which products to try to sell them, and what "argument" to use to convince them they should buy the product.

- Point-of-sale marketing. When someone enters a store, they 'swipe' their ID card across a reader, and receive some custom-printed coupons (with brief, custom-tailored "arguments.")
- Direct mail marketing. As above, but can be longer, glossier, and more detailed, and can include "products" not sold in stores (e.g., 900-... numbers.)

(3) Online brokering of goods/services

Using a common vocabulary and shared fundamental knowledge about transactions, enable a large set of buyers/sellers to find each other and negotiate deals - online and (semi-)automatically.

(4) Data base cleaning

Relate a data base's fields and keys to Cyc predicates and terms, and use Cyc's common sense constraints to detect possible errors and inconsistencies in that data base.

(5) Data base integration

Use that same sort of "articulation" approach to have several heterogeneous data bases all relate their contents to one central knowledge base. Use this to (1) detect and resolve contradictions *among* data bases, and (2) handle queries that require accessing - and integrating the results from - multiple DB's.

(6) Corporate knowledge assets management

Represent a company's know-how, policies, important documents, programs, and data bases - all in a form that a program can effectively reason with. Use as an online policy manual, or as a sort of "smart Yellow Pages" for employees.

(7) Smart spreadsheets

Explain the "meaning" of the rows and columns, by tying them to a large corpus of general knowledge about occupations, human capabilities, goals, household objects, etc. Highlight abnormal (though not illegal) values, and suggest places where there seems to be a missing equation or constraint.

(8) Smarter interfaces

Use an online model of the "meaning" of what the application program is doing, what the user's goals are, what constraints they're under (e.g., deadlines), etc., to adjust the interface. E.g., reorder questions on a 'form'; rearrange windows; highlight or cross out various parts; guess at values to fill in certain 'blanks;' and so on.

(9) Machine translation of technical documents

Provide a set of flexible templates into which 95% of some class of technical documents (e.g., installation instructions for a laser printer) can be captured. Then, using currently-available natural language software,¹ process such documents, converting them into a set of filled template instances. Conversely, the 'first writing' of a manual could itself be done by filling out such templates. Translations into different languages could then be produced by composing the translations of each template, and each (disambiguated) term used. Difficult portions get flagged for checking by hand.

(10) Enriched 'Artificial Reality' (AR)

Simulated 'worlds' in which physical objects behave "realistically," and machine-run personae behave "intelligently" - particularly when one interacts with those objects and personae in ways that were not specifically preconceived. In addition, complex, changing images must somehow be transmitted in real time, among a large set of interacting human users, at compression ratios exceeding one million to one.

In the next ten sections, we focus on each of the applications listed above. We go into more detail of what the application would be, why and how Cyc would be useful, list some alternative specific usage scenarios, and try to characterize what the "general opportunity" really is.

3. Advice Services

First, let's reiterate the terse description of one usage scenario for this application:

Over a network (e.g., CompuServe), offer a service that helps people select which type of new car to buy. Cyc is useful because almost any aspect of their life and situation might be relevant to the choice.

¹ The software developed at General Electric by Paul Jacobs and Lisa Rau, in conjunction with Cyc, would be ideal for this purpose.

We built a full-scale system for this task in Spring, 1991. For some details, see Appendix B. The prospective buyer has a `form' to fill out - a list of questions which may be answered in any order. Cyc is used as follows:

- Draw on knowledge already entered in its KB, to determine new questions that should be asked and to provide a menu of legal answers to many questions (e.g., regarding where the person lives, their occupation, marital status, image they wish to project, handicaps, hobbies, and so on.)
- Use the answers given so far to infer new constraints. (E.g., a Realtor will likely want a 4-seater, even if they live alone.)
- Detect contradictions (or unlikely values) in the answers given so far. These turn out to be typo's, or cases where the user misunderstood the question, etc.
- Use its TMS (truth maintenance system) to allow the user to relax or remove constraints (e.g., if there's a contradiction, or if they've overconstrained the selection.)

More generally, Cyc can be used as the framework for a `selecting an X ' application. Besides cars, three other promising choices for X are: (a) which colleges to look into attending; (b) which stereo system components to buy; and (c) which information service (e.g., data base) to query.

That latter application, (c), would make use of knowledge about information services; e.g., a data base data base. This concept has surfaced before; it is, e.g., what Marvin Weinberger of Telebase calls a ``metabase."

Actually we need a *knowledge* base about data bases.² E.g., use Cyc to store information about the scope (breadth and depth) of each data base, its particular places of accuracy/timeliness/completeness and inaccuracy/outdatedness/sparsity, resource costliness (money, time delays, etc.,) cases in which relying on it has led to good or bad results, and so on.

A variant of this, which people are likely to frequently want to consult, would be an oracle that provided advice about how to deal with various organizations (e.g., the IRS, the INS, the Phone Company, the Post Office, a Car Dealer, etc.) in order to achieve some goal. People often want advice about such things without wanting to ask those organizations directly. All sorts of ``common knowledge" would be required for this, in addition to detailed policies and scripts for each organization - e.g., knowledge about negotiation, emotions, communication, etc.

A related service to `selecting an X ' would be `debugging an X ' - e.g., for helping to diagnose problems with household appliances. One nice touch here is that the appliances themselves will, more and more over time, incorporate ``embedded" computers which can directly feed state information to the diagnostic program.

As wireless palmtop computing becomes a reality, we foresee even more opportunities for useful possible topics of advice.

² Note that by "*knowledge* base about databases" we mean knowledge about the *content* of the databases and not the database theory.

One could think of all these "advice services" as expert systems that deal with mundane tasks, and are robust enough (user-friendly and novice-proof)³ for use by laypeople. Even if Cyc singlehandedly provides the robustness and novice-proofing, this sort of application will not take off if the investment required for building each of these advice-providers is too large. And that might very well be the case if building a particular application ends up being as big a project as building a large expert system from scratch. Cyc can accelerate the construction of such ES-like bodies of knowledge, in two ways:

- (i) By providing a vocabulary and a rich body of axioms to start with, and a context mechanism and numerous particular contexts.
- (ii) By providing specialized problem solving contexts for standard tasks such as "Selecting an X," "Diagnosing an X," etc.

4. Directed Marketing

Use a person's buying history to infer their hobbies, interests, occupation, physical needs and preferences, etc. From that model, decide which products to try to sell them, and what "argument" to use to convince them they should buy the product.

Of course, the model of the person might be based on all sorts of information about them, besides their buying history; e.g., school and employment records, police records, credit bureau information, banking transactions, motor vehicle registries, tax records, medical records, etc.

Rather than simply recording these different sorts of information about a person, we should also be able to infer some of the person's habits, lifestyle, interests, personality traits, areas of intelligence and irrationality, passions, and so on.

Moreover, combining the model (built over an extended period of time - perhaps the person's entire lifetime) with recent actions, we might even be able to *predict* changes in their lifestyle, habits, needs, etc.

The model (whether "static" or "predictive") of an individual person could be used by shops and direct mail advertisers to infer what products and services s/he needs (or could be convinced into buying.)

Many other kinds of organizations might be able to make effective use of the resulting user-models. E.g., the public library might send out notices of particular books the person might like to read. The IRS might use the models to do a better job of predicting who to audit, and for what. The FBI might use such models to make guesses about who might have committed some crime, where someone might be hiding, what new identity they might have assumed, and so on. They might be used to detect instances of "Welfare cheating," misuse of food stamps, or child abuse.

³ We use the term novice-proof rather than idiot-proof, because all of us need help when we first enter an unfamiliar domain.

This sort of model building is done to a very limited extent today by credit rating organizations. Typically, the "user model" follows a rigid template in each application, but there is no universal "standard" format, hence no easy automated means of hooking up this information to arbitrary other programs, or to other repositories of data about the same individuals. Cyc allows for a much more expressive, open-ended model; and Cyc itself could fill in most of the inferrable aspects of the model and also help significantly in using the model to make organization-specific predictions (like predicting what s/he might like to buy.)

