Common Sense Reasoning – From Cyc to Intelligent Assistant

Kathy Panton, Cynthia Matuszek, Douglas Lenat, Dave Schneider, Michael Witbrock, Nick Siegel, and Blake Shepard

Cycorp, Inc.
3721 Executive Center Drive, Suite 100, Austin, TX 78731, USA
{panton, cyndy, lenat, daves, witbrock, nsiegel, blake}@cyc.com

Abstract. Semi-formally represented knowledge, such as the use of standardized keywords, is a traditional and valuable mechanism for helping people to access information. Extending that mechanism to include formally represented knowledge (based on a shared ontology) presents a more effective way of sharing large bodies of knowledge between groups; reasoning systems that draw on that knowledge are the logical counterparts to tools that perform well on a single, rigidly defined task. The underlying philosophy of the Cyc Project is that software will never reach its full potential until it can react flexibly to a variety of challenges. Furthermore, systems should not only handle tasks automatically, but also actively anticipate the need to perform them. A system that rests on a large, general-purpose knowledge base can potentially manage tasks that require world knowledge, or "common sense" – the knowledge that every person assumes his neighbors also possess. Until that knowledge is fully represented and integrated, tools will continue to be, at best, *idiots savants*. Accordingly, this paper will in part present progress made in the overall Cyc Project during its twenty-year lifespan – its vision, its achievements thus far, and the work that remains to be done. We will also describe how these capabilities can be brought together into a useful ambient assistant application.

Ultimately, intelligent software assistants should dramatically reduce the time and cognitive effort spent on infrastructure tasks. Software assistants should be *ambient systems* – a user works within an environment in which agents are actively trying to classify the user's activities, predict useful subtasks and expected future tasks, and, proactively, perform those tasks or at least the sub-tasks that can be performed automatically. This in turn requires a variety of necessary technologies (including script and plan recognition, abductive reasoning, integration of external knowledge sources, facilitating appropriate knowledge entry and hypothesis formation), which must be integrated into the Cyc reasoning system and Knowledge Base to be fully effective.

1 The Evolution of Cyc

1.1 Beginnings of the Cyc Project

In the early 1970s, rule-based expert systems such as MYCIN [18] and DENDRAL [3] were AI's major success story. MYCIN acted as an assistant in the diagnosis of

Y. Cai and J. Abascal (Eds.): Ambient Intelligence in Everyday Life, LNAI 3864, pp. 1 – 31, 2006.

blood infections, while DENDRAL's expertise was in chemical analysis. These applications used rules to solve problems within circumscribed domains. Expert systems represented a major step forward in AI technology and are used today to address problems as diverse as camera lens design and cargo placement [5], but their limitations quickly became obvious. Lenat and Guha [10] provide several examples of the *brittleness* displayed by expert systems. Two will suffice here:

> An expert system authorizes a car loan to someone who stated, on his application, that he'd worked at the same job for twenty years. A good risk? Perhaps, but the individual also stated he was 18 years old.
>
> A skin disease diagnosis program is told about a "patient" that is a 1969 Chevrolet:
>
> ***Program:*** Are there spots on the body?
> ***User:*** Yes.
> ***Program:*** What color spots?
> ***User:*** Reddish-brown.
> ***Program:*** Are there more spots on the trunk than elsewhere?
> ***User:*** No.
> ***Program:*** The patient has measles.

In the first example, the system failed to notice what was likely to have been a simple typo; perhaps the applicant meant that he had been at his current job for two years, or 20 months. Rules encoded in that system might nevertheless conclude that someone employed by the same company for 20 years is a very good credit risk, resulting in an easy loan approval. The system breaks down because it cannot detect what, to humans, are very obvious contradictions. These errors can have effects far more dire than in the car loan case. For example, a medical transcriptionist accidentally transposing a patient's weight and age in a patient record could lead to that patient being prescribed medications at dangerously incorrect dosage levels.

The second example illustrates that expert systems work only within the domain for which they were explicitly engineered; this software cannot correctly diagnose rust spots on a car. Furthermore, the system is unable to use its knowledge about skin infections to do things like recommend treatment or explain to users how long the disease might last and what other symptoms the patient may be experiencing. In short, this software contains many handcrafted rules that encode useful information about skin diseases, but this knowledge is isolated and opaque: it is useless when applied to an object outside its domain, and cannot be reused across similar or related problems.

Expert systems have no understanding of what they are for, or the extent of their own knowledge. But their brittleness is mainly due to a lack of *common sense*. This is the general knowledge that allows us to get by in the real world, and to flexibly understand and react to novel situations. We remember and use (though usually not consciously) heuristics such as "Water makes things wet"; "Wet metal may rust"; "No two objects can occupy the same space at the same time"; and "Inanimate objects don't get diseases".

The driving force behind the Cyc Project was the realization that almost all software programs would benefit from the application of common sense. Expert systems would gain protection against user error or intentional fraud; the consistency of data in spreadsheets could be checked automatically; information-retrieval systems and word processors could exhibit more useful behaviors based on an understanding of the user's goals at any given point. The greatest impediment to the achievement of AI was the inability of programs to accumulate, apply, and reuse general knowledge.

Lenat began the Cyc Project in 1984, at the Microelectronics and Computer Technology Corporation in Austin, Texas, with the goal of building a single intelligent agent. This agent would be equipped not just with static facts, but also with heuristics and other problem-solving methods that would allow it to act as a substrate, an almost invisible performance-boosting layer, underlying a variety of software applications. The Cyc Project was initially envisioned as a series of ten-year, two-to-ten-person-century efforts, in (1) knowledge base and ontology building, or "pump priming"; (2) natural language understanding and interactive dialogue; and (3) automated discovery.

1.2 Representing Knowledge

Three preliminary research questions presented themselves: How much does a system need to know in order to be useful? What kinds of knowledge are necessary? How should this knowledge be represented?

1.2.1 Amount of Knowledge

The "annoying, inelegant, but apparently true" answer to the first question was that vast amounts of commonsense knowledge, representing human consensus reality, would need to be encoded to produce a general AI system (Lenat and Guha 1990) [10]. In order to mimic human reasoning, Cyc would require background knowledge regarding science, society and culture, climate and weather, money and financial systems, health care, history, politics, and many other domains of human experience. The Cyc Project team expected to encode at least a million facts spanning these and many other topic areas.

1.2.2 Kinds of Knowledge

Lenat and his team understood that the "pump priming" information was not specific facts (e.g. "President Lincoln was assassinated"), but rather the "white space" – the unstated knowledge which the writer of such sentences assumes the reader already knows, such as the fact that being President requires one to be alive.

In order to truly comprehend a sentence such as "President Abraham Lincoln was assassinated", and be able to make inferences about its likely consequences, a person must have already learned many facts about the world. Someone unable to answer the following questions cannot be said to have fully understood the example sentence:

What is a President?
Was Lincoln President two months after he was assassinated?
Was Lincoln alive 300 years before he was assassinated?

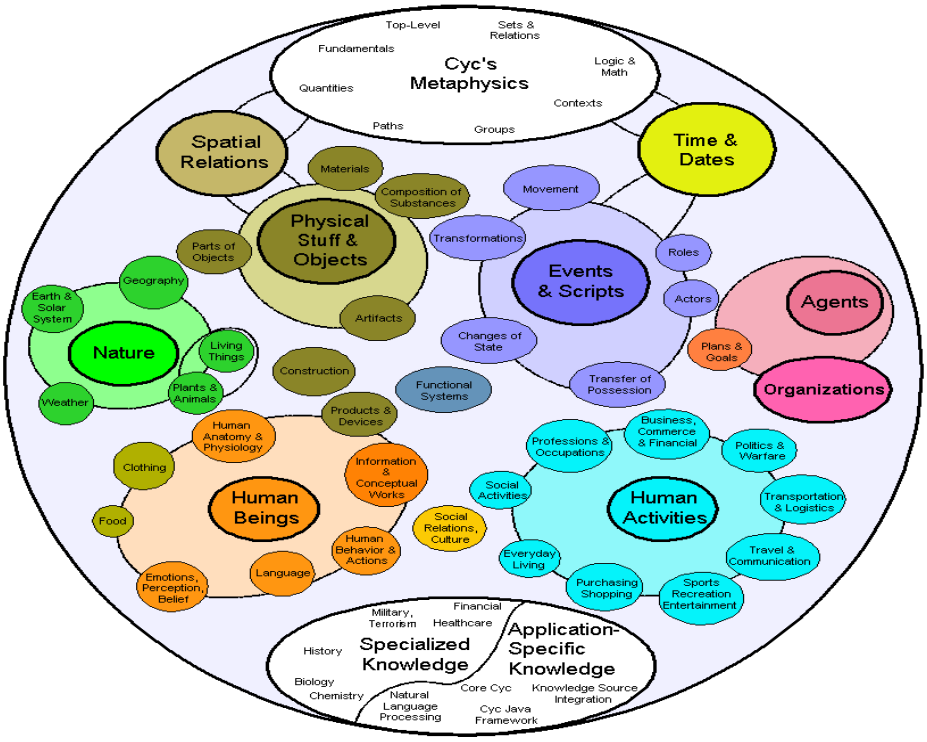

Fig. 1. Cyc KB Topic Map. The information in the Cyc KB can by subdivided into loosely grouped, inter-related "blocks" of knowledge at various levels of generality. In this diagram, Cyc's understanding of metaphysics is the most general block of knowledge, gradating down to the very specific knowledge in tightly defined domains.

> Can a chair be assassinated? Can a cat? An idea?
> Was Lincoln's assassin on the same continent as Lincoln when the assassination occurred?

The task, then, was transformed into one of entering not raw facts but rather a kind of pre-factual knowledge, the type of information humans learn effortlessly but rarely need to articulate.

