Interactive Natural Language Explanations of Cyc Inferences

David Baxter, Blake Shepard, Nick Siegel, Benjamin Gottesman, Dave Schneider

Cycorp, Inc., 3721 Executive Center Drive, Suite 100, Austin, TX 78731
{bshepard, baxter, nsiegel, bgottesm, daves}@cyc.com

Abstract

This paper describes the inference explanation capabilities of Cyc, a logical reasoning system that includes a huge "commonsense" knowledge base and an inference engine that supports both question answering and hypothesis generation. Cyc allows the user to compose queries by means of English templates, and tries to find answers via deductive reasoning. If deduction is fruitless Cyc resorts to abduction, filling in missing pieces of logical arguments with plausible conjectures to obtain provisional answers. Cyc presents its answers and chains of reasoning to the user in English, provides drill-down to external source references whenever possible, and reasons about its own proofs to determine optimal ways of presenting them to the user. When a chain of reasoning relies on conjectures introduced via abduction, the user can interact with the inference explanation to confirm or deny the abduced supports. These capabilities are grounded in the integration of Cyc's natural language components with the knowledge base and inference engine, and in Cyc's capacity to maintain an explicit in-memory record of the facts, rules, and calculations used to produce successful proofs during inference.

Introduction: The Cyc System

The user of a rule-based system will find, ideally, that the system responds to queries by returning informative and possibly surprising answers. Depending on the task at hand, the user might want to see how the system has reasoned to obtain its results. Indeed, the more important the task and informative or surprising the results, the more we might expect the user to want to assess the soundness of the reasoning that produced the results. The utility of a rule-based system for high-impact, money- or life-saving, mixed initiative applications, therefore, is often proportionate to the transparency of its explanations.

The challenges to be surmounted in order to clearly present complex proofs grow as the sheer number of relevant facts and rules in the knowledge base (KB) increases, and as the variety of inference patterns used to generate proofs multiplies. These challenges are especially extreme for Cyc, a knowledge-based reasoning system that includes a huge KB, an inference engine, Cyc's Semantic Knowledge Source Integration (SKSI) facility, and several natural language (NL) components. The knowledge stored in the Cyc KB is represented in a formal language, CycL, which subsumes and extends the predicate calculus of first-order logic. The KB contains 2.7M assertions (facts and rules) that interrelate 300K concepts, encoding both "commonsense" knowledge (consensus reality) and specialized domain knowledge. Cyc's millions of assertions are distributed over thousands of explicitly represented and semantically significant logical contexts, or *microtheories*.

Cyc's inference engine can produce a single inference result (answer set) by combining information from a myriad of contexts, and by employing an armory of defeasible reasoning procedures, both sound (resolution, class subsumption, argumentation) and unsound (abduction). The inference engine consists of a "harness" that invokes any of 800 pattern-specific reasoning modules, and falls back on general resolution-based theorem proving as a last resort. Each module implements an efficient inference procedure for a common type of problem, such as a technique for calculating the transitive closure of a binary predicate. One significant suite of modules implements abductive inference, and thereby enables Cyc to introduce new conjectures (hypothetical statements) during inference. The SKSI facility allows Cyc to access the structured contents of external data sources of information, such as relational databases and web pages.[1] Once the structure and meaning of an external knowledge source have been described by assertions in the KB, the inference engine can automatically create specialized inference modules for the source and can treat it, thenceforth, as a virtual extension of the KB.

An explicit in-memory record of completed proofs underpins Cyc's capacity to explain its own reasoning. Cyc can reason about the features of it own proofs when the user asks to view them, determining which statements are the most salient, which statements are too trivial to show, and which statements are most appropriate for the different levels of detail supported in the display.

Cyc's NL components include modules for parsing English to CycL, template-based generation of English from CycL, and discourse management, along with a general-purpose lexicon and several smaller, domain-specific lexi-

[1] For more information about the development of SKSI, see [Masters and Güngördü 2003]. For more information about Cyc, past and present, see [Lenat and Guha 1990], [Lenat 1995], and the whitepapers and other material available on Cycorp's web site: http://www.cyc.com .