More generally, one of the prerequisites to having a model that can cover a very large range of aspects of an individual is the vocabulary (and heuristics that go with this vocabulary) for giving the machine information about these different aspects. Cyc already has the vocabulary for representing a very large array of information about people, and hence forms the ideal substrate on which such a generic modeling application can be built.

As with every conclusion Cyc reaches, each such recommendation would come with an *argument* - a sequence of sentences similar to a proof, but incorporating assumptions that are only true by default. That argument might be printed out, along with the coupon or letter. Since it's customized for that individual, it might be quite powerful, indeed, as a tool to convince them to follow its conclusion and buy the product or service.

Of course in many cases, the argument leading to the recommendation is not the one we would want to present to the person. For instance, suppose the "real" argument for Fred buying a torque-wrench-calibrator is that he is infatuated with tools that have technical-sounding names. That might be a good argument for sending Fred an advertisement for the tool, but explicitly *telling* him that is not going to increase the chances of making the sale. What sales pitch should we present to Fred, then? Think of it itself as being a reasoned-about object. What we tell the customer is based on knowledge about people's goals, their emotional and psychological needs and capabilities, data about which sorts of arguments Fred seems most susceptible to, etc.

4.1 Point-of-Sale marketing

When someone enters a store, they 'swipe' their ID card across a reader, and receive some custom-printed coupons (with brief, custom-tailored "arguments.")

Separate technologies - with increasingly high hardware costs, and (hence) longer time-frames - would be used depending on exactly where during the shopping "script" the marketing is performed:

- **At the checkout register.** This is the easiest and shortest-term. It could be an extension of the already-existing point-of-sale coupon-printing that goes on in many supermarkets. Instead of just simple rules (e.g., "If they bought Coke, print out a coupon for the same size Pepsi," however, the full power of Cyc could be used. The rules could be more subtle; they might "correct" purchases, instead of merely being coupons for use next time (e.g., "If they buy some device that requires batteries, and they buy an insufficient

number, or the wrong size, Then point this out to them;") and the things being printed out might be notices of interest to this person, rather than just coupons (e.g., people who regularly buy quilting products might like to know about a local quilt show.) Notice that all of these could make good use of a persistent user-model, rather than only knowing what the person bought this time.

- **When entering the store.** This is slightly non-standard. The same checkout-register technology could be used, but it would have to be made available at the entrances to the stores. Moreover, it would be best if a persistent user-model existed, since these coupons would be ones that the person would (hopefully) use that same trip, not the next trip. Because of the same-trip redemption feature, the percentage of coupons being redeemed might be much higher. Another benefit this could have, for the customer, is to begin the automated credit-checking process long before they get to the checkout register, rather than delaying them once they get there.
- **During shopping.** One way this would work is if you swipe each item as you pick it up. This requires a major equipment acquisition, by a store, to equip each shopping cart (or each aisle) with suitable I/O devices. The advantages are, of course, the possibility of having a "dialogue" about a product, maybe even negotiating a better coupon on the spot if you try X instead of Y, or a larger size than the one you picked up, etc. A mundane advantage is having a running total of your purchase, and drastically speeding up the final checkout procedure.

One could build a prototype system in a small amount of time (less than a person year of effort) to test out some of these ideas, without having to put them into practice in stores. For example, one could use a data base of credit card purchase records, from 1-12 months ago, to build up a user model, use it to predict purchases (or nonpurchases) of a particular new product this month, and compare the predictions with this month's receipts. That same sort of short-term "testbed" could be used for the application described in the next subsection as well.

4.2 Direct mail marketing

As above, but can be longer, glossier, and more detailed, and can include "products" not sold in stores (e.g., 900-... numbers.)

This could be quite effective for marketing new products. Dynamic, in-store marketing is bound to capture a very small sliver of the customer's attention, whereas direct mail can get his or her full attention, for a much longer time period.

New products are of course one area where alternative technologies such as those based on statistical analysis break down: by definition of "new," there aren't statistics available on who buys them, and preconceived market categories (about which there *are* statistics) are rather crude predictors and filters. To understand whether a person might or might not want new product X - e.g., a new type of detergent that is especially good on blood stains - might potentially draw on any part of your knowledge about people, devices, geography, illness, jobs, hobbies, and also on the functionality of the new product. One can either deal with each case idiosyncratically, by

hand, or use the crude market-categories approach, above, or draw on something with the breadth and depth of Cyc's KB.

Direct mail marketing is well suited for selling products or services which people are reticent to buy in a store (e.g., due to extreme modesty.) One extension of this would be to have customers pre-authorize (via credit card) the system to automatically select and ship items to them, "on approval," which the customer returns if s/he doesn't want them. One reason this is not in favor today - with buyers or with sellers - is that today's "shallow" direct mail targeting rarely has a success rate higher than 2%. But if the Cyc-based user-models are good enough at predicting who will want which products, the buyers will gradually gain confidence in this approach, and the "hit rate" will be high enough for sellers to operate this way.

So building and maintaining users' "profiles" - plus having them preauthorize this service by having their credit card number on file - might enable products to be automatically sent on approval to people who are likely to want them, saving them the time and in some cases embarrassment of explicitly having to order them. In the case of 900- numbers, the person's profile form might be used to pair them up with other callers, or (as Michael Lesk suggested) recommend other 900- numbers they might want to try, or match them with a particularly appropriate professional "speaker" or recorded message, or even - long before any AI passed the general Turing Test - *perform* adequately as that speaker. We can conceive of a wide range of 900- services of this sort, ranging from "gee-whiz-let's-talk-to-the-computer", to story-telling for children, to therapy and match-making and titillation for adults.

5. Online Brokering of Goods/Services

Using a common vocabulary and shared fundamental knowledge about transactions, enable a large set of buyers/sellers to find each other and negotiate deals - online and (semi-)automatically.

More generally, the problem being addressed here is inter- and intra- organization communication. Much of an organization's communication is on paper, in a form that is not machine-understandable.

Why is it desirable to make memos, announcements of meetings, purchase requisitions, etc. (more) machine-understandable? There are two reasons:

1. The information contained in the documents is more likely to be available to people who need to have it, when they need to have it.
2. Programs can take the information and combine it, compare it, reason with it, etc. This enables better automated decision support systems. E.g., one company's purchase orders could automatically be "read" by another company's software and, used in everything from its manufacturing equipment scheduler program, to its shipping label printing software, to its billing system.

One of the first steps to making communication machine-understandable is simply to make it online; e.g., to have employees make widespread use of electronic mail. The resultant text files are still not machine-understandable, however. Application programs, such as spreadsheets and calendar-arranging software, can't peruse the mail messages and run directly based on the contents of the messages.

Why not? Communication depends critically on having shared knowledge, between the parties communicating. The more knowledge in common that they possess, and assume the other possesses, the terser and more (superficially) sloppy or ambiguous the messages can be.

So the next step would seem to be to make available to the machine the same body of shared knowledge that the communicators assume that each other (their human targets) already possesses. That shared knowledge includes:

- (a) common sense about physics, politics, math, human nature, jobs, tools, etc.,
- (b) technical details shared by members of a particular profession,
- (c) context information, drawn from past messages, familiarity with this particular company, this particular person, past events, etc.

Without the shared knowledge, the messages will make no sense. The message writers won't - and probably couldn't, even if they tried - add to the message all the relevant common sense, technical definitions, and contextual details, so that it could become machine-understandable. Instead, the machine must already have that knowledge, if it is to understand the content of the message.

Here's another way to look at this. A machine can 'understand' only sentences in its vocabulary. To maximize machine-understandability, then you need (almost) unlimited vocabulary; hence Cyc.