The question of how to get this knowledge into Cyc provoked much discussion. Many AI researchers were turning to techniques such as Natural Language Understanding (NLU) or Machine Learning, in an attempt to glean knowledge from textual sources or from a seedling knowledge base. After having worked in these areas in the 1970s, however, Lenat became convinced that there was no "free lunch" in building a real intelligence. There was the chicken-and-egg problem: in order to translate text into a useful semantic representation, an NLU system needs common sense. Imagine a system with absolutely no knowledge of the world trying to determine what the ambiguous word "bank" means in the sentence, "The bank is

closed on Sundays", or deciding to whom the pronoun "they" refers in the following sentences:

The police arrested the protestors because they feared violence.
The police arrested the protestors because they advocated violence.

It became apparent by 1983 (Lenat et al. 1983) [9] that the kind of knowledge an intelligent system must have – pre-factual consensus knowledge – was not likely to be found in any textbook, almanac or manual. Cyc could not directly learn facts such as "Once people die, they stay dead" from text sources, even if an NLU system that could handle such input were available. Cyc's developers realized that NLU and Machine Learning would be worthwhile approaches to building the Cyc knowledge base (KB) only after a broad foundation of common sense knowledge already existed. An intelligent system (like a person) learns at the fringes of what it already knows. It follows, therefore, that the more a system knows, the more (and more easily) it can learn new facts related to its existing knowledge. It was determined that common sense must come first, and that initially it would have to be codified and entered manually into Cyc. This was the basis for the creation of the Cyc Project at MCC in Austin, in 1984.

1.2.3 Knowledge Representation Formalism

Now that the question of what to teach Cyc had been addressed, the next step was to find an adequate formalism – a representation language – in which to encode that knowledge. The earliest approach adopted was the now-familiar frame-and-slot language, in which terms are related by binary predicates:

```
GeorgeWBush
-----------------------------------------------
likesAsFriend: VladimirPutin

"George Bush considers Vladimir Putin a friend."

Mercury
-----------------------------------------------
genls: UnalloyedMetal

"Mercury is an unalloyed metal."
```

However, limitations of the frame-and-slot method quickly became apparent. First, it was impossible to make quantified statements such as "All mammals have legs." Second, modal operators such as "believes" and "desires", which require embedded clauses, could not be expressed; nor could other (often implicit) aspects of context such as time, space, and number be captured (e.g., "In modern Western cultures, adults are generally permitted to have only one spouse at a time.").

Clearly, more expressivity was needed. Other desiderata for a representation language fell out from the requirement that Cyc be able to perform useful types of reasoning over its knowledge. In summary, Cyc's representation formalism needed to:

- Have a clear, simple, declarative semantics;
- Include conjunctions, disjunctions, quantifiers, equality, and inequality operators;

- Allow for meta-level assertions, or statements about statements (e.g., "This rule was entered by Doug Lenat on March 2, 1986");
- Support inference mechanisms such as verifying conjectures, finding appropriate bindings for variables, and planning;
- Allow nested expressions, such as those found in statements of propositional attitudes (e.g., "Joe believes that Tom's birthday is tomorrow.").

1.3 CycL: Cyc's Representation Language

Cyc's representation language is known as CycL. It is essentially an augmented version of first-order predicate calculus (FOPC). All of the FOPC connectives, such as *and*, *or*, and *implies*, are present, as are the quantifiers. One crucial extension was the ability to handle *default reasoning*; aside from intrinsically definitional information (e.g., "All dogs are mammals"), there are few general statements one can make about the world that cannot have exceptions. Some examples:

> You can see other peoples' noses, but not their hearts.
>
> Given two professions, either one is a specialization of the other, or else they are likely to be practiced independently.
>
> Dogs have four legs.

These statements are usually true, but three-legged dogs do exist; thoracic surgeons routinely see their patients' hearts; and there are some people who practice two separate professions simultaneously. Therefore, standard truth-conditional logic, in which statements are only either true or false, would not suffice. Currently, every assertion in the KB carries a truth value that indicates its degree of truth. CycL contains five possible truth values: *monotonically false, default false, unknown, default true,* and *monotonically true.* Most assertions in the KB are either *default true* or *monotonically true.* Default assertions can be overridden by new knowledge, whether it comes from a person using Cyc or is derived by Cyc's own inference engine. Instead of using only a single support or line of reasoning to determine if an assertion is true or false, Cyc's inference engine uses argumentation. This is the process of weighing various arguments, pro and con, to arrive at a truth value for the assertion. Cyc employs a number of heuristics during argumentation. One simple example is to prefer monotonic rules: if two rules conclude P but with different truth values (i.e., one concludes that P is monotonically true but the other concludes that P is default false), then, all else being equal, Cyc sets the truth value of P to the one suggested by the monotonic rule.

Arguments consist of justification chains showing which proofs (ground facts, rules, and inference methods) were used to arrive at a conclusion. Figure 2 shows a partial inference chain constructed in response to the query, "Can the Earth run a marathon?"

1.4 Structure of the Cyc KB

The Cyc KB consists of terms representing individuals (e.g., `CityOfParisFrance`, `BillClinton`) and natural kinds (`Platinum`, `PineTree`). Cyc predicates can

have as many arguments as are appropriate, though one-, two-, and three-place predicates are most common. Functions that can be applied to existing terms in order to construct new ones – e.g., `LiquidFn` and `BorderBetweenFn` – permit the compositional reification of an unlimited number of new "non-atomic" collections and individuals, such as `(LiquidFn Nitrogen)` and `(BorderBetweenFn Sweden Norway)`.

Early on, it became clear that a very large knowledge base would eventually run into the problem of internal inconsistency: statements would begin to contradict one another. Such contradictions do not necessarily indicate a fault in the formulation of the assertions themselves; humans encounter this phenomenon on a regular basis. Imagine the following conversation:

> CHILD: Who is Dracula, Dad?
> FATHER: A vampire.
> CHILD: Are there really vampires?
> FATHER: No, vampires don't exist.

This exchange is contradictory at first blush, but does not remain so when we consider that the truth of each statement depends on some implicit *context of reference*. In answering the child's first question, the father frames his response within the context of mythology and fiction. His second answer, however, is framed in a real-world context. In the fictional world of vampires, it is true that they exist, tend to be pale, and have sharp canine teeth used for puncturing necks. In the actual world, there are no vampires (though there are vampire stories).

Since it is impossible to maintain global consistency in a knowledge base containing millions of assertions, the Cyc Project's approach was to aim for local consistency instead; this approach was later extended in the thesis work of R.V. Guha [10]. This is achieved by situating each assertion in a particular *microtheory* (essentially, an explicitly represented logical context). Assertions within a microtheory must be consistent with one another, but need not be consistent with those in other microtheories. Additionally, bundles of assertions in a microtheory tend to share many background assumptions. The microtheory mechanism allows these assumptions to be stated once, at the level of the microtheory, instead of having to be repeated for each affected assertion. For example, the microtheory named `NormalPhysicalConditionsMt` contains assertions such as "Fluorine is a gas" and "Glass has a high amount of shear strength". These assertions share the domain assumptions that temperature, pressure, etc. are in the normal range for Earth's surface.

Cyc's microtheory mechanism also allows for more efficient inference, in many cases. When solving certain types of problems, Cyc knows, or is told, that some microtheories must be consulted, while others are irrelevant. This reduces the search space, speeding up the inference process.

1.5 Addressing Structured External Knowledge in Cyc

In the Cyc KB, it is useful to make the distinction between *knowledge* (the underlying heuristics that allow us to reason) and *data* (facts or statements about specific items in

Fig. 2. Partial Inference Tree for a proof that the planet Earth is not capable of running a marathon, supported by a proof that, more generally, inanimate objects can't run marathons

the world). Knowledge must be hand-crafted and entered into Cyc; it is knowledge that allows Cyc to even begin to understand data. Over the past few years, Cycorp has developed components, jointly referred to as SKSI (Semantic Knowledge Source Integration), which allow knowledge and data to each be stored and accessed in an appropriate way (Masters and Güngördü 2003) [11].

Just as a human researcher would not bother to memorize certain facts, many kinds of data are best kept out of the KB proper. Storage constraints are one reason: although memory is progressively less expensive and more available, a true common-sense system will require an amount of information that is vast even by modern standards. Why use up precious memory to reify every brand-name product sold in the world? As well, many types of data are unstable or ephemeral, varying significantly over time. One simple example is stock prices. Telling Cyc about the stock price of each company on the New York Stock Exchange would be essentially pointless, since that information would rapidly become stale. Rather than continually updating that data within the KB, it would be much more appropriate to consult that data at the source, in real time, so Cyc can always use the most up-to-date figures. These concerns were the motivation for SKSI, which allows Cyc to access data in structured sources, such as databases, and reason with the facts it finds in exactly the same way it reasons with its own native knowledge.

Cyc's SKSI technology allows the meanings of database columns (for example, "This column represents a person's office phone number") to be described in CycL. Only the *interpretation* of database information is represented within Cyc; all the actual data remains in its native format, in the original database. Cyc, in effect, knows how to generate SQL queries to access any particular field in any particular row, and how to interpret what it finds there. These mappings allow Cyc to combine information from multiple databases, or from a combination of databases plus the Cyc KB itself, to answer user queries. Many current Cyc applications assume the need to consult a variety of sources, such as the KB itself; structured data in databases; and unstructured data, such as news articles and Web pages.

