Appendix A

Applications of CYC

The development of CYC was a very long–term, high–risk gamble that has begun to pay off. Started in 1984, CYC is now a working technology with applications to many real–world business problems. CYC's vast knowledge base enables it to perform well at tasks that are beyond the capabilities of other software technologies.

This appendix indicates CYC's potential by describing some tasks to which CYC is currently being applied and by suggesting future applications.

Applications Currently Available or in Development

- Natural-Language Processing
- Integration of Heterogeneous Databases
- Knowledge–Enhanced Retrieval of Captioned Information
- Thesaurus Management
- Distributed AI
- WWW Information Retrieval

Potential Applications

- Online brokering of goods and services
- "Smart" interfaces
- Intelligent character simulation for games
- Enhanced virtual reality
- Improved machine translation
- WWW Information Retrieval

- Improved speech recognition
- Sophisticated user modelling
- Semantic data mining

A.1 Natural–Language Processing

As seen in lectures, NL understanding needs CYC–like common sense is a prerequisite for human–level competence at this task.

Consider the following pair of sentences:

1. Peter saw the plane flying over Zurich.
2. Peter saw the mountains flying over Zurich.

Traditional NL systems will have difficulty resolving this syntactic ambiguity, but because CYC knows that planes fly and mountains do not, it will be able to parse these sentences just as easily as a human. It's difficult to see how this could be done without relying on a large database of common sense.

Here are a couple more examples; these involve pronoun disambiguation:

1. The police arrested the demonstrators because *they* feared violence.
2. The police arrested the demonstrators because *they* advocated violence.

1. Mary saw the dog in the store window and wanted *it*.
2. Mary saw the dog in the store window and pressed her nose up against *it*.

The CYC–NL system has three components: the lexicon, the syntactic parser, and the semantic interpreter. Currently it is possible to correctly parse many different sentence types, including ambiguous and syntactically complex inputs. CYC is capable of handling negation, modals, and nested quantifiers. Interfaces are being developed to make assertions and query CYC using English instead of *CycL* together with a generation component, which will produce english strings from CycL.

Future directions for CYC–NL will include:

- exploring the role CYC could play in machine translation
- using CYC–NL to post–process output of speech recognition systems
- harnessing CYC–NL to enhance user interfaces.

A.2 Integration of Heterogeneous Databases and Data Mining

Current database systems are information–rich but knowledge–poor. They have a very flat structure that incorporates little or no semantic–level knowledge. A personnel database may know that "John Tucker" is a"professor" who works for "Swansea University", but it does not know what these strings mean, what a person is, what a professor is, or what a university is. CYC, however, does know!

To provide everyday knowledge about the world, CYC is given a description of the content of each database to be used, using *CycL*. For example, in a table, the field CITY contains text strings which can be mapped directly to instances of the CYC term `$City`, like `$NewYorkCity`. Once you have the description of the database in the CYC knowledge base, you can use that knowledge in all sorts of ways.

Let's say, for example, that we have two tables. The first is a table of people with three fields: the person's name, the person's job title, and the name of the person's employer. The second is a table of employers with two fields: the employer's name and the name of the state where the employer is located.

Now imagine that we want to use these two tables to answer the following query: "Show me people who hold an advanced degree and live in South Wales." But wait! The tables don't say anything about degrees people hold or where they live, and they don't mention New England. But CYC knows that doctors, lawyers, and professors hold advanced degrees, that people generally live near their place of work, and that New England is comprised of six specific states. So CYC converts this query into a query for doctors, lawyers, and professors whose employer is located in South Wales. CYC automatically generates the necessary SQL, talks to the relevant databases, and converts the results back into *CycL* where appropriate. If John Tucker is listed a professor working at Swansea, then he will be found to satisfy the original query.

The only way to get the same result using existing database tools would be to formulate the SQL query by hand, using your *own* knowledge of both the world and, more importantly, of the structure and content of the available tables. With CYC, on the other hand, you can issue a query in plain english (using CYC's NL capabilities), and you don't need to know anything about the tables, or even which tables are available. The process is fully automatic.

CYC's database tools handle a query in three phases: the interface phase, the planner phase, and the executor phase.