But nothing like Cyc existed, so partial work-arounds had to be found. One partial solution has been to *structure* the communication as much as possible. Hence the widespread use of online *forms*. But this suffers from the problem that it severely restricts what can be said to - and understood by - the program.

For true machine-understandable communication to succeed, there should be few restrictions on what can be said. So it's no wonder that forms are used only for a small fraction of business communications; and that there is such a diversity and divergence of forms from one company to another, and even from one task to another.

Some work is underway in the PDES community, MADE-USA, and elsewhere, to bring more standardization of online forms. But these are just temporary fixes. The long-term solution must involve the use of dynamically tailored forms and, later, human-assisted natural language understanding (NLU) software and, eventually, completely unassisted NLU.

This application deals with the 'middle stage' of this evolutionary process: having structured online forms that are at least partially machine-understood.

The machine-understanding emerges from having a shared body of knowledge about common business objects and processes - purchase requisitions, inventory, usage agreements, etc., plus the default rules about them and interrelating them. The questions (line labels) on the form, and hopefully many of the answers, talk in these terms. Cyc provides the representation language for describing those objects, processes, and rules of thumb; the inference engines to reason about them efficiently; and the underlying general knowledge about physical objects, pieces of information, human beings, etc., that might also be required because (as mentioned above) it is part of the knowledge which the form-filling human beings presume that 'everyone' possesses.⁴

Programs could take the filled-in forms and use them for assorted decision making procedures. What should the machine try to do with these online forms, other than sending them from one place to another? For examples, look at any of the 'advice services' mentioned earlier, above. Any of them could be gotten off the ground sooner, using 'smart forms' as an interface. Or consider the brief example we gave at the beginning of this section: enable a large set of buyers/sellers to find each other and negotiate deals - online and (semi-)automatically.

Here are some types of forms that seem amenable to this approach:

- Legal documents. Naturally some of them are complicated, or contain idiosyncratic ties to "external" knowledge. But many follow standard forms already, and often whole pages full of clauses are standard boiler-plate, with at most a few blanks filled in. Deeds, wills, liens, leases, etc., come to mind. Today's online forms don't yet have quite the flexibility required, but Cyc could be applied (as discussed above) to provide that. Uses for this include faster and more accurate title searches, precedent-finding, and in general automatically answering common questions without always having to go through lawyers to do so.
- Purchase requisitions, and listings of goods/services offered. This obviously would enable the sort of match-making required to do automated bidding (and automatic vendor selection⁵), etc. It would enable "just *past* time" manufacturing and "just *in* time" delivery. One can imagine hooking this up to office or manufacturing machines which report on their own status (as high-end Kodak copiers do today, e.g.), and which in effect could then automatically place orders for their own routine supplies, issue work orders for their maintenance and repair, recommend their own replacements, and so on.
- User profiles. While not an ultimate solution to user modeling, "profiles" are a fast and powerful first step. The problem is that there is no standardization of these, let alone standard communication between them and application programs.
- Requisitions for service. This ranges from an individual consumer subscribing to a magazine or reporting a faulty appliance, through multiple corporations having a uniform

⁴ 'Everyone' meaning both the creators of the blank forms and the ultimate recipients/processers of the filled-in forms.

⁵ ISI's "FAST" system does this now, for a narrow class of electronic parts.

system for requesting and recording services to be performed, to one department requesting a service from another.

6. Data Base Cleaning

Relate DB fields and keys to Cyc predicates and terms, and use Cyc's common sense constraints to detect possible errors and inconsistencies in the DB.

One can think of a new generic family of data services, which we have clumped into one category: sanity checking on the data (pointing out unusual entries), diagnosing the reason for the error (typo, misunderstanding of fieldname, just plain out of date, missing some constraint, etc.), guessing at how to correct those errors, guessing the value for entries which are simply missing, noticing unusual patterns in the data, etc.

These services should probably be kept independent of the source of the data (i.e., the way in which the data came to be in the data bases: manual entry, automatic recordings from equipment, etc.)

There are literally millions of data bases in use today, so it's a high leverage idea to make these services be as generic as possible. I.e., we don't want to have to tailor the code, and modify the KB, for every data base that's to be handled in this fashion. *Some* of them may require detailed knowledge of the domain of the data base, but most of them require only knowing that, e.g., people in a certain type of job are generally paid twice a month; or that ``INCM" is filled with a person's annual gross income; or that ``AB" in this field of this table means that driver-side airbags come standard on this model of car - plus of course knowing things about income, years, cars,...

Let's go over that list of services, one at a time, giving examples of how Cyc could help. The next section (Section 7) will focus more on those applications that involve translating or combining information across multiple data bases; here (Sections 6.1 - 6.5) we focus on ways of using Cyc to process a single data base.

6.1 'Sanity Checking' of data

The recent US census lists one of the Austin neighborhoods as having 115 houses and 54 residents. Apparently a digit was dropped off the number of residents. Since Cyc knows that most houses have more than one resident, and almost all of them have at least one resident, it would be easy for Cyc to detect and flag this kind of error.

A person is listed as 18 years old, and having 11 children. Cyc knows that that's possible but unlikely. It's more likely that someone's finger slipped and hit the ``1" key twice, when they were typing in the number of children (1 kid instead of 11); or hit the ``1" key instead of the ``2" key when they were typing in the patient's age (28 years old, not 18.)

A field in one DB is named ``INCOME," and is meant to record the after-tax annual income of employees in a certain company. One employee's income is listed as \$30,000.00. This is so round a number that it's worth examining to see if their *pretax* figure was accidentally entered. Cyc can (doesn't currently) know that people's gross salaries are round numbers, that taxes take a percentage ``bite" off those and turn them into non-round numbers, etc.

Of course this needn't be limited to static data bases. The same idea applies to dynamic ``forms," such as purchase requisitions, as were discussed in the previous section. Here is a real-life example: A US Army engineer in Saigon filled in the ``number" blank on a requisition form with 1,000. Unfortunately, the item being ordered - telephone poles - had recently changed their ``unit of measure" from one pole to groups of 40. Cyc could infer this if it had information (not currently in it) about how much wire each pole uses, and it could note that the amount of wire being requisitioned was just enough for 1,000 poles, not 40,000; or through knowledge about what the poles were for (one army complex) and roughly how many should be required for that purpose; or through knowledge that the unit of measure changed recently, and it's a common mistake for orders to use the old units of measure for a while; etc. Incidentally, in real life, this was not caught, several freighters were diverted, and the 40,000 poles all showed up.

As another example, consider phone company workers who must take down information from customers (for phone installation, listings, etc.) Often, what they hear will be slightly ambiguous, and the ambiguities could often be resolved by accessing other information already present in the same data base (in this case, examining the rest of the White Pages to see how that name is usually spelled, or used to be spelled given their old phone number, or to see if the street name they heard exists the way they're spelling it, and whether that particular house number is reasonable given the other already-present numbers on that street, etc.)

6.2 Diagnosing the source of the error

Simple typo's are one common source. Models of the keyboard, conservation of the number of keys struck, common errors such as transposition of letters or numbers (especially if struck by different hands), etc., can all be brought to bear here.

A different sort of error is where the person entering the data misunderstood the fieldname, or tried to cram extra information into a field (something that wasn't allowed for in the schema) - e.g., they entered 17-18 where a single number was required, and the interface was ``smart" enough to do the arithmetic and simply record this as a -1. Or the example above, about the pretax versus after-tax income figure.

A third type of error is when the data is simply out of date. E.g., if the entry hasn't changed in over a year, and it's the sort of value that ought to change each year, (e.g., a person's salary, or age), then it probably just needs to be updated.

Conversely, if the unit of measure has changed (as in the telephone pole example), or some other aspect of the DB schema has changed, the person putting in the data might not be aware of that change.

Sometimes a constraint will be missing. E.g., a person's pretax income was updated, but their post-tax income wasn't. Knowing what the fields mean is all it takes to diagnose some of these errors - i.e., having the common sense to know that x depends on y.