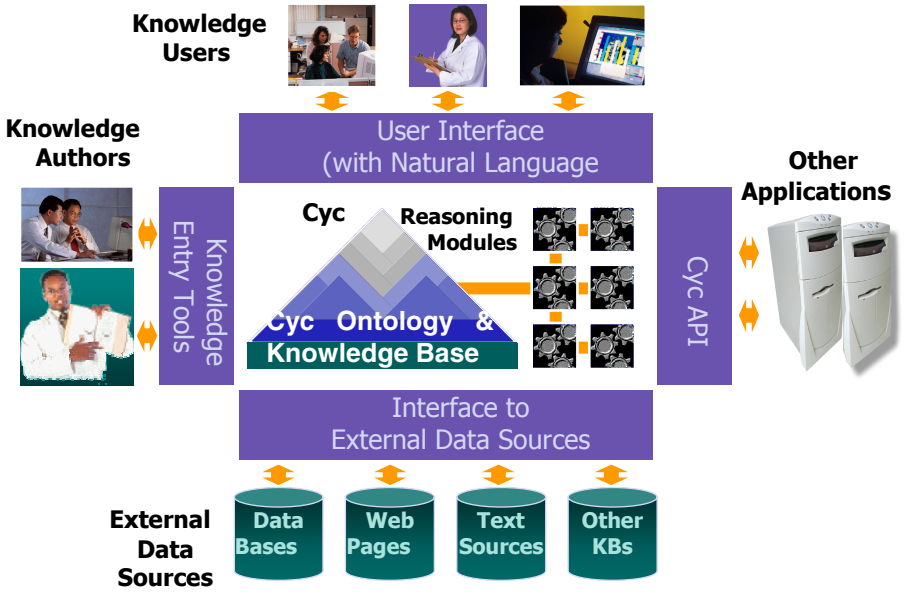

Fig. 3. Overview of Cyc. The complete Cyc system has users who retrieve knowledge in a variety of ways, and authors who generate knowledge (and who overlap with users). The possible interfaces to the Cyc KB and reasoning capabilities – both those based on NL and more programmatic interfaces to other applications and data sources – are shown on the right and bottom.

2 The Case for an Ambient Research Assistant

Performing scientific research is a knowledge-intensive task. Locating, maintaining, and organizing the knowledge required for effective investigation, without interfering with the creative process, catalyzes successful research. Lacking truly intelligent support, researchers must spend a significant portion of their professional time performing infrastructure tasks, all of which are crucial to the overall process. Researchers must manage their personal information flow, locate related work and sources of related expertise, stay abreast of work going on in the field they are

working in, ensure that critical laboratory and computing infrastructure is in place, perform task tracking, write proposals, write and submit publications, and a score of other functions only tangentially related to the actual cycle of hypothesizing, testing, and revising scientific knowledge. In a broader sense, this is true in every field of human endeavor. Providing better tools and interfaces can reduce the time and effort involved in any given task. Project management software simplifies task tracking, and good word processors make writing papers easier. However, some duties do not lend themselves to the creation of specialized tools – arranging for laboratory infrastructure is a time-consuming role that varies enormously across workplaces. Furthermore, such tools do not eliminate the need to perform the tasks, or the attention absorbed by task switching.

Historically, the only agents capable of reducing that load are fairly highly trained human assistants. The range of tasks such an assistant may be called upon to perform is broad, whereas the capabilities built into even the most useful tools are *deep* – that is, focused on a particular task; while Google Scholar, Cora (McCallum et al. 1999) [14], and CiteSeer (Bollacker et al. 1998) [1] reduce the time spent researching related work, they cannot handle arbitrary queries that draw on real-world knowledge, e.g., when was a particular piece of research performed, what is the relationship among research groups in a particular field – concepts that a competent assistant possessed of common sense and a relatively small amount of domain expertise can handle readily. Semi-formally represented knowledge, such as the use of standardized keywords and publication formats, is a traditional and valuable mechanism for helping scientists access information. Extending that to formally represented knowledge (based on a shared ontology) is an effective way of sharing large bodies of knowledge between groups, and reasoning systems that draw on that knowledge are the logical counterpart to tools that perform well on a single, rigidly defined task. A system that rests on a large, non-domain-specific knowledge base can potentially manage tasks that require world knowledge, or common sense – the knowledge that every person can reasonably assume every other person possesses. Until that knowledge is fully represented and integrated, tools will continue to be, at best, *idiots savants*.

The underlying philosophy of the Cyc Project is that software will never reach its full potential until it can react flexibly to a variety of challenges. Systems should not only handle tasks automatically, but also actively anticipate the need to perform them. This requires the development of a variety of technologies (script recognition, integration of external knowledge sources, facilitation of appropriate knowledge entry, hypothesis formation, and so on). These technologies must be integrated into a reasoning system that is possessed of a broad base of pre-existing knowledge, such as the Cyc knowledge base (KB), which provides the system with enough information to understand the context in which each task is being performed.

2.1 The Role of Assistance

Flexibility is key to creating a truly useful assistant. A good assistant is capable of handling arbitrary problems in arbitrary domains with a minimum of instruction. People are particularly well suited to this because they possess a store of real-world knowledge and skills that allows them to rapidly switch from one type of task to another; even a comparatively lightly trained human can find a submission address

and fax a paper to it, schedule a meeting among several people, and respond to external queries about availability. While different duties may demand different levels of expertise in the scientific domain, almost all require some background knowledge as well. Meanwhile, existing computational "assistants" are deep rather than broad; each focuses on solving a particular problem, and is brittle when faced with anything outside that limited area.

Another key characteristic of a useful assistant is *ease of communication*. If describing or spawning off a task is too expensive, in terms of time or cognition, it becomes impractical to hand those tasks off rather than simply performing them. While a particular formal representation might work best when describing a particular type of task (e.g. how to compute variance for an experiment), the best way of communicating information about a variety of different tasks with minimal cognitive load to the researcher is natural language.

Finally, even if the description and training process is initially complex, an assistant must be capable of *learning*. Generally, tasks do not need to be described each time they must be performed; capable assistants can learn from tasks performed successfully, task failures, and analogous functions they already know how to perform. This idea of learning actually describes a wide variety of functions and capabilities:

1. **Deciding what facts to learn.** An assistant system must reason about what knowledge gaps would be most cost-effective to fill in any given context. If a researcher is considering submitting a paper to an upcoming conference, finding submission dates and contact information is likely to be more useful than organizing older work, and should be a higher priority task.
2. **Learning those facts.** The factual gaps should be filled, from available documentation, online sources, and/or communication with the scientist being assisted. In the aforementioned example, the system should set out to learn any missing facts by appropriately querying all its available sources, both online ones and people, starting with the conference web site or call-for-papers and progressing to information that requires some knowledge of the research in question. The submission information may depend on the track to which the paper is being submitted, which requires knowledge of the research topic.
3. **Learning of rules.** Once knowledge is acquired, it is possible to hypothesize general rules. If several conferences have been identified, an assistant might correlate information about each of them and conclude that conferences in some broad field (e.g. machine learning) are often of interest, or that knowing submission dates is often useful. Such a rule can then guide the selection and prioritization of tasks.
4. **Generalizing rules.** Carrying this example through, an effective assistant might learn from one or more identified rules that, for some particular user or researcher, learning and then tracking dates by which some particular action must occur is valuable.
5. **Testing and revision.** The rules, especially the generalized rules, will need to be tested independently of how they were produced. For example, when a general rule about tracking dates is hypothesized, a system might discover after experimentation that it is less helpful to track and remind a user of recurring dates, such as a weekly report that must be made to an overview body. This discovery would force revision (tightening) of the generalized rule.

2.2 The Limitations of Human Assistance

Ultimately, the goal of a personal assistant is to reduce the time and cognitive effort spent on infrastructure tasks. In some ways, computational systems have the potential to assist researchers at a level that a human assistant could never match. Some tasks are pervasive – it makes little sense to have a human assistant file each piece of email after reading, as the time spent splitting off many tiny tasks is greater than the effort of simply performing each task. An assistant's *availability* at the moment some duty must be performed is crucial. An ambient system, in which a user works within an environment in which some agent is actively trying to classify the behavior the user is engaging in, and perform subtasks, has the potential to assist with the many small tasks that create a burden of day-to-day effort, thereby providing assistance that a human assistant could not.

Another crucial behavior for a non-intrusive assistant – that is, one with minimal cognitive load for the user – is *anticipation* of the needs of the researcher. Ambient software assistants have the potential to classify the behavior the user is engaging in, predict useful subtasks and expected future tasks, and either perform those tasks or perform introductory steps before they are required, thus obviating the need for the researcher to identify and describe tasks. This requires *plan recognition,* identifying a user's actions as part of a script, which is either predefined (as in "reading email, then responding, then filing") or generated on the fly by the system. A script describing reading a research paper may include following reference links and seeking deeper definitions of key terms. If this script is recognized, the system might display recognized terms and links from those terms to other relevant ontologized knowledge while the user is still reading; a user might then be presented with a knowledge acquisition interface to define unrecognized terms, expanding the knowledge base.

2.3 Components of a Truly Intelligent Computational Assistant

Plan recognition: Creating a truly intelligent assistant will require substantial computational infrastructure. A crucial piece will be a component responsible for gathering information about how people actually perform certain tasks, in as much detail as possible; this is a prerequisite for figuring out how to automate pieces of those tasks. Similarly important is the capacity to *recognize* what a person is trying to do, and to *generate* new scripts to help a user optimize his workflow. All of these actions must occur more or less transparently to the user of the system; otherwise the cognitive load introduced by the intrusiveness of the tool will render it unusable.

Learning: Making truly intelligent use of any information collected, and minimizing the effort involved in using the system, requires many different kinds of *learning*, ranging from learning facts and rules to identifying patterns that can serve as a basis for script recognition. Although human training and reinforcement can be involved to some extent, especially with respect to reviewing the system's conclusions, the majority of this learning must perforce take place automatically. Taking advantage of the inferential capabilities present in Cyc allows the automatic or semi-automatic

conjecture of facts, collection of new facts, and production of hypothetical rules and scripts that can then be generalized, tightened, corrected, and used.