In the interface phase, the user constructs a query using an interface tool. In some versions of the database application, the interface is a simple template form which the user fills out and submits; in others, the user enters a plain english query, which is converted by CYC's NL module into a *CycL* expression with free variables. So, for example, "Professors living in South Wales" might get converted to

```
($and
   ($isa ?x $Professor)
```

```
   ($residesInGeographicRegion ?x $SouthWales-UKRegion))
```

In the planner phase, the CYC inference engine uses assertions in the knowledge base to backchain from that expression to expressions which refer to descriptions of database columns. One such expression might be:

```
($and
   ($isa ?x $Professor)
   ($employees ?y ?x)
   ($residenceOfOrganization ?y $WestGlamorgan))
```

Next, these expressions are converted into a intermediate representation format called CSQL, which represents logical database queries at a high level (that is, without reference to the specific location or format of the underlying data).

In the executor phase, CYC uses knowledge about the physical aspects of the underlying databases to convert the (logical) CSQL queries into (physical) SQL queries. It is at this stage, for example, that logical terms (such as `$WestGlamorgan`) are translated into terms appropriate for the specific database (such as `WGlam` or whatever).

When the results of the SQL query come back from the database, CYC translates objects it knows about back into *CycL* (e.g `WGlam` back into `$WestGlamorgan`) and leaves rest as raw data.

During 1995, Cycorp developed an application along the lines described above for a major pharmaceuticals company. The application is used to integrate tens of gigabytes of data from over 60 tables. The databases used are Oracle databases, but the CYC database technology is compatible with any other SQL database. In fact, it would be quite easy to modify the executor to communicate using ODBC or OpenDoc.

A.3 Knowledge-Enhanced Searching of Captioned Information

Large organizations frequently possess large libraries of "opaque" information sources; that is, data which do not lend themselves to traditional text-based searching methods. A news agency may possess a library of thousands of news photos; a movie studio thousands of film clips; a software help desk thousands of text articles too unwieldy to index directly. When such libraries must be searched, a common solution is to attach to each item a short text caption describing its contents. Thus a news photo of a soldier holding a gun to a woman's head might be captioned "a soldier holding a gun to a woman's head", plus a few tags for time and place, and then could be retrieved by querying for "soldier" or "gun".

This solution, while certainly adequate, is far from ideal. It would be nice if the photo could also be retrieved by queries for someone in danger", or "a frightened person",

or "a man threatening a woman". Such an achievement, however, lies far beyond the abilities of even the most sophisticated of traditional text–searching tools, all of which are fundamentally based on simple string matching and synonyms. Most search tools lack the ability to handle NL queries, and even those that do have some NL capability lack the background of commonsense knowledge required to make a connection between having a gun to one's head and feeling frightened.

CYC is not crippled by such a liability. CYC knows that guns shoot bullets and are designed to kill people; that having a gun to one's head therefore threatens one's life; that those whose lives are threatened feel fear; and that the vast majority of soldiers are men. CYC can therefore conclude that the image in question is, in all likelihood, a good match for each of the queries above.

A major focus of the CYC team's efforts in 1994 was the creation of an image-retrieval application for a major corporate partner which possessed a library of hundreds of thousands of captioned images, but no fully satisfactory way to search them. Building on the CYC inference engine and CYC's NL capabilities, a system was developed along the lines described above.

First, the content of each image in the library is described to CYC by converting the english captions to *CycL* and adding the resulting formulas to the knowledge base. CYC's NL tools permit the english-to-CycL translation to be mostly automatic; human intervention is required only in unusual cases.

Once the target images have been described to CYC, the system is ready to accept queries, which can be issued in plain english. CYC begins by converting the english queries to *CycL*, again using CYC's NL tools. For example, the english query "a frightened person" might be parsed as:

```
($and
   ($isa ?x $Person)
   ($feelsEmotion ?x $Fear $High))
```

After asking the user to confirm its parse (or, occasionally, to choose one of two or more equally valid parses), CYC begins to backchain from the query expression, using the image descriptions and other knowledge in the KB. When it is able to unify all the free variables with the elements of a picture (in our example, `?x` would unify with the woman in the picture), CYC knows it has found a match.

Cycorp has generalized this approach to image retrieval to extend it to other opaque–information–source retrieval applications. For example, document retrieval from large libraries of text documents described by short abstracts (analogous to image captions), is a task nearly identical in structure to that of retrieving captioned images. In fact, any database of captions, summaries, or abstracts could be handled similarly, whether the corresponding library contained images, sounds, video, text, or anything else.