Of course, if this is all one intends the 'watchdog' program to do, then a large fraction of it could probably be codified and programmed without recourse to something like Cyc. Why use Cyc, then?

Detecting certain errors might require having fairly broad, general knowledge (e.g., knowing how long various sorts of data are true, before they're likely to be out of date). Certainly, one could encode all this information in the watchdog program. But in the process of doing so, this watchdog program would be recreating some sizable fraction of Cyc. Further, there are many, many databases and it would be infeasible to write individual watchdog programs for each. A Cyc-based watchdog could be a fairly generic one, a watchdog that could work with almost any database.

6.3 Correcting those errors

Let's turn from diagnosis to treatment. Here are a few examples of how the sort of knowledge in Cyc could help correct some of the errors that were detected in a data base:

A DB says that someone lives in Virginia and works in Washington state. Presumably they meant Washington, D.C., unless there's strong evidence to the contrary (such as, they're an airline pilot, and the airline is based in Seattle.) Cyc knows that Americans today rarely commute more than an hour or two to work; commuting is usually done via autos, buses, trains, subways, bicycles, and walking; generally these vehicles travel at a speed under 100 miles per hour; and so on.

Suppose someone enters a monthly amount where they were supposed to enter an annual amount. If you know the number of months in a year (which Cyc does), you can correct that error, after having noticed it and having diagnosed it. Further use of Cyc's knowledge might be made to decide whether this person is likely to be paid only for part of the year, and if so for how many months (e.g., very short term IRS clerking, seasonal migrant farm labor, or nine months' per year of elementary school teaching.)

Consider the case of an error which was a valid entry once, but no longer is, due to a change in the data base schema. The solution might be, when the program suspects this to be the case, to try to interpret the data in terms of the old schema (e.g., assume they meant the old unit of measure for telephone poles) and see if that is then both legal and more "common-sensical."

6.4 Inferring absent entries

Suppose someone has 3 children listed as dependents for 1988 in a data base, and the same 3 in 1990, but the field is blank in 1989. Obviously, it's very likely that its value should be the same set of 3 dependents during the middle year.⁶

On the other hand, there are other attributes (e.g., deductions for political contributions, on IRS tax forms) where it's quite reasonable to have large expenses in 1988 and 1990, but none in 1989.

Simple statistical analysis won't suffice. E.g., there are many "common sense" reasons why the person in the above example might actually have had the 3 kids as dependents in 1988 and 1990, but not 1989. Perhaps he was in prison that year, or perhaps he was fighting for and eventually received custody of the children, following his divorce. Evidence for these could come from other data bases; we will defer discussing such inter-DB integration until Section 7.

Consider the case where there is an error in the data base because of a missing constraint. Fixing the error depends on knowing more about the constraint than just that one probably exists. E.g., you might know that a person's withholdings should change if they gain or lose a dependent. If the withholding stays the same, you've detected an error. To fix the error, though, you need to know the various IRS formulae.

There are numerous other rules - assertions in Cyc - that can come into play, that deal with common business practices (e.g., if person x works for person y, then x and y are likely in the same division of the company, they likely work in the same city, x's salary is probably lower than y's, etc.), human psychological and physical limitations (e.g., you can't be in two places at once), cultural facts and customs (e.g., how often elections are held), and so on.

6.5 Noticing patterns in the data

It is not uncommon for the schema associated with a databases to be available on-line. However, it is rare for the schema to be represented in a machine-understandable form. Moreover, and in many cases, employees who must use the DB do not have the ability to contact the designers of the database and ask them questions. Given the immense number of databases in use, it would therefore be most useful to have a program that could *induce* (or at least help in inducing) the schema associated with a database.

The process of guessing the schema would proceed by inducing/learning some of the constraints that hold between the different fields and mapping these constraints back into the constraints that hold between different attributes and slots in Cyc. By doing this, a mapping can be obtained between predicates in Cyc and fields in databases, i.e., for translating the schema into the Cyc vocabulary.

⁶ Since most data bases make a *closed world assumption*, they would interpret the absence of data to signify that the person did *not* have any dependents in 1989!

E.g., knowing that this is a DB about employees of a company, you'd expect one of the fields to be something like their ID numbers, another to deal with their age, another with their date or duration of employment, another with their supervisor, etc. ID numbers are usually either related to their 9-digit social security numbers (sometimes in the form nnn-nn-nnnn), or else are a separate numbering scheme; in the latter case, the numbers are likely to be sequential, with the employee's number increasing monotonically based on how recently they were hired, and in any case no two employees should have the same ID number. Armed with those rules, you can see how a program could look over the DB and make a guess as to which field is the ID number. Similarly for income, first and last names, address, job, supervisor, age, duration of employment, and so on.

Most of the necessary rules would apply across a wide segment of American business today, and many of them are even more general than that.

Once the patterns have been noticed, they can be treated almost the same as any other constraints; e.g., used to detect possibly-wrong entries in the DB. As we remarked earlier, the way that knowledge can be used and processed should, as much as possible, be independent of the source of the knowledge.

All the various statistical learning methods (including statistical analysis itself, neural net 'space-partitioning' algorithms, genetic learning algorithms) and symbolic learning methods (e.g., version spaces) can potentially be brought to bear on this problem.

Cyc has implemented some learning mechanisms, over the past two years, and we expect to enlarge that set of methods significantly in the coming two years.

7. Data Base Integration

Use that same sort of "articulation" approach to have several heterogeneous data bases all relate their contents to Cyc's KB.

The scheme we developed, in 1990, was to have a separate micro-theory in Cyc, corresponding to each data base. A set of "lifting" or "articulation" rules maps between tuples in that data base and more general CycL assertions in Cyc's KB. Cyc is therefore serving the role of an "interlingua." Instead of n^2 sets of rules, to interrelate n data bases, we need only n .

The previous subsection (Section 6.5) gave one glimpse of how such rules might be automatically induced, but for now just assume that they have somehow been acquired, either manually or automatically.

Actually, there are two sets of statements in Cyc for each data base: An object-level microtheory and a set of meta-level statements about the database itself.

The object-level micro-theory contains content which is equivalent to the tuples of the data base. Other Cyc micro-theories call on it when they want to know things stored in the data base. The

way we implemented this, e.g. in the automobile selection application, is to not actually ``lift" all the tuples explicitly from the DB and put them into CycL axiom form. Rather, they are all just left in the DB. When a request comes to the micro-theory for, e.g., the types of transmissions available on Dodge Dynasty LE's, the micro-theory generates an SQL-like query to the data base which it represents, gets the relevant tuples back, uses the articulation axioms to ``lift" them into CycL form, and returns those CycL expressions to the caller.

The meta-level micro-theory is the second set of assertions we enter into Cyc, about each data base. It contains such knowledge as where the DB is sparse, how costly it is to use it, how accurate it is in various ways (recency, precision, etc.), what the source for this information was (its ``pedigree"), alternative data bases to try under various circumstances, and so on.

There are two basic tasks which all this knowledge can be used for:

Inter-DB Data Cleaning

One use is to notice and fix conflicts between one DB and another. This is much like the previous applications described throughout Section 6 (cleaning of a single DB), but here we are detecting and resolving contradictions that occur across multiple data bases, even if each single DB is internally error-free and consistent.

One of the nightmares in many organizations is that data - a nontrivial share of the valuable assets of the organization - is inconsistent. Using Cyc to help fix this problem, within a single DB (as in the previous section) or across multiple DBs, is a no-lose bet.

Semantic Schema Integration

The other use is to answer queries that require obtaining - and integrating - several pieces of data from multiple data bases. Note that this use has to be tackled before the other use (inter-DB cleaning.)

MCC's CARNOT project is particularly interested in using Cyc for this purpose, for doing semantic schema integration.