Natural Language: Finally, natural language understanding and generation are required for optimal interaction – a true assistant would be capable of handling aspects of full discourse processing. An assistant system must be able to remember questions, statements, etc. from the user, and what its own response was, in order to understand the kinds of language 'shortcuts' people normally use in context. Input from users will not always be in the form of full sentences or questions. The assistant will need to use the context of what has been said before, along with knowledge of the user's goals, to interpret requests or commands. For example, a researcher might ask "Can you find me any articles by M. Smith of Stanford on bioethics?" The assistant might then return a list of 200 such articles. The user might reasonably then ask, "Just show me the ones from 2001 or later", or "Which ones was he the main author on?" In order to seamlessly handle this interaction, the system needs to be able to interpret references (like "the ones" or "he") to objects already present in the discourse space.

Background on the history, goals, and current state of the Cyc Project were given in Section 1; what follows is an overview of work being done at Cycorp on each of these three high-level components: natural language processing, learning, and ambient interaction.

3 Natural Language Processing in Cyc

Spoon-feeding knowledge into Cyc (creating terms and hand-ontologizing each fact about each concept) is time-consuming, and Cyc's ontological engineers must become adept at translating back and forth between English and CycL. This is a tedious mechanism for knowledge acquisition, and furthermore does not allow free interaction with users who are not trained in CycL. A few years into the Cyc Project, work began on limited natural language processing using Cyc as a substrate, though the plan was always to focus more on NLP during Cyc's second decade of development.

3.1 Components of Natural Language in Cyc

Natural language understanding (NLU) and natural language generation (NLG) capabilities allow users to interact with Cyc using English instead of CycL. With these capabilities, Cyc can start down the road toward being able to read texts and learn new information on its own.

3.1.1 Cyc's Lexicon

At the core of Cyc's natural language processing capabilities is its English lexicon (Burns and Davis 1999) [4]. This lexicon contains information about the syntactic properties of words and phrases (e.g. "tree" is a noun; "eat" has both transitive and intransitive uses), as well as a compositional morphological representation for derived words (for example, "flawless" is decomposed into the root "flaw" plus the suffix "less"). Most important, though, are the semantic links: pointers from words and

phrases to concepts and formulas in the Cyc KB. It is these links that allow for translation of English sentences into fully formed CycL representations, and vice versa. Cyc's lexicon currently contains entries for over 20,000 single-word noun, verb, adjective, and adverb forms; 40,000 multi-word phrases; and more than 100,000 proper names. Following are partial lexical entries for the words "tree" and "eat":

Lexical Information for "tree"

- CycL: `(#$denotation #$Tree-TheWord #$CountNoun 1 #$Tree-ThePlant)`
  Meaning: `#$Tree-TheWord` is a count noun denoting `#$Tree-ThePlant`
- CycL: `(#$singular #$Tree-TheWord "tree")`
  Meaning: One singular form of `#$Tree-TheWord` is the string "tree".

Lexical Information for "eat"

- CycL: `(denotation Eat-TheWord Verb 1 EatingEvent)`
  Meaning: `#$Eat-TheWord` is a verb which denotes an `#$EatingEvent`
- Semantic translations of 'eat':

```
(verbSemTrans Eat-TheWord 0 TransitiveNPFrame
       (and
              (isa :ACTION EatingEvent)
              (performedBy :ACTION :SUBJECT)
              (consumedObject :ACTION :OBJECT)))

(verbSemTrans Eat-TheWord 0 IntransitiveVerbFrame
       (and
              (isa :ACTION EatingEvent)
              (performedBy :ACTION :SUBJECT)))
```

Cyc's lexicon makes use of a specialized microtheory structure to represent languages, including various dialects of English as well as other non-English languages (German, French, Spanish, etc.). Small numbers of words and phrases in these other languages have been added, mainly as a proof-of-concept that Cyc's lexical representation vocabulary can handle or be easily extended to handle a variety of languages. The current focus is on parsing and generating English, though projects involving other languages, such as Chinese, are underway (Schneider et al. 2005) [17].

3.1.2 Natural Language Generation

Cyc's NLG system produces a word-, phrase-, or sentence-level paraphrase of KB concepts, rules, and queries. This system relies on information contained in the lexicon, and is driven by generation templates stored in the knowledge base. These templates are not solely string-based; they contain linguistic features that allow, for example, a variety of verb tenses, and correct grammatical agreement, to be produced. The NLG system is capable of providing two levels of paraphrase, depending on the demands of the application. One type of generated text is terse but potentially ambiguous, while the other is more precise, but potentially wordy and stilted. Automated interface tools assist users in adding new generation templates as they introduce new concepts into the knowledge base.

Generation for the predicate #$hasDiet

- CycL Template:

```
(genTemplate hasDiet
    (PhraseFormFn NLSentence
        (ConcatenatePhrasesFn
        (BestDetNbarFn-Indefinite (TermParaphraseFn :ARG1))
        (BestNLWordFormOfLexemeFn-Constrained Adverb
TypicalTheWord)
        (BestHeadVerbForInitialSubjectFn Eat-TheWord)
        (BestDetNbarFn-Indefinite (TermParaphraseFn :ARG2)))))
```

- CycL: `(#$hasDiet #$Termite #$Wood)`
- Generated Text: "Termites typically eat wood."

3.1.3 Natural Language Understanding

With regard to NLU, depth of parsing from natural language to CycL can range from very shallow (for example, simply mapping strings to concepts) to deep (full text understanding, via translation to CycL formulas). Cyc-based applications have differing needs with respect to parsing speed, depth, and accuracy. This has resulted in the development of a number of in-house parsing tools, including standard CFG parsers and template parsers. External parsing tools, such as Charniak's (2001) statistical parser and the LINK parser developed at Carnegie-Mellon University (Sleator and Temperley 1991) [20], have also been adapted and used with Cyc's semantic translation tools.

Parsing Example

- English: "Bill Clinton sleeps."
- Parsed CycL:

```
(#$thereExists ?SLEEPS
       (#$and
              (#$isa ?SLEEPS #$Sleeping)
              (#$bodilyDoer ?SLEEPS #$BillClinton)))
```

3.2 Question-Answering Via Natural Language

Cycorp has developed a Cyc-based natural-language question-answering application, designed to operate over heterogeneous text and structured data. The goal of this work is to enable the proactive seeking out of sets of facts that may be relevant to any given task that a user is pursuing, such that that information can be presented in a clear, coherent context. Ultimately, it will assist users in the task of reasoning systematically about entities of interest, hypotheses and facts about those entities, and the likely explanations of and consequences of those hypotheses and facts. The resulting compilations are not just static shoeboxes for facts; they support a computing environment that dynamically recommends actions for researchers to consider. A compilation of facts about a particular conference, for example, might include all relevant submission dates, location, organizers, conference topic, and second-order information such as hotels found in that location. This allows Cyc to suggest

appropriate actions at each stage, and even to initiate actions on the user's behalf. In the long term, the goal of this work is to build infrastructure that assists the general capability of the system to manage an information-gathering task, by providing the means for a user to ask specific questions via natural language. Below is an idealized use case pertaining to the example of scientific conferences:

1. *Upon encountering the expression "the proceedings of the ILP 2004 conference" in a scholarly context, the QA system tries to determine that ILP is a conference series, held in (at least) the year 2004, which publishes formal proceedings.*
2. *Based on this information, the information-gathering system suggests that it might be worth finding out where the conference was held and what the main theme of the conference was. This in turn may lead to questions and/or inferences about who participated and what papers were presented. All such queries are optional – the user can ignore them all if he or she wishes.*
3. *If the user does choose to pursue one of these, the system may suggest several alternate paths, each consisting of several subtasks. For example, suppose the researcher selects a query about whether it is likely that inductive logic programming is the main topic of the conference. One way to handle this query is to search recent articles linking inductive logic programming to known researchers in this field, infer who might have presented at such a conference, and search citation sites for evidence that one or more such persons did indeed present at the conference in question.*
4. *The subtasks then decompose into finding researchers in a particular field, searching citation records for papers involving either a set of those individuals or one of them with a very unusual name, etc.*

Making such suggestions would be of enormous value, and stretches the limits of current knowledge-based systems technology. The next step – which stretches the limits of current natural language parsing and understanding technology – is for the NL system to automatically read through the online documents and extract these subtask answers, after which they can be combined logically to produce an answer. An important feature of the information-gathering system is its *Fact Sheet facility,* which provides a framework for organizing and managing information about entities and events of interest. The formal representations used in the Fact Sheets allow for easy sharing of information across users, and for automated queries to be run against the information in the Fact Sheets. Some of the information in Fact Sheets can be gathered and verified automatically, before being presented to a user (Schneider et al. 2005) [17]. Included in Fact Sheets is meta-data about the provenance (both which document a fact came from, and who interpreted the document) of pieces of information in the Fact Sheet (see Figure 4).

As implemented, the initial step is to gather facts about an entity. This could be an entity that a researcher is specifically interested in, or an entity which Cyc has determined, based on what it knows about the user's interests, would likely be relevant to the task. The system first determines that entity's type. If the entity is already known to Cyc, type data will be extracted from the KB; if not, the system searches using the entity's name (currently it searches using Google), gathering sentences from the returned documents that mention the entity. To obtain a coarse typing – sorting into the kinds of categories one would expect from a traditional Information Extraction system such as FASTUS (Hobbs et al. 1997) [7] or ALEMBIC (Day et al. 1997) [6] – the sentences are run through third-party named-entity recognizers (Klein et al. 2003) [8], (Prager et al. 2000) [15]. Once a rough typing is

obtained (e.g. the entity has been determined to be a person, or a place), syntactic patterns in the retrieved sentences are analyzed in order to refine that typing. This identifies, for example, appositive phrases such as "___, a German pharmaceuticals researcher"; these phrases are then interpreted semantically using Cyc's lexicon.