A.4 Thesaurus Management

Cycorp has built a Thesaurus Manager tool that allows users to import and simultaneously manage and integrate multiple industry–specific and other thesauri. This tool is the most advanced multi-thesaurus manager available; it allows any combination of thesauri to be "active", it has machine–guided "conceptual merging" of thesaurus terms based on the CYC ontology, and it is designed to work with various existing thesaurus tools. A working prototype of this application has been created using medical thesauri, and is currently in the evaluation stage of development. Cycorp is considering productizing the Thesaurus Management tool for clients in the medical field and in other fields.

A.5 Distributed AI

Within the next few years, CYC may be installed at sites where even thousands of users need to harness its power simultaneously. An important question is: what should the architecture of such an installation look like?

While one CYC server can easily support a dozen clients at the same time, handling operations from thousands of clients would bring a CYC server to its knees. How do we address this scalability issue?

The obvious alternative is an architecture in which each user (or workgroup) runs its own local CYC server, with its own copy of the CYC knowledge base. But this scenario does little to alleviate the scalability problem. Now, each user is saddled with the responsibility of hosting a knowledge base which already contains nearly half a million assertions, and could easily grow into the millions as more and more specialized knowledge is added. Moreover, each user must tolerate the computational and communicational overhead associated with keeping each of the thousands of copies of CYC in sync with each other.

A great deal of the redundancy and inefficiency of such an architecture can be eliminated by adapting CYC to work in a distributed fashion. In fact, this is how humans work: we *specialize* and *collaborate*. Although most humans share a core of common knowledge which allows them to communicate, no one human possesses all the knowledge of the human race. Instead, we have experts in law, medicine, construction, automotive repair, and software design. When a human lacks the knowledge to solve a problem, he can often find a solution by collaborating with an expert. When I go to the doctor with a persistent headache, the doctor's knowledge of medicine combines with my knowledge of my symptoms to produce a solution that neither of us could have reached alone.

In a distributed CYC architecture, the network is populated with CYC agents, each of which shares a common core of knowledge, but possesses in addition one or more additions to the core knowledge base that extend its expertise into new domains. Most importantly, the various CYC agents are endowed with the ability to communicate with each other and to perform inferencing in a collaborative fashion. The inter–agent communication can be handled flexibly by using KQML or some other knowledge-sharing protocol, or it can be implemented using a more efficient CYC–specific protocol.

CYC agents in a distributed architecture share knowledge describing how they can be reached (via which network address, port, and protocol) and what their areas of expertise are. During inferencing, when an agent tries to expand a formula whose content lies outside its area of expertise, it determines (by consulting its own knowledge base or consulting with a central knowledge-broker agent) whether another agent is available that might be able to help. If so, the agent sends a message to its remote counterpart asking it to help expand the formula in question. The answer may be a complete set of bindings, but more commonly it is simply a partial result, still containing unbound variables. The local agent incorporates the result into its own proof tree just as if it had done the work itself, and continues its task. In many cases, the local agent may need to consult the remote agent, or several remote agents, multiple times in the course of its inferencing process.

Cycorp has cooperated with the Computer Science Department of the University of Maryland in Baltimore County (UMBC) to develop a demo of such a distributed architecture. In the demo, three CYC agents communicate with each other using KQML. While all three agents possess the same core knowledge base, each possesses additional knowledge about an additional domain in which it is consider expert: the GeoAgent in geography, the PolAgent in politics, and the EcoAgent in economics. Working together, they can answer queries that no one of them could have answered alone.

For example, suppose a user asks the GeoAgent for "elected heads of government of countries north of the equator". This might be represented as:

```
($and
   ($headOfGovernmentOf ?x ?y)
   ($hasAttributes ?x $Elected)
   ($northOf ?y $Equator))
```

The GeoAgent is able to find bindings for the third clause by using its own knowledge of the geography domain:

- Britain is in Europe.
- Europe is in the northern hemisphere.
- The northern hemisphere is north of the equator.
- If region A is part of region B, and region B is north of region C, then region A is north of region C.
- Therefore, Britain is north of the

But to find bindings for the first two clauses, the GeoAgent must enlist outside help. It sends these as queries to the PolAgent, which is able to find bindings for them using its knowledge of politics domain:

- Heads of government of democratic countries are elected.