The Car selection application mentioned in application (1), and detailed in Appendix B, uses the above approach for integrating information from a number of databases for determining which car a person should buy.

8. Corporate Knowledge Assets Management

Represent a company's know-how, policies, important documents, programs, and data bases - all in a form that a program can effectively reason with. Use as an online policy manual, or as a sort of ``smart Yellow Pages" for employees.

There are two different applications here waiting to be teased apart.

One is just representing - "knowing" - the policies, etc. This gave rise to the "online policy manual" mentioned above.

The other application is being the technical information server, request router, etc. This gave rise to the "smart Yellow Pages for employees" application mentioned above.

Cyc is useful for the expressiveness (and, less importantly in this case, the efficiency) of its representation language. The lasting value, though, is in harnessing Cyc's content, its KB, to this application. E.g., fielding questions which are not syntactically matched to the online material will require some understanding of the question, the questioner (and their situation), and the online policies.

In an earlier section we mentioned the importance of making a corporation's knowledge assets available on-line, in a machine-understandable form. However, realistically speaking, it is going to be many years for this dream to come true. The knowledge assets of most organizations today is in the form of manuals, technical reports, experiences and personnel. We would like a corporate information server to have these properties:

To begin with, it should know *about* the information a corporation has, i.e., it may know what knowledge assets the corporation has, where these are available, what they can be used for, etc., without necessarily understanding this information. There can then be a gradual move towards making more of this information machine-understandable.

However, even in the interim, the server can be very useful as both an archive and as a server for information. When someone has a problem (or needs some information), they approach the server. The server can then point them to the right documents, people, etc.

More specifically, the server should have three kinds of information:

- (a) Information about what documents the company has along with an outline of the contents of the document.
- (b) Information about the personnel the company has along with a *precis* of their areas of competence, interest, and experience.
- (c) Contextual information, about the company's purpose, organizational structure, its products, customers, suppliers, competitors, and regulators.

Given a problem/request for information on a particular topic, the server would produce an "Extended table of contents" which would include portions of documents, pointers to individuals in the corporation, and so on. The approach here is one of *partial understanding* of the documents, based on which extremely accurate retrieval can be done.

Why would Cyc be needed for this application? The answer, as with many of the above applications is that even partial understanding - in a potentially unrestricted domain - requires a

system `primed' with a significant fraction of ``common knowledge." Today, and for the foreseeable future, that means Cyc.

One example is a query of the form ``What were some bad stock market investment recommendations that were published in the corporate newsletter this year?"⁷ Current text retrieval technology (transforming and simplifying the syntax, substituting synonyms and generalizations and specializations, etc.) won't get the right answer, because no articles are going to say ``Here are some bad recommendations..." or anything equivalent to that. But just a little understanding (about recommendations, human goals and desires, investing in stocks, etc.) is enough to dig out the desired articles.

Another example is a query of the form ``Is it okay to send a copy of our project plan to someone in another Division?" Current text retrieval techniques could result in massive amounts of irrelevant information about your project, the other Division, etc. The relevant articles are *not* likely going to mention either of them specifically, let alone both of them together. A little understanding could direct the search for relevant corporate policy memos, people who are expert in this area, similar cases in the past (and people who were involved in them), and so on.

9. Smart Spreadsheets

Explain the ``meaning" of the rows and columns, by tying them to a large corpus of general knowledge about occupations, human capabilities, goals, household objects, etc. Highlight abnormal (though not illegal) values, and suggest places where there seems to be a missing equation or constraint.

Consider a spreadsheet program, pertaining to a school district's budget, being run on a computer. The machine has some level of understanding of the school district data, in the form of a set of equations and constraints. It can use them to manipulate, modify and fill in some values, in the spreadsheet.

But in a deeper sense, it does not have any idea of what these rows and columns - and corresponding `cells' in the spreadsheet - actually denote. Though the *human user* of the spreadsheet may associate a certain kind of attribute (such `expenses', `revenue', etc.) with each column in a spreadsheet, and certain objects with each row (such as individual employees of the school district) the *program* itself does not understand this. E.g., you could make absurd statements about the employees, and if there weren't some constraint/equation given to the spreadsheet program, it would have no inkling that anything was amiss.

The obvious problem is that it does not have the vocabulary⁸ to do this for a wide range of domains. This is where Cyc comes in.

⁷ In both this and the next example, we'll ignore the issues of natural language understanding, and assume that the user has successfully squeezed his/her query into some designated SQL-like query syntax.

⁸ Nor understanding of the vocabulary, even if the user invented a vocabulary

Each column in the spreadsheet would be linked to a certain Cyc slot and each row to some object. Updates in the spreadsheet could be reflected as updates in the corresponding Cyc slot values, either immediately or asynchronously.

Some of the services that could be provided by adding that sort of Cyc-based ``rear end" to a spreadsheet are:

- Point out ``interesting" or ``anomalous" entries. Determination of what is unusual could be done based on comparison with Cyc's default predictions.
- Examine the constraints between different variables and look for missing exogenous variables, suggest new constraints, etc.
- Perform ``Data Cleaning" (of the sort mentioned in an earlier section) on data entered by the user. This would be especially useful if the spreadsheet were hooked up to a ``dirty" database (which of course most large real-world DBs are.)

10. Smarter Interfaces

Use an online model of the ``meaning" of what the application program is doing, what the user's goals are, what constraints they're under (e.g., deadlines), etc., to adjust the interface. E.g., reorder questions on a `form'; rearrange windows; highlight or cross out various parts; guess at values to fill in certain `blanks;' and so on.

We're talking here about more *dynamic* forms. Innumerable forms are filled out each day, and most of the time the form itself is rather rigid and static, even if it is being displayed on a VDT. Our experiences filling out income tax forms for the IRS suggests a solution, in the way that certain answers to certain ``blanks" on a tax form cause entire new schedules to be required. Imagine generalizing this, to active forms which can be filled out in any order, and which, depending on what you've already filled out, might add or remove or highlight certain questions, and make guesses at answers to other questions.

As a first step, Cyc could act as an intelligent front end to existing performance programs such as MacinTax.

There are two modes in which the interface may use Cyc. In the `running hot' mode, Cyc would be part of the interface, performing inferences on the entries already made by the user, guessing at what other fields become meaningful to the user, etc. In the `cold' mode, Cyc could be used at interface development time to determine the different paths which might be taken during the process of the form being built. The information in Cyc used during this process could then be `extracted' and included as part of the interface.

The `branchiness' and `open-endedness' of the form would determine which of these two modes is to be used. If the number and scope of potential changes/modifications to the form (in response to additions by the filler of the form) is very large, it would be necessary to incorporate Cyc `hot,' as part of the interface.

This choice - of incorporating into the application all of Cyc's KB versus a small part of it - is available in most of the applications described in this document. The factors involved in making this choice include the fraction of the KB potentially used by the application, the kind of platforms the application is expected to run on, what sorts of time delays are permissible, the amount of time available in which the application would otherwise just be sitting idly waiting for input, etc.

One way to make this choice is empirically, by starting with all of Cyc present, and using its various tracing mechanisms to keep track of the parts of Cyc that were used. If not too large a fraction is being used, and very short realtime response is required, etc., then it would be worth the effort to extract the heavily-used parts of the Cyc KB and recode them as part of the application.

One more aspect of using Cyc for "smarter interfaces" is the following. A person or organization could have its publicly available information in a file in a standard language, e.g. CycL. A program belonging to another organization could read this information and fill in its own forms. Or, that other program could make use of the user's publicly available "model" to reconfigure its interface specially for that user, to make itself more familiar to (and hence usable by) them.

11. Machine Translation of Technical Documents

Provide a set of templates into which 95% of some class of technical documents (e.g., installation instructions for a laser printer) can be captured. Using natural language software akin to Jacobs and Rau's at General Electric, process such documents, converting them into a set of filled template instances. Conversely, the 'first writing' of a manual could itself be done by filling out such templates. Translations into different languages could then be produced by composing the translations of each template, and each (disambiguated) term used. Difficult portions get flagged for checking by hand.