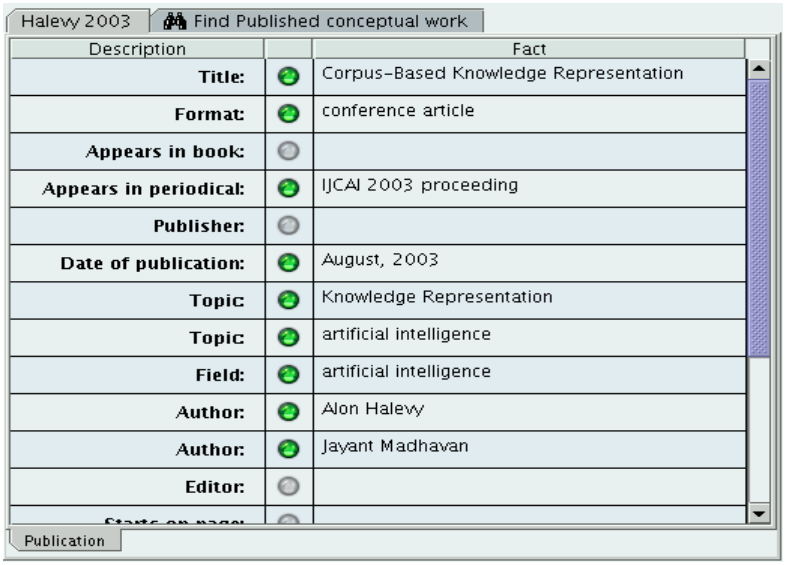

Fig. 4. Information about a specific paper. Relevant KB content is generated into English and displayed in an editable format. The "green light" metaphor for each assertion shows that there are no consistency issues with the information displayed.

Once an entity has been typed, the next step is to determine which kinds of facts should be gathered for that entity. There are three reasons why a fact might be relevant for research: (1) a user requests it; (2) the system is aware that it is an appropriate type of information (e.g. because an ontologist asserted that it is an appropriate type of information for a particular type of entity, or because knowing that fact will trigger interesting inferences); and (3) it is a type of information that is commonly known for that type of entity. When performing automatically-guided fact gathering about a particular entity, the Cyc system finds out which facts are relevant by consulting existing Fact Sheets, which were created (some manually and some automatically) on the basis of reasons (2) and (3). Next, the system constructs search strings suitable for use by an information retrieval engine. If the search engine finds results, the portions of the resulting sentences that correspond to the blanks in the search strings are semantically analyzed, and the results substituted into the variable position(s) in the original CycL query. Before actually asserting the resulting formulas into the knowledge base, Cyc attempts to verify the proposed facts, using KB consistency checks and additional corpus checks. Finally, verified facts are added to the Fact Sheet, along with meta-information about their sources. (This approach to knowledge acquisition is described more fully in the discussion of learning in Cyc in section 4.2.2.)

After preliminary testing, we believe that this system shows substantial promise as a tool that researchers can use to gather and manage information. Because a formal representation underlies the data stored in the system, others can readily reuse its knowledge (unlike text documents), and automatic updates made by one user can be automatically disseminated to other users. We expect that both the fact-gathering ability and the verification methods will improve as we extend and refine our initial solutions to these problems.

Depending on the particular domain of interest, the fact-gathering system produces results that have a precision level between 50% and 70% (Matuszek et al.; Schneider et al. 2005) [12,17]. The level of fact acquisition is lower; these same tests show that the system finds the sought-after information anywhere from 8% to about 20% of the time. This relatively low number is not surprising, given that the system is asking about entities that in many cases are not well known (e.g., it would not be surprising for no web page to list the organizing committee for a departmental colloquium series). Additionally, the techniques currently being employed are very shallow (requiring exact string matches); the addition of more sophisticated NLP methods should allow for substantially better retrieval rates.

4 Learning Within Cyc

In the early days of the web, the "Ask Jeeves" site offered a very exciting prospect: millions of people would pose questions, and hundreds of librarians would find the answers and add them to the site. Over time, the system would become increasingly comprehensive, gradually accumulating the knowledge it would need to answer almost any question.

Nearly a decade after the founding of "Ask Jeeves" in 1996, it has become clear that one of the biggest hurdles facing projects that try to store information with only minimal understanding of the background concept is one of is *combinatorics*. The following question is typical, reasonable, and should be answerable; but it will probably never be repeated this century: "What time tracking programs for a Palm Pilot can track on quarter hour intervals, track at least 10 projects, and synchronize with Excel?" Tens of thousands of librarians could not hope to anticipate even a fraction of questions that users asked. The situation is even worse in scientific applications where almost *all* questions are likely to be unique.

Given the availability of a large knowledge base, and the ability to augment that knowledge base using information obtained from the web, it should be possible to successfully direct the learning of new knowledge that in turn improves the system's ability to anticipate and answer the needs of scientists. We have hypothesized two ways for Cyc to use what it already knows to "bias" or guide this learning: (1) guiding the gathering of facts needed to answer queries (or sub-queries), automatically or via directed dialogues with knowledge workers; and (2) guiding the induction of new knowledge from what is already known.

Performing machine learning over the knowledge centralized in the Cyc knowledge base (or accessible from that knowledge base, as in the integration of relational databases) requires the application of a variety of techniques that are normally used separately. Cyc's inferential capabilities allow the automatic or semi-

automatic conjecture of facts, the collection of new facts, and the production of hypothetical rules and scripts that can then be generalized, tightened, corrected, and used. This iterative process starts with the low-hanging fruit of implicational rules, and can be expanded to acquisition of more and more complicated knowledge structures. Cyc is a good testing ground for this use of knowledge collection and induction; its large pre-existing corpus of facts and rules provides models that ease the process of fact acquisition and rule induction.

4.1 Learning in Cyc: Goals

Current work in the Cyc Project is taking steps towards beating the combinatorics problem mentioned earlier in relation to Ask Jeeves, creating something like an Ask Jeeves that can directly answer questions. There are two parts to this proposed solution:

1. Use a large knowledge-based system, relying on Cyc, as a representational interlingua, so that *n* fragments of information from *m* sources can arithmetically and logically combine into answers for novel questions. To do this, the *meaning* of the *n* fragments must be available to Cyc. In the specific case of automated (but imperfect) extraction of desired facts from text corpora, *n* can be very large.
2. Apply a combination of statistical, deductive, and inductive techniques, to get Cyc to ***learn*** to answer questions. Some of this learning will be 100% automatic, such as inducing rules from specific facts and generalizing existing rules. Some of this learning will be semi-automatic, such as automatically identifying a small number of specific "pivotal" questions for human users to answer.

Consider even the relatively simple question of booking travel for a conference. To manage this task, the system needs to be able to go out to the web and determine the location of the conference; it needs to know things about the person who is doing the traveling; it needs to be aware of when the travel must occur, which requires sophisticated temporal understanding; and it should have heuristics (rules of good judgment) that specify that for uncertain dates it should volunteer the argument it used to make its guess, but for true specific dates, it should only show the argument if prodded.

The first stages of the necessary learning can be subdivided into two categories: *fact gathering,* the collection of basic knowledge that translates into simple CycL assertions, and *rule production,* which allows the system to conclude to higher-level knowledge given those facts. Fact gathering currently targets ground-level assertions, either at the instance level or the type level, while rule production has focused on generating rules from large sets of ground facts via induction.

Instance-Level Fact: `(isa Lenat ArtificialIntelligenceResearcher)`

Type-Level Fact: `(genls ArtificialIntelligenceResearcher ComputerScientist)`

An AI researcher is a kind of computer scientist.

Rule:
```
(implies
    (and
```

```
        (isa ?CONF Conference)
        (topicOfIndividual ?CONF ArtificialIntelligence)
        (presenter ?CONF ?PER))
    (isa ?PER ArtificialIntelligenceResearcher))
```

By default, presenters at AI conferences are AI researchers.

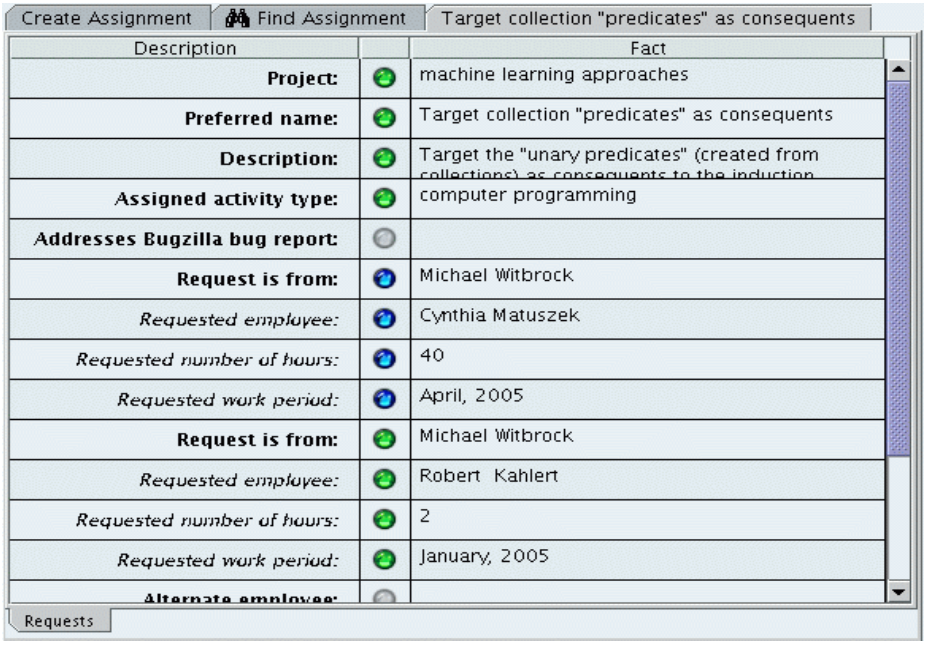

Fig. 5. Factivore used for task assignments. In this form, the employee Cynthia Matuszek is being assigned to the task of importing and exporting CycL collections to an induction system as unary predicates, to which they are logically equivalent. The different colors of "lights" indicate assertions that are already made (green) or in the process of being made (blue).