- Great Britain is a democratic country. item Tony Blair is the head of government of Britain.
- Therefore,Tony Blair is the elected head of government of Great Britain.

When the GeoAgent agent receives this answer back from the PolAgent, it combines it with its own partial answer to produce the final result: Tony Blair.

An important research focus of the Cycorp/UMBC project has been optimizing the implementation of collaborative inferencing so that the costs of managing the necessary communications overhead are significantly outweighed by the increase in scope and efficiency which result from sharing the inferencing workload.

It should also be pointed out that, while the description above assumes that all the agents participating in the distributed architecture are CYC agents, this is not a requirement. Cycorp has defined a very simple protocol for implementing cooperative inferencing, and any agent which has been taught to adhere to this protocol, and which possesses knowledge of interest to other agents, can meaningfully participate in such an architecture. For instance, the role normally played by a CYC agent in a distributed architecture could equally well be filled by a gateway to

- an expert system implemented in Prolog
- an SQL database
- a special-purpose inferencing tool (e.g. for spatial inferencing)
- a human expert, etc.

In fact, the WWW Information Retrieval application, described next, builds on the distributed CYC architecture by filling that role with a gateway to a WWW information source.

A.6 WWW Information Retrieval

The explosion of the World Wide Web during the last two years has created a tremendous opportunity. The WWW is home to a vast quantity of information, much of which could, in principle, be used to make CYC more intelligent, while shortcutting the laborious process of manual knowledge entry.

This could happen in one of two ways: either online information could be extracted, converted to *CycL*, and incorporated directly into the knowledge base, or CYC could be taught to treat external information sources as extensions of the KB, without directly incorporating their contents.

Cycorp has explored the first approach during 1996 CYC is nearing the critical mass required for the reading and assimilation of online texts (news stories, encyclopedia articles, etc.). In this scenario, CYC's natural language tools are used to process online texts, converting them to *CycL* for inclusion in the knowledge base.

Cycorp is also pursuing the second approach. In this scenario, we write gateways which disguise WWW information sources as CYC agents (with a very limited domain of expertise), available to operate in a distributed CYC architecture. There is a large and ever-growing number of information sources on the WWW which might fill this role. All, however, share the following characteristics:

- They embody a large corpus of knowledge,
- they can respond to HTTP queries for specific knowledge, and
- they present their knowledge in an HTML format which, while generally not as regularly structured as a database, is sufficiently structured that it can be parsed by a relatively simple algorithm.

An example is the Internet Movie Database, a truly stupendous compendium of movie knowledge. A quick browse of the IMD demonstrates that it embodies virtually everything there is to know about movies; that it can respond to queries for particular actors, movies, etc.; and that it displays the results in a format, which, while it varies somewhat from one actor, movie, etc. to another, depending on what information is available, is nevertheless fairly regular.

The contents of the IMD can be annexed to the CYC KB by creating a gateway which, on the one hand, interacts with CYC agents exactly as a CYC agent operating in a distributed CYC architecture would, and, on the other hand, simulates the interaction of a WWW browser with the IMD HTTP server.

This gateway is advertised to CYC as an expert in the movie domain, so that whenever CYC receives a query in that domain, it turns to the gateway for assistance. For example, let's say a user asks CYC, "What movies did Ronald Reagan act in?". This might be represented in CycL as:

```
($actedInMovie $RonaldReagan ?x)
```

CYC hands this CycL query to the gateway, which understands enough about movie–related CycL vocabulary to translate this query into an HTTP request to the IMD server for a page on Ronald Reagan. When the IMD server returns the page, the gateway parses the HTML, extracts a list of the movies in which Reagan appeared, converts the result to CycL, and then constructs a suitable reply to the CYC agent making the request. (For reasons not worth explaining, the reply contains more than just the answer.)

To the user interacting with CYC, this transaction is entirely transparent. It appears to the user as if CYC now knows everything there is to know about movies, and yet the KB still fits on the hard disk!

Enhancing CYC's cinematic erudition may be a fairly frivolous application of the techniques described above, but the WWW contains plenty of large sources of semi–structured information on weightier topics (stock quotes, company profiles, WWW indexes, resumes, the CIA World Factbook, etc.). The ability to effectively incorporate vast portions of

the WWW into a "virtual" knowledge base is a compelling possibility. Not only would it greatly expand the effective scope of CYC's knowledge, but it would do so at little cost to the CYC development team.