An *open ended* machine translation program - one which can take an arbitrary text in one language and convert it into a text in a different language - is still quite far away (certainly outside the 5 year scope of this document). But most of the money spent today on machine (and human) translation is spent on a very restricted class of documents such as technical manuals. Based on this, we believe it should be possible to produce programs for handling just this class of documents. The key is to exploit the fact that these documents have a very restricted structure, and that the translation does not have to be "literary quality".

The basic approach is as follows. For each restricted class of documents (e.g., installation instructions for printers), a template/form is created. We're using "template" loosely here; in actuality, each template would be a first-class object in Cyc, interconnected through various predicates with many other Cyc objects: sub-templates, assertions being made, the context of the text, and so on.

The document is represented using this template. Most likely, an "active form" interface of the sort described in Section 10 would be appropriate to facilitate this process. Using current language generation technology, the manuals in different languages would be generated from the template complex.

The quality of the prose in these documents might be rather low, but most of that problem could be corrected by a monolingual speaker of the target language. The chief savings come from not requiring much involvement from individuals who are both technical and polyglots.

The template itself could be generated in one of two ways:

- It could be filled in by using a text understanding program (such as the GE one). As the DARPA-sponsored MUC's (Message Understanding Competitions) have demonstrated, such programs can be tuned to be quite good at the task of taking a large but fixed set of questions, and a huge set of messages, and extracting the answers to those questions for each of the messages. I.e., filling out a template based on the text.
- It could be manually filled in by the user. I.e., the document is "written" by a user by filling out the template. Not only would this facilitate translation into foreign languages, it might even take less time in one language (e.g., less time to fill in the template, and let the program generate an english version of it, than to write the document in english.) Also, this opens up the possibility of *english-to-english* (for example) translation: producing customized translations for individuals and groups, based on what vocabulary they do and don't know, concepts and details they can be expected to already know or not know, etc.

There are two hypotheses around which the success of this application hinges. One is that at least 95% of the information in most manuals can be captured in specialized templates. One way to make the success of this application less dependent on the truth of this hypothesis would be to incorporate some mechanism for information not storable in the template to be incorporated as raw text. Of course, humans would then be required for translating those pieces of text. However, if 95% or more of the information can be captured in the template, there would still be very significant reductions in the cost and speed of producing translations.

The other factor on which this application depends is the ability of current NL generation technology to generate prose from the template. We hypothesize that very simple natural language generation strategies should be sufficient. Why? For one thing, the literary quality of the class of documents being dealt with - technical manuals - need not be very high. Secondly, these documents have very constrained organizations, which itself could be specified as a 'control template' or 'table of contents'. Thirdly, there are numerous examples of online manuals which, once 'templated', could serve as cases to analogize to; this case library would grow as more and more documents were translated and vetted. Fourthly, it would be acceptable for humans - mostly monolingual ones - to go over the translation; and the need for this should significantly go down after a year or two of fine tuning of the generation program.

12. Enriched `Artificial Reality'

Simulated `worlds' in which physical objects behave ``realistically," and machine-run personae behave ``intelligently," particularly when one interacts with them in ways that were not specifically preconceived. In addition, complex, changing images must somehow be transmitted in real time, among a large set of interacting human users, at compression ratios exceeding one million to one.

There are two major bottleneck problems, today, to achieving many Artificial Reality applications:⁹

Realism.

As one encounters objects in the simulated world, they ought to behave realistically. For example, I should be able to write with my simulated pencil, on my simulated pad of paper. The problem is not just how simple physical objects should behave more ``realistically," following the laws of gravity, exhibiting friction, and so forth. Nor how they should exhibit their *obvious* functionality (e.g., the pencil and paper.) The problem involves having the objects be modeled in enough depth so that the user can employ them in novel ways - ways that weren't anticipated by the designer of the AR software. For instance, I might want to use my pencil as a bookmark, or as a straight-edge, or to tap against my glass to make a ringing noise, or as a weapon, or as kindling, or as part of a crude protractor to compare two angles, or as a pointer, and so on.

This requirement is even harder to satisfy when some of the objects are animate (e.g., fish or other animals), and harder still if some of them are intelligent beings - i.e., simulated people who are ``in the same world" (e.g., at the same simulated conference table as you.) Even harder than that is when these people are fully machine-run personae, not just icons for human-controlled ``participants." At that point, the problem becomes AI-complete; to be truly realistic, the machine-run people would have to act and react and converse intelligently with the ``real" human beings. The question, then, is how far along this spectrum we can go, and how will Cyc help move us farther along it. We discuss that below.

Semantic Image Compression/Synthesis.

In many AR applications, there are several individuals, sometimes thousands, situated at geographically distributed locations (sometimes, as in SIMNET, at nodes all over the world), who are participating in the same scenario. E.g., they're in the same simulated conference room, amusement park, library, battlefield, or whatever. They will want a

⁹ Artificial Reality (AR) encompasses a bundle of applications, generally of the following form: you, the user, are in a simulated world. The objects you see, the actions you do, etc. are metaphorically (often ironically) mapped to those of some particular task. In the simplest case, this mapping is just a 'depicting', in which a gear is represented by an image of a gear, a person by an image of a person, and so on.

high quality image of the simulated world; that means, as others move around and do things in the world, they will want to see those movements and the effects of the actions - probably in real time.

For a long time to come, the bottleneck will be the communication bandwidth or, equivalently, the cost of "being there" in the AR world. One solution is to transmit terse linguistic descriptions of situations - in the SIMNET case, a vector of each tank's location and state variables - rather than transmitting massive amounts of pixel-level information. At each user's terminal, then, these descriptions are expanded back into images. The images won't agree completely, at the pixel level, but should be semantically equivalent to the "real situation." E.g., one message might say that Tank404 is on fire. Everyone who can see it, in the simulated world, would then have animated flames flickering around it. The exact size and shape of the fire, the instant that a flame went up instead of down, etc., would not be the same from one viewer to another. But they would never know that, they would never *need* to know things at that level of detail, given their task. The synthesized image is 100% *semantically adequate* for their purpose.

Cyc is relevant to each of these tasks. Consider the first one: adding realism. What does it mean for an object to be used in an unexpected way? It means that you are bypassing the normal, specialized, scripted use of the thing, and drawing instead on more general properties that it has - size, shape, mass, opacity, hardness, wetness, etc. - and comparing those to the requirements for serving as, e.g., a bookmark. But that sort of knowledge is just some of what we're putting into Cyc, about tangible objects.

For having realistic behavior from artificial personae, Cyc is even more necessary. E.g., they should act as though they have common sense, hence they should act as though they know all the material in Cyc.¹⁰ Moreover, Cyc can predict what sorts of specialized knowledge they ought or ought not to have. Most Americans over 16 know how to drive, most kids don't; most physicians know a lot about medicine, and are not very likely to be superstitious or illiterate or poor; and so forth.

Now let's turn to the second bottleneck problem. To consider applying Cyc to semantic image processing, first consider the more restricted task of image compression. Traditional - and current - techniques use fractals, wavelets, etc., and (in cases of continuous video) achieve compression ratios of "a few tens to 1," today, perhaps as high as 100:1. Now contrast that with the compression ratios achieved in going between, say, a two-hour movie and a screenplay or novel version of it. The latter are on the order of a few tens of *million* to 1. I.e., a million times more compressed than we can currently achieve, numerically. And yet the subjective experience, and knowledge learned, from reading the novel often equals or exceeds that obtained from watching the movie. How is that possible? It has to do with all the shared knowledge in the head

¹⁰ Excluding of course any expert-level knowledge which may happen to be present in Cyc's KB, due to some other applications having been built as permanent "pseudopods" to Cyc.

of the author and the viewer. So the author can mention a short phrase, and the reader can imagine, in their mind's eye, a detailed image of what is being described.