4.2 Gathering Facts

4.2.1 Gathering Facts Via User Interaction

In previous work, Cycorp developed a general knowledge acquisition interface called the Factivore (Witbrock et al. 2005) [23]. This component of the Cyc system uses templates, represented in CycL and stored in the Cyc KB, to describe "forms" which, if filled in, in English, to the system's satisfaction, will result in appropriately formalized assertions appearing in the KB.

This allows users who are not trained ontologists to enter information that might not be possible to obtain otherwise, such as the creation of task assignments for a research group. It is ultimately necessary for a scientist to describe that information somehow – it cannot be retrieved from a web page or calculated from implication rules (at least until AI technology has become substantially more sophisticated!). The least intrusive behavior an assistant can display is to remind a researcher of the need

to provide such instructions to a team, and to provide a mechanism that ensures that it can be done with minimal effort expended on tool use.

For the Factivore's initial deployment, ontologists asserted the underlying templates into the KB. While this somewhat painstaking process was reasonable in that context, given the relatively small number of entity types to be represented, and the large number of those instances, the situation with an ambient research assistant is quite different. Depending on task exigencies, new entities of *any* type may need to be represented in the KB. This longer-term goal was the reason for storing the templates in CycL in the KB. It is now possible for Cyc to autonomously produce templates for any type of entity for which Cyc knows a few instances. Over time, it is intended that the system will learn to produce and display these forms in context, eliciting the information required to handle the current task properly, storing it in the KB for future use as learned information, and providing an ever-growing base of "ground facts" to use for rule induction (and improved Factivore form design).

4.2.2 Gathering Facts Via Web Search

Given the existing infrastructure in Cyc for representing types and parsing simple natural language sentences, combining these capabilities for targeting knowledge collection from the World Wide Web (e.g. via Google) is a natural mechanism for gathering knowledge (Matuszek et al. 2005). The advantages of targeting the web, rather than a human user, are compelling – the user, who need not answer questions or struggle with the system when the information cannot be understood, need expend no cognitive effort. Given the sea of web pages that can be accessed using tools like Google (Brin and Page 1998) [2], Cyc can simply skip pages it cannot understand, with a substantial likelihood of finding alternate web pages that provide the answer. The implementation of this approach decomposes into several subtasks:

- **Generating strings for web searches** and web page counts from partial logical sentences and logical terms using Cyc's natural language lexicon:

  Given a partially-bound CycL phrase, such as:
  `(occupation Lenat ?WHAT)`
  Return one or more search strings:
  "Lenat has been a ____"
  "Doug Lenat has been a ____"
  "Lenat is a ____"

- **Using Google's public API** to have Cyc access web pages in their ranked order.

- **Identifying a match** in the web page, then interpreting that match into a formal representation that is consistent with the constraints imposed by the logical sentence:

  Finding the string "Lenat has been a professor of …" in a web page
  Interpreting into the term `#$Professor`
  Substituting into the original CycL: `(occupation Lenat Professor)`

- **Eliminating bad interpretations** or erroneous websites by semantically verifying the claim against the KB. Cyc would never suggest `(occupation Lenat PrimeNumber)`, (which might be incorrectly parsed from the sentence "Lenat is a prime example..."), because `PrimeNumber` is a member of a class known to be disjoint with occupation type.

- **Verifying the correctness** of the parse interpretation by using Google to search for the natural-language interpretation of the assertion we have now constructed, and performing a second round of search over those strings, which allows us to reject interpretations that are a result of bad parses, or are simply too broad. One of the strings "Lenat is a professor" or "Lenat has been a professor" produces results, whereas "Doug Lenat has been a paramedic" – resulting from a mis-parse of a page about machine translation – does not.

- **Asserting the CycL Sentence** into the KB, once it has passed all automatic verification tests.

Targeting the automatic acquisition of simple sentences minimizes the difficulty inherent in generating and parsing complex natural language constructs. As well, simple facts are more likely to be described in a single sentence on the web that can be found and parsed. In initial trials, the search and verification process produced sentences that were correct, according to human review, approximately 50% of the time. While this is not adequate for the needs of a fully autonomous system, when combined with the validation provided by a user who is treating the information provided as a suggestion, it has the potential to be very useful, c.f. (Witbrock et al. 2005) [23].

4.2.3 Using Inference to Generate Facts: Abduction

An efficient approach to generating candidate sentences relies on using the *abductive* reasoning capabilities of the Cyc inference engine. In the literature on AI and logic programming, an abduction is generally understood to be an argument of the form:

$$\{Ga \wedge [\forall x\ (\ Fx \rightarrow Gx\ )]\} \rightarrow_{abduced} Fa$$

Thus abduction is generally performed as deduction-in-reverse[1]: working backwards from rules that have the desired result in the *antecedent* from *consequents* that are known to be true and relevant.[2] If new abductive rules are desired, they can be

[1] Logically, abduction and deduction-in-reverse are distinguishable (Mayer and Pirri 1996); in practice, this form of abduction is productive when applied to a large knowledge base containing many deductive rules and an inference harness designed to take advantage of those rules.

[2] More formally, finding candidate hypotheses in Cyc is done by querying the inference engine about the truth of observation *o* (the seed query), which will be a CycL formula containing unbound variables, and then treating the generated deductive proof attempts as candidate explanations of *o*. If we have a seed of the form `Ga`, and a rule of the form `(∀x)(Fx → Gx)`, `Ga` is passed to Cyc's inferential query mechanism. With one transformation step the tactician finds `(∀x)(Fx → Gx)` and transforms the problem to `Fa`. It then attempts to prove `Fa`. Whether or not `Fa` is true, the inference process stores the problem `Fa`. `Fa` is thus generated as a problem in a deductive inference's search for a proof of `Ga`. Such generated problems are also abduced hypotheses for `Ga`. Accordingly, the converse forms of rules that use `implies` are used to generate abductive hypotheses.

written with `implies`, and they will thereby be available for both deduction and abduction.

As an example: given the previously introduced rule, a new *seed query* (for which hypothetical bindings are desired), and pre-existing knowledge about researchers and conferences:

- **Seed query:** `(presenter AAAI-05 ?WHO)`
- **Assertion:** `(isa Witbrock ArtificialIntelligenceResearcher)`
- **Assertion:** `(isa AAAI-05 Conference)`
- **Assertion:**(topicOfIndividual AAAI-05 ArtificialIntelligence)
- **Rule:** (implies

```
(and
    (isa ?CONF Conference)
    (topicOfIndividual ?CONF
 ArtificialIntelligence)
    (presenter ?CONF ?PER))
 (isa ?PER ArtificialIntelligenceResearcher))
```

An abductive reasoning process matches the assertion to the consequent of the rule, and generates the following defeasible hypothesis:

- **Candidate Hypothesis:** `(presenter AAAI-05 Witbrock)`

The candidate sentences suggested are checked (via inference and specialized well-formedness-checking modules) for consistency with the current knowledge in the KB. They are then evaluated for *inferential productivity*, i.e., the degree to which they are likely to be subsumed by the antecedent of some rule. Any highly inferentially productive statement has the potential to disproportionately increase the knowledge that can be deduced. When actually performing abduction of this sort, either web verification systems or a human knowledge worker evaluates each sentence for truth and plausibility (Matuszek et al. 2005, Witbrock et al. 2005) [12,23]. The results of this evaluation will be used to improve automated filtering of conjectured sentences, as well as to determine whether those sentences should be immediately applied to tasks where assistance is desirable.

4.3 Rule Induction

The successful acquisition of a large, consistently structured data set that is well connected to the existing Cyc knowledge base is possible, based on abductive hypotheses, web search, and gathering facts from users via the Factivore. The logical next step in learning is the *induction* of rules.[3] Induction of implication rules is a process of arguing that a rule is a possible explanation underlying a set of correlated

[3] Consistency of form minimizes the inference-based data transformation necessary to perform successful induction, and data that is thoroughly connected to the rest of the knowledge base allows for the automatic formation of better and more descriptive rules.

facts (e.g., if every person that is known to be a mother is a female person, perhaps it can be concluded that all mothers are female). An example rule induced from data might be:

$$[Pa \wedge Pb \wedge Qa \wedge Qb \wedge Rb \wedge Sa \wedge \neg(Sb)]$$
$$\rightarrow_{induced}$$
$$\{\forall x\ [Px \wedge Qx \wedge \neg(Rx) \rightarrow Sx]\}$$

Typical inputs to induction (Srinivasan et al. 2003) [21,22] – a list of first-order facts constructed using a set of related predicates – can be readily produced by simple queries to Cyc's inference engine over some set of predicates, meaning that running induction continuously oveor large sets of knowledge found in the Cyc KB is limited only by the speed of the induction process. Induction, like abduction, is not guaranteed to produce *sound* assertions; some are true (accurate when applied to future data points that may be introduced), while others are true only over the training set and must be revised when other, conflicting data points are introduced.

Induction was tested in the project management/personnel management domain in the Cyc Knowledge Base, using a combination of the FOIL 6 (Quinlan and Cameron-Jones 1993) [16] and ALEPH systems (Srinivasan 2001) [21], over an initial set of 10 predicates, which had a combined extent of 1,680 assertions. The predicates used for induction were:

`primarySupervisor`	`projectManagers`
`projectParticipants`	`projectTasks`
`assignedEffortPercent`	`requestedEffortPercent`
`participantIn`	

- `(assignedEffortPercent TASK AGT)` means that the IntelligentAgent `AGT` has been assigned the task `TASK`.
- `(primarySupervisor AGT1 AGT2)` means that `AGT2` is the default directingAgent in any work-related action in which `AGT1` is a deliberateParticipant.
- `(participantIn AGENT GATHERING)` means that `AGENT` is an intentional participant in the SocialGathering, `GATHERING`.
- `(projectManagers PROJECT MANAGER)` means that `MANAGER` manages `PROJECT`, an instance of Project.
- `(projectParticipants PROJECT AGENT)` means that the intelligentAgent `AGENT` participates in (usually, receives tasks from) the project `PROJECT`.
- `(projectTasks PROJECT TASK)` means that `TASK` is a task sanctioned by the Project `PROJECT`. Typically, this means that `TASK` is a sub-task of the overarching task of 'completing `PROJECT`.'
- `(requestedEffortPercent TASK AGENT)` means that an authorized person (usually the project manager of some parent task of `TASK`) requests that `AGENT` work on `TASK`.

A human reviewer then evaluated the rules. For the initial test runs, sets of predicates from a variety of different domains were selected, and approximately 150 rules were produced over those sets. Two human reviewers independently evaluated all rules using a tool specific to that task (Witbrock et al. 2005) [23]. On average, 7.5% of the automatically produced rules were considered good enough to assert into the KB immediately; 35% more were found to need only quick editing to be assertible. The most common editing required was the deletion of extraneous clauses in the antecedent. On average, it took reviewers 7 hours to review 150 rules, meaning a production rate of ~10 rules per hour when minor editing was allowed, about three times the rate at which a senior Cycorp ontologist can produce rules by hand. Inter-evaluator agreement is approximately 90%. This suggests that, even with human evaluation of rules, induction over ontologized data has the potential to be a comparatively efficient way to discover correlations in at least some bodies of assertions. Some examples of rules that were produced by this system follow:

```
(implies
     (and
           (projectParticipants ?PROJECT ?AGENT)
           (primarySupervisor ?AGT2 ?AGENT)
           (projectManagers ?PROJECT ?AGT2))
     (primaryProject ?AGT2 ?PROJECT))
```

If someone is a participant in some particular project, someone else is that person's primary supervisor; and the second person is the manager of the project; then the first project is probably that person's primary project.

```
(implies
     (and
           (primaryProject ?AGT ?PROJECT)
           (projectTasks ?PROJECT ?TASK)
           (requestedEffortPercent ?TASK ?AGT ?PRC))
     (assignedEffortPercent ?TASK ?AGT ?PRC))
```

If someone's primary project has a task that must be performed; and some percentage of that person's time is requested for that task; then they will probably be assigned at that percentage to that task.

While these relationships may seem obvious to a person (such as a human assistant), they represent fairly deep reasoning, of the sort that an ambient assistant system must handle transparently and quickly. Not only must the system perform induction successfully; it is necessary to reason about the nature of the terms found in the data set. For example, if this information is being drawn via NL parsing from some corpus, such as a status report, the concept of "participant" may not be explicitly defined. People may be described as "participants," "manager [of]," or "reviewer," and the generalization of those to the category "project participant" must be made by either a human, or by a system with broad world knowledge.

5 Developing an Ambient Cyc Application

This section outlines current efforts to deploy Cyc to assist with information-gathering tasks by human researchers, analysts, ontologists, and others working in information-intensive fields. These efforts focus largely on building an application framework that extends the Cyc system. This framework derives its core capabilities from the Cyc knowledge base and inference engine, which enable automated, context-sensitive reasoning over multi-source data. The features most relevant to the construction of an ambient assistant are highlighted in this section: anticipation of a user's information needs, and hypothesis generation and tracking.

5.1 Anticipation of Information Needs

Cyc incorporates two classes of models of users and analytic processes (illustrated in Figure 6). The first, object-level, class is based on the nature of the objects of inquiry (e.g. models of structured interpersonal transactions guide analyses of social networks). The body of prior knowledge (such as knowledge of people, objects, places, actions, and real-world event types) in the Cyc knowledge base underpins models of object-driven analytical processes. The second meta-level class of models derives from principles of effective research techniques. For example, one such default principle is: "If there are **n** *prima facie* equally good alternatives to pursue at a particular stage of analysis, it is equally important to pursue each of the **n** alternatives". The first class of models will be used to anticipate users' information needs and to proactively fulfill them. The second class of models will be used to build increasingly accurate individual and composite models of researchers and their needs.

5.1.1 Models of Object-Driven Analysis Processes and Anticipation of Information Needs

Speaking at the level of highest generality, any analytical inquiry about something involves: (1) investigating the thing's parts (using a very broad construal of "parts"), and (2) investigating how the thing bears significant relationships to other things. Cyc understands this feature of analysis by virtue of having a very general script[4] that says "To learn about a thing, one must find out about its significant parts and its significant external relations". (Siegel et al. 2005) [19]

Once realized, an assistant using this analytical schema will know what sorts of fact are relevant when studying certain broad classes of things. For example, when studying an event, such as travel, it will know the importance of understanding what type of event it is in order to situate it in the context of other events – in particular, sub-events and super-events of a focal event like a scholarly conference. Similarly,

[4] Scripts are type-level representations of events that have some sort of complexity in terms of either the temporal ordering of their sub-event types (or "scenes"), or in terms of the types of things that typically play roles in their scenes. As such, scripts could enable Cyc to recognize complex actions based on matching sensor data to the patterns of scripts, and they could enable Cyc to perform complex actions according to a default prescribed action sequence.

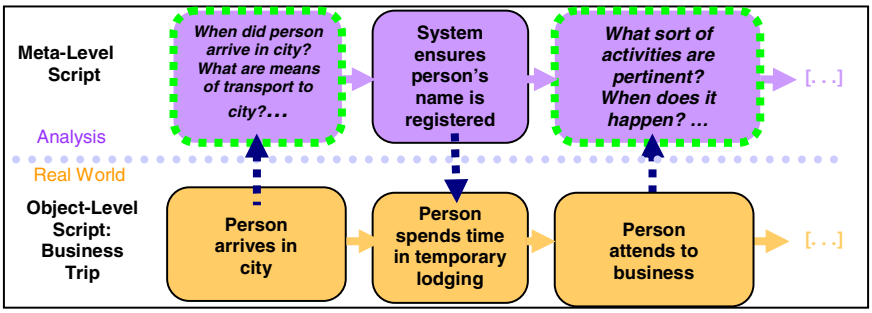

Fig. 6. Analysis Procedure for Business Travel, illustrating the relation between the process of analyzing the behavior of a person and a script that describes the typical progression of a certain class of human behaviors. On the analysis side, finding a person's name in a hotel register corresponds to a behavior in the real world: a person spending time in a temporary lodging. Spending time in a temporary lodging is typically associated with both travel and a purpose for the travel, each of which are features of the object of analysis, and each of which suggest natural behaviors for an assistant system to pursue.

when studying a physical thing, it will try to understand what kinds of things it can be a part of, and what kinds of parts it has. Armed with general principles of this sort, such an assistant will not need to be explicitly told that if an individual is traveling for a conference, it is important that that person be registered in a hotel somewhere in the vicinity of the conference in the time frame in which the conference is taking place. This knowledge will result from Cyc's understanding of location and co-location, temporality, event occurrence, the meaning of "attendance," and a host of other factors. Like a good assistant (and unlike much current software), these details can be recognized as related and relevant – not only without the researcher spelling out every detail of his or her needs, but without any explicit interaction between the researcher and the system. By recognizing *classes* of questions as significant, Cyc has motivation to instigate lines of inquiry that will reveal the information (e.g., by accessing transportation schedules), thereby having this information ready for the researcher when the time comes to book travel.

5.1.2 Ideal Analysis Processes, Script Learning and the Detection of Bias

The successful realization of Cyc as an ambient assistant relies on having large amounts of data showing what human agents with extensive knowledge requirements actually *do* as they work. The development cycle currently takes advantage of a transaction capturing environment, which allows researchers to capture and record activities occurring during the course of real tasks, including the stream of tasks, queries, documents examined, and reports produced. To the extent possible, it also captures working or draft documents produced. Actions are captured in very fine detail, often down to the keystroke. By interpreting transaction logs and comparing logged behaviors to descriptions of ideal processes, it is anticipated that Cyc will learn new strategies, expose some single-occurrence errors, and detect patterns that indicate important systematic biases on the part of individual users. This knowledge

can then be generalized to broader and broader scripts of human behavior, resulting in knowledge of general scripts that specify how particular tools (such as IR engines, Cyc's own hypothesis generation tools, and other analytical tools) should be used.

For example, a script might be developed over time that says that, when a user is searching the Internet for information about work related to statistical approaches to data retrieval, it is important to execute searches that ultimately incorporate all major relevant research sites – CiteSeer, the ACM portal (http://portal.acm.org), etc. The assistant will then be able to detect when a user has deviated from ideal behavior, and suggest a search specifically including a missing site. Discrepancies such as this – unique errors on the part of the researcher – have quick remedies, such as a set of links to IR searches that have already been performed on the remaining relevant terms.

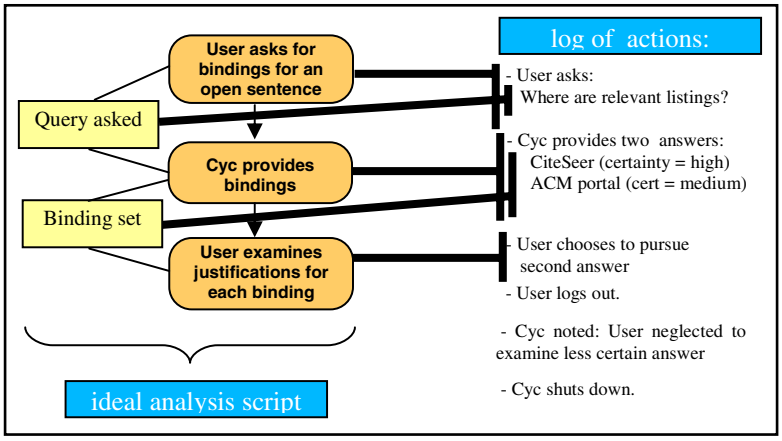

Fig. 7. Planned user action tracking. The right-hand side of the diagram displays a list of the actions Cyc and the user take; on the left-hand side is the idealized script. First, the user asks a query and Cyc provides two answers. Next, the user views one of the answers (the more certain one), and then logs out. Cyc logs the fact that the user neglected to pursue one answer, and detects that the behavior logged differs from the learned case.