Another common case of compression is political cartoons. They have something important to teach us about the storage and communication of meaningful visual information. Cartoonists have learned how to distill the "essence" of something in a few bits. Exaggeration plays a critical role, but it is a very controlled exaggeration. A serious study of political cartoons could be very fruitful; from it we might learn how to augment or *automate* cartoon production for instructions, directions, synopses, etc. This would of course have implications for many other applications (e.g., converting templates into illustrations as well as prose for technical manuals, the application we discussed in Section 11.)

Although communication speeds keep increasing, and costs declining, nevertheless the overall volume keeps increasing as well. So the overall economic benefit to having many orders of magnitude compression will be high for at least the next several decades.

By partially understanding an image, then, human beings can "parse" it - into, e.g., a set of key objects, some attributes of the objects, important relations among the objects, and so on.¹¹ Such a "summary" of the image might be expressed in logic, natural language, or even in some simpler image form such as cartoons or icons or even just a "cropped" version of the original image.

The right level of summarization depends on how faithfully the images need to be reconstructed later - e.g., maybe a screenplay summary of a movie is good enough (with suitable models for the various actors and objects in the scenes, models of how they move, etc.) and maybe it's not. It depends on the purpose to which it is later going to be put. One use of Cyc might be to help decide what granularity and modality (text, sketches, etc.) is most appropriate and cost-effective for summarizing an image.

The choice of granularity need not be uniform across the image. E.g., in an image involving people, it's usually more important to get the faces right than the hands, which in turn are usually more important than the clothes, which are usually more important than the background, etc. Given an image, and a fixed number of bits in which to encode it, it is therefore suboptimal to simply spend those bits on a uniform rectangular raster of sampled points, and it is suboptimal to spend them based on the variance of light/darkness in the image (e.g., suppose the people are wearing "busy" patterns on their clothing.) Just as happens in the human eye, we could concentrate the bits - and the ultimate locations of high resolution - at those places where it's

¹¹ We do not foresee a short term application involving "real" image parsing, however; there are just too many difficult problems to solve (esp. low-level perceptual ones). Nevertheless, let's continue with this line of reasoning, and we will come across several shorter-term applications that rely only on image generation, not image understanding.

important (given the viewer's purpose) to do so. What gets transmitted to the user's workstation, then, is both a "map" of this distorted, homoncular sampling, and then the bits themselves.¹²

The basic idea here is for each user's "node" to maintain a model of the simulated world: the locations, orientations, states, actions, appearance, etc., of the various objects and actors in that world. Then, as they do things (move around, shoot weapons, whatever), they broadcast only a terse description of what's going on.

Our claim is that it would help to have Cyc be present at the "nodes," serving as some of the shared knowledge located there. Let's consider some examples.

Suppose we say "Fred sits down at his desk and phones Sally." What knowledge do you - or a program - need, to produce an image of that? You need to know what people (and in particular, Fred) and desks and telephones look like, that desks usually have chairs which is how people sit "at" a desk, how big all these things are (both absolutely, and compared to each other), the relative orientation of these objects to each other (e.g., the chair to the desk, the desk phone to the desk top), what objects are likely to be on the desk, what other things (windows, doors, scenes outside the window, bookshelves, etc.) are likely to be present in the room, the fact that the chair must have been empty, there must be some light source for Fred to see to dial, people sit down by bending... well, you get the idea. Once again, this is all more or less knowledge that is (or should be) in Cyc. Also important is script-like knowledge: action sequences for people entering a room, walking, sitting down, dialing a phone, and so forth.

In some cases, the program may already have a pretty good idea of what is being portrayed - and where - in the image. This might happen if, e.g., the current image is one of a sequence, such as the temporally adjacent frames of a motion picture. In those cases, there are many physical (and other) constraints on objects: how rapidly they can move, accelerate, and jerk; which objects are opaque; which ones are solid; what "supports" relationships are and are not in effect, etc. Of course the same knowledge can be used, in reverse, during image synthesis.

Much of the imaging, image synthesis, and image transmission, of the coming decade or two will be interactive. I.e., the viewer will see some scene or display, and then take some action which will effect (or even fully determine) what the next image is. E.g., maybe they're in a simulated world, and they decide to move into a different "room" or just turn their head in a certain different direction. Or perhaps they're shopping, and based on the item they've just seen, they'd like to see one in a different color, size, price, etc. Cyc could help do a kind of *anticipatory* image preparation. I.e., based on the current situation, recent past, models of the user and the domain, etc., predict the most likely information that is going to be needed to image next. E.g., if they hear the door open to their left, they're likely to look over that way. If the item they're examining is close to what they want but outside their budget, they're likely to ask to see similar but less expensive ones. And so on.

¹² Of course, numerical methods can then be applied to further compress this overall message – the foveal technique is more or less independent of whether other compression algorithms are then applied to the distorted image.

The above focuses on the visual, but clearly generalizes to other modalities. In the coming five years this is likely to be mostly oral (verbal voiceover, recorded "real" sounds), until there are good I/O devices for touch and taste and smell.

One of the many AR applications that Cyc might revolutionize is computer games. Both the "added realism" and the "effectively enormous image compression" aspects of Cyc come into play here. It might lead one day to a new hybrid form of entertainment, combining aspects of (and eventually replacing, in some areas) television, movies, interactive computer role-playing and adventure games, information browsing, multi-person team sports, social gatherings, etc.

Another area of AR that Cyc might improve is 'civilian wargames.' These are simulations of economics, factories, police, law, real estate, politics, etc., which are made more realistic (because of Cyc's KB - as described above.) One appropriate use of such programs would be to improve training, by allowing a much broader range of "what-if" scenarios to be explored by trainees. Such scenario generation might be partially or fully automated, allowing populations of scenarios to be analyzed for unsuspected patterns. E.g., there might be one outcome which appears intuitively unlikely, because no "very probable" scenario leads to it, but it so happens to be the result of many different low-probability paths, making it in aggregate worth attending to.

13. Conclusion

We expect Cyc to have a significant impact on everyday life, and on business. As with almost all broad, powerful technologies, some of those effects will be positive and some - hopefully fewer - will be negative.

It has not been the aim of this paper to moralize, to advocate only what we feel are the "positive" applications. As a result, we have presented some of the applications in what may seem a rather cold-blooded fashion.

Yes, some of the projects we've discussed in this paper are utopian, and some are dystopian. But it would be naive (not to mention bad business) to pretend that the distasteful applications will not be commercialized right along with the tasteful ones very soon after they become technically feasible.

E.g., consider using Cyc to prepare powerful individualized arguments to sell a product to a consumer. In cases where buyers realistically perceive their own needs and resources, it will be doing a positive service, matching them up more accurately with goods and services which suit them. In cases where the "argument" plays on a buyer's fears, guilt, frustrations, etc., to make a sale, it may not be doing them any favor. We expect that, just as with "personalized" direct mail letters twenty years ago, the public will soon build up defenses against this latter, shallow sort of model-based targeting and argumentation-based selling. There will probably be a multi-year window of opportunity in which it will be quite effective. Hence profitable.

So, what makes for a good Cyc application?

- **Low infrastructure cost.** Until Cyc has proven itself in other areas, it would be unwise to commit to a Cyc-based application that required an enormous capitalization or retooling. E.g., using Cyc in classrooms, which might be great, but would require each classroom to have a computer running Cyc.
- **Incremental use of Cyc.** A fail-soft property: if Cyc can recommend something, great. Otherwise, the program carries on much as the existing software would, for that task.
- **Off-line.** Choosing an application where rapid real-time response is required, is just creating one more obstacle to having a Cyc-based application succeed.
- **Moderately 'stylized' use.** That is, the application program's use of Cyc should be characterizable by some small number of high-frequency query types, which can be optimized by adding one or more new modules to the Heuristic Level (and adding the corresponding new rules to the EL-to-HL Translator.)
- **Need for common sense.** There may be little need for Cyc, if the task is very technical and/or very narrow and/or very simple, and/or very highly stylized. In such a case, it might suffice to use an expert system, or a non-AI application program.
- **Centralized running.** Ideally, the application should only have to run on one centralized machine (e.g., one machine running Cyc in each supermarket.) If the end-user devices must each run Cyc, that would eliminate a vast fraction of the market for a decade; namely, businesses built around low-end (as of late 1991) PC's.