Ultimately, it will be possible to detect the systematic biases of specific users. By building long logs of the ways behaviors diverge from the ideal in each case, it will be possible to help scientists recognize their own systematic shortcomings, and ultimately make up for those shortcomings. Of course, when there is a discrepancy between Cyc's view of an ideal strategy and a researcher's behavior, it will not always be the user who is at fault. In some cases, a discrepancy will arise because Cyc does not yet know about a particular approach, or simply does not recognize the correct termination case; perhaps the user in the diagram above was not doing a broad literature search, but rather trying to find a specific piece of work, and was satisfied before exhausting the search possibilities.

In some cases, Cyc will potentially be able to *learn* that what may appear to be evidence of bias is actually evidence of effective use of tacit expert knowledge. This would allow the system to distinguish omissions from situations where the user has

yet to perform an action, or already knows what the result of the behavior would be. This kind of sophisticated user model would provide a notable advantage over systems that produce intrusive warnings stating the obvious.

5.2 Hypothesis Generation and Tracking

One way in which an assistant can assist a researcher to obtain knowledge is by hypothesizing, in advance, what approaches a researcher would benefit from pursuing. Such hypotheses need to be presented in a context that makes it clear why they should be of interest, and – in order to minimize the cognitive and time cost of tool use – it will be necessary to provide easy one-click tools that enable users to assess hypotheses based on existing documents, and then to confirm or deny the hypotheses. Several such tools have already been developed for a variety of uses within Cyc, ranging from simple sentence reviewers (and more complex rule reviewers) to predicate populators that allow an untrained user to select from a checklist of possible values for an argument position (Witbrock et al. 2005) [23].

Hypotheses are generated, again, by allowing abductive inference over Cyc's common sense and domain-relevant rules. For example, consider an inference initiated during an attempt to find answers for the question: "Who has done work relevant to our current inductive approach to machine learning of rules?" This question can be answered in a fashion analogous to that of the example presented in Section 4.2.3: working backwards from rules that have the desired result in the *antecedent* from *consequents* that are known to be correct and relevant (Siegel et al. 2005) [19].

In addition, work is progressing on a project designed to extend Cyc's capacity to develop hypothetical supports for focused queries into a richer capability called "scenario generation". This scenario generation will be initiated by a description of a "seed event", which users will be able to describe using the Factivore. For example, the system could be tasked with determining in what ways an upcoming conference trip could result in scheduling difficulties. The system will then generate scenarios under which the result could occur – for example, another conference that is likely to be of interest is scheduled for the same block of time, or the person who should attend may be on vacation, or – less probably – the conference might be cancelled. Cyc would generate a range of scenarios ranked in terms of relevance, using rule-clustering technology that ensures reasoning with the most salient rules first. Having generated hypotheses via abduction, Cyc refines the hypotheses, by adding useful information to them via deduction. The next round of abductions will branch again, producing a tree structure among scenario contexts.

6 Conclusion

A true AI can fully manage a variety of tasks, not just simplify them or make them faster. Like a good human assistant, a fully realized AI assistant will make tasks silently vanish: you will never be aware that they even needed to happen. Failing that, such an assistant will bring things to your attention only when you must do something about them. Human assistants use common sense in determining which tasks may simply be carried out and then dismissed; on which tasks the supervisor must be kept

apprised of progress; and when to alert the supervisor that some serious roadblock has been encountered. By drawing on a body of knowledge about scientific research, and about such common-sense concepts as what sorts of things motivate agents to act, how time works, and what is or is not a difficult problem, Cyc will be able to carry a part of the cognitive burden of day-to-day scientific research overhead tasks.

Accomplishing this will depend on having many of the same characteristics that are the hallmarks of a human ambient assistant: flexibility, availability, ease of communication, the ability to learn from a variety of sources, and the ability to correlate learned information and learn higher-level information about expectations and priorities. Furthermore, an ambient assistant – one that is part of the environment in which a researcher lives, and can reason about every aspect of the interrelated events and factors that make up the day-to-day life of a researcher – has the potential to do much more, and much less obtrusively, than a human assistant.

Simply developing many different kinds of special-purpose software will not accomplish this extraordinarily challenging vision; it requires common sense, the ability to learn, and deep integration with the tools and tasks a researcher uses daily. While we would hardly claim that Cyc contains all of the information and abilities to achieve an unobtrusive, useful ambient assistant, we have mapped out in this paper a number of abilities that, when combined together, should prove sufficient to provide the basis for such an ambient assistant. Cyc has made substantial strides in each of these areas, and also has an active research program that should enable ever more progress in the future.

Acknowledgements

Research as wide-ranging and ambitious as the Cyc Project and the many components described in this paper would not be possible without the support of a number of persons and agencies, including AFRL, ARDA, DARPA, NSF, NIST, and others; corporations and research agencies such as Google, which generously allows us to access their API for research such as this; and many other researchers in the field of artificial intelligence who have guided, assisted, and criticized our progress over time.

References

[1] Bollacker, K., Lawrence, S., Giles C.L., 1998. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. *Proceedings of the 2nd International Conference on Autonomous Agents*, New York, pp. 116-123.

[2] Brin, S., Page, L., 1998. Anatomy of a large-scale hypertextual search engine. *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, pp 107-117.

[3] Buchanan, B.G., Feigenbaum, E.A., 1978. DENDRAL and Meta-DENDRAL: Their applications dimension. Journal of Artificial Intelligence 11, 5-24.

[4] Burns, K., Davis., A., 1990. Building and maintaining a semantically adequate lexicon using Cyc, in: Viegas, E. (Ed.), Breadth and Depth of Semantic Lexicons, Kluwer, Dordrecht.

[5] Copeland, B.J., 2005. Artificial intelligence. Encyclopædia Britannica Online, http://www.britannica.com/eb/article?tocId=9009711.

[6] Day, D., Aberdeen, J., Hirschman, L., Kozierok, R., Robinson P., Vilain, M., 1997. Mixed initiative development of language processing systems. *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, D.C., pp. 348-355.

[7] Hobbs, J., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., Tyson, M., 1997. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text, in: Roche, E., Schabes, Y. (Eds.), *Finite State Devices for Natural Language Processing*, MIT Press, Cambridge, Massachusetts, pp. 383-406.

[8] Klein, D, Smarr, J., Nguyen, H., Manning, C., 2003. Named entity recognition with character-level models. *Proceeedings of the Seventh Conference on Natural Language Learning*, pp. 180-183.

[9] Lenat, D. B., Borning, A., McDonald, D., Taylor, C., Weyer, S., 1983. Knoesphere: building expert systems with encyclopedic knowledge. *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 1, pp. 167-169.

[10] Lenat, D. B., Guha, R.V., 1990. Building Large Knowledge Based Systems. Addison Wesley, Reading, Massachusetts.

[11] Masters, J., Güngördü, Z., 2003. Structured knowledge source integration: A progress report. Integration of Knowledge Intensive Multiagent Systems, Cambridge, Massachusetts.

[12] Matuszek, C., Witbrock, M., Kahlert, R.C., Cabral, J., Schneider, D., Shah, P., Lenat, D., 2005. Searching for common sense: Populating Cyc™ from the Web. *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005).* Pittsburgh, PA. In Press.

[13] Mayer, M.C., Pirri, F., 1996. Abduction is not deduction-in-reverse. Journal of the IGPL, 4(1): 1-14, 1996.

[14] McCallum, A., Nigam, K., Rennie, J., Seymore, K., 1999. A machine learning approach to building domain-specific search engines. *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999)*, pp. 662–667.

[15] Prager, J., Brown, E., Coden, A., Radev, D., 2000. Question answering by predictive annotation. *Proceedings of the 23rd SIGIR Conference*, pp. 184-191.

[16] Quinlan, R.J., Cameron-Jones, R.M., 1993. FOIL: A midterm report. *Proceedings of the European Conference on Machine Learning,* 667: 3-20.

[17] Schneider, D., Matuszek, C., Shah, P., Kahlert, R.C., Baxter, D., Cabral, J., Witbrock, M., Lenat, D., 2005. Gathering and managing facts for intelligence analysis. *Proceedings of the 2005 Conference on Intelligence Analysis: Methods and Tools*, McLean, VA.

[18] Shortliffe, E., 1976. Computer-based Medical Consultations: MYCIN. New York: American Elsevier.

[19] Siegel, N., Shepard, C.B., Cabral, J., Witbrock. M.J., 2005. Hypothesis generation and evidence assembly for intelligence analysis: Cycorp's Noöscape application. *Proceedings of the 2005 Conference on Intelligence Analysis: Methods and Tools*, McLean, VA.

[20] Sleator, D. D., Temperley, D., 1991. Parsing English with a Link Grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

[21] Srinivasan, A. The Aleph Manual. University of Oxford, http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html.

[22] Srinivasan, A., King, R.D., Bain, M.E., 2003. An empirical study of the use of relevance information in inductive logic programming. Journal of Machine Learning Research 4(7): 369-383.

[23] Witbrock, M., Matuszek, C., Brusseau, A., Kahlert, R.C., Fraser, C.B., Lenat D., 2005. Knowledge begets knowledge: Steps towards assisted knowledge acquisition in Cyc. *Proceedings of the AAAI 2005 Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC)*, Stanford, CA.