All ten applications satisfy *most* of the criteria to some extent, and contain the potential for tremendous business impact. Not all ten applications satisfy all of these criteria. To the extent that they do satisfy them, they are more appropriate uses of Cyc, more short-term, and/or more likely to succeed as profitable Cyc-based products.

Acknowledgements

We wish to thank the Cyc group for building Cyc. As regards this paper, we appreciate all the editorial help Wanda Pratt and Mary Shepherd have provided. We thank Craig Fields, Alan Kay, Michael Lesk, John McCarthy, Tom Murphy, Bob Simpson, Marvin Weinberger, and especially Steve Chenoweth, for useful comments and for meeting with us to discuss various application ideas.

APPENDIX A: The 3 Ways that CYC Can Help

Here are three ways in which Cyc can empower the aforementioned ten applications. Other AI systems (e.g., EDR, LOOM, PENMAN, PRODIGY, KIF, ONTOLINGUA, etc.) have advocates who might object to the first two claims: Cyc's superiority in regard to expressiveness and efficiency. But we expect they would at least acknowledge the third claim, regarding Cyc's organization, size, and breadth. A company might devote adequate resources to "fast-follow" Cyc on the first two, but even a monumental effort would be likely to fail, or slowly follow at best, on surpassing Cyc on its final claim, *content*.¹³

1. CycL is extraordinarily expressive. It includes reification (so you can make statements about other statements), reflection (so you can make statements about what sorts of progress or blind alleys have been pursued so far in trying to solve a particular problem), modals (so you can talk about things that one believes, or expects, or desires, or intends, or fears, etc., and nestings of such), default reasoning (based on assembling pro- and con-arguments and comparing them), set operators, contexts (what we often refer to as "microtheories"), disjunctions, negations, nonconjunctive capability (e.g., a car can turn left or right at a corner, but not both), and much more.

It should be no harder to state to a machine than to a person sentences such as: "I want to buy a car that's either a fast red convertible or else is both very safe and causes most people to believe it is much more expensive than it really is." CycL does this. This is not too surprising, if you think about it, in that CycL evolved from literally millions of small cases; as we came across things which were impossible or awkward to state, CycL was enhanced a bit to handle that.

2. Cyc can efficiently do inference. This is due to its having dozens of special-purpose inference engines, so it rarely needs to fall back on general - and inefficient - methods. In particular, it can efficiently¹⁴ retract assertions. This is important in an application where one adds some constraints, examines the resulting set of candidates and may (or *must*, if that set is null) choose to relax or completely remove some of them. Included here is CycL's special ability to efficiently detect and expunge self-justifying cycles - unwanted remnants that are found either inefficiently by some truth maintenance algorithms (e.g., JTMS), or at a potentially exponentially explosive space cost by others (e.g., ATMS.)
3. Cyc's KB is already enormous and broad. Sometime during 1990, it crossed the one million mark, as far as how many "rules" it contains, and the 10,000 mark as far as how many predicates (unary collections, binary slots, ternary predicates, etc.) More than size, however, *breadth* is important, and Cyc's KB spans human consensus reality. When considering "where the power comes from," though, *completeness* of coverage is even

¹³ This is discussed more fully in the MCC Tech Report "Comparing Cyc to other AI Systems," Guha & Lenat, issued simultaneously with this one. This appendix is excerpted from that document.

¹⁴ That efficiency, by the way, is due to its having dozens of special-purpose truth maintenance modules, rather than none general one (or none at all), as all other systems have.

more important than sheer size or breadth. Data bases can be immense, e.g., and encyclopedias broad, but neither captures the knowledge (or even the *type* of knowledge) currently in Cyc, knowledge which we believe will be useful in the ten applications listed above. Of course Cyc's KB isn't complete down to the level of expert knowledge, but we present the following empirical claim: The most general levels of Cyc's ontology are by now more or less complete.¹⁵ I.e., it is very rare nowadays for us to have to make any changes, even small ones, in the content or organization of the most general 100,000 assertions in the KB, what many people refer to as its ``crown."

APPENDIX B: The `selecting a new car' Application

This appendix expands on, and supplements, the material presented in section 3, Advice Services. We give some further details of the Cyc-based application which was built in the spring of 1991, to help a person select which new '91 car to buy. A much more complete account of that application, including scripts of running it, traces of the inference chains it draws, etc., is available now in R. V. Guha's just-finished thesis (December, 1991), which is being issued as an MCC tech report.

The prospective car buyer has a `form' to fill out - a list of questions which may be answered in any order. Cyc is used as follows:

- Where possible, Cyc reports on legal answers, and the user can then just select one of these from a menu. E.g., many of the questions are fill-in-the-blank sort, such as ``What is your occupation?" Cyc accesses all the instances of WorkerType it knows about, and lists these as possible answers. (As of the writing of this paper, there happen to be 310 such occupations known to Cyc.)
- As various things become known about the buyer and the car, new questions get added, and old ones get removed (or answers to them are guessed at.)
- Gradually, the list of new car models shrinks from hundreds to a small number. Often, the user will overconstrain things, and the list will become empty. At that point, they can just point to one of the constraints currently in place, and relax or remove it. Cyc's truth maintenance facility efficiently retracts it, and the list of cars still in the running grows.
- As questions get answered, Cyc infers new things about the buyer, and, hence, infers new constraints on what sort of car they'll want. E.g., if they're a Real Estate Agent, they're much more likely to want a luxurious car that seats 4-5 in comfort, even if they live alone. If their hobbies include boating, the car may need to be able to pull their boat. And so on. In other words, much of the needed information is guessed at by Cyc, rather than having to be explicitly told it by the buyer.

¹⁵ As an anecdotal example, Raj Reddy visited here last week, and we made this claim to him. He asked s to tell Cyc "Most marriages in India are arranged by the parents." This is not the sort of thing we had entered into Cyc, yet. But Cyc already knew about the act of getting married; parents; humans arranging and controlling others' actions; India; etc. So we were able to easily enter this default rule, without creating new instance or collection or predicate terms, or adding "surrounding" rules, or new methods, or reorganizing the KB, etc.

Here is a longer inference chain. Suppose the buyer tells Cyc that they are retired and live with their wife. Cyc knows that since they live with their wife, they're male. Since he's retired, he's probably 65 or older. In the US today, most husbands are usually around the same age as, or a few years older than, their wives. So his wife is probably at least 55. Their kids are probably all of driving age, and most likely do not live at home any more. Hence, regardless how many kids there are, they won't need to be chauffeured around in the car. So what? So the question "How many children do you have?" - which the buyer has not yet answered - is almost certainly irrelevant, hence can be removed, or at least given a lower priority place on the ordered list of questions for the buyer to attend to.

- As new information comes in - either by being told or by inference - there are sometimes contradictions or suspicious values in that data. E.g., someone who says they're single but elsewhere says that they live with their wife. Or someone who says they're 18 but have 11 kids. Sure, it's biologically possible, but it's more likely that there was a typo, and the person meant 1 kid but their finger slipped and they hit the "1" key twice, or they mistyped their age, or they misread one of the questions, etc. Or someone who says they work in Texas but live in Georgia. But if they're a flight attendant, maybe it's not so improbable. You get the idea. Knowing when to be suspicious, and when not to, potentially could draw on almost any aspect of Cyc's large knowledge base.