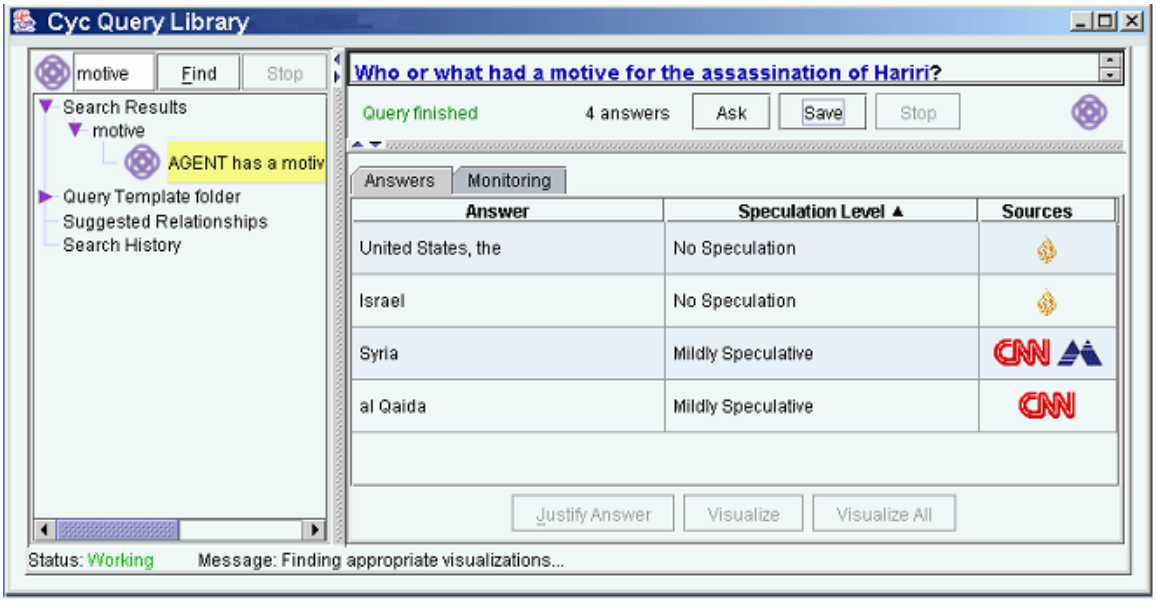

Figure 1: Cyc's Query Library (QL) interface displaying a query and answers.

cons. Nearly all of Cyc's grammatical and lexical knowledge is represented in CycL and stored in the KB, thus making it available for use by the inference engine. Cyc supports several text-based and graphical user interfaces. Most notable for the focus of this paper is the Query Library (QL) interface, which allows the user to construct queries, submit them to the inference engine, view the resulting answers (Figure 1), and obtain a detailed display of the proofs that support each answer (Figures 2 and 3). The QL employs Cyc's natural language generation capabilities to display queries, answers, and proofs (also referred to as *justifications*) in English, rather than in CycL.

An Inference Example

To illustrate the complexity of a typical Cyc justification, here we consider an example of interest to intelligence analysts that draws on Cyc's knowledge of recent events in the Middle East. Let us suppose that the date is February 16, 2005, and that an analyst (user) has constructed the following English query with the QL interface:

[EQ1] Who had a motive for the assassination of Rafik Hariri?

The corresponding CycL version of this query, and the form in which it must be posed to the inference engine, is as follows:

```
[Q1] (agentHasMotiveForAction ?WHO
       TerroristAttack-February-14-2005-Beirut)
```

This query asks the inference engine to find terms that can be substituted for the variable `?WHO` to yield a closed logical sentence. When the analyst poses the query [Q1], the inference engine tries to find answers by using heuristic search, drawing on data and rules stored in the KB, and possibly in other sources accessible through SKSI. After a few seconds, the inference engine obtains the following substitution terms (CycL values) for `?WHO`, each of which constitutes an answer to [Q1]: `UnitedStatesOfAmerica`, `Syria`, `Israel`, and `AlQaida`. The rest of this section describes part of the chain of reasoning that produces one of the answers, `Syria`.

First, the inference engine selects rule [R1], since the predicate in the consequent of [R1] matches the predicate in [Q1]:

```
[R1] (implies
       (and
         (adoptionTypeOfNormByAgent
           ?POLICY ?ADOPTER ?ADOPT-TYPE)
         (policyForAgent ?POLICY ?ADOPTER)
         (isa ?AGENT IntelligentAgent)
         (negativeVestedInterest ?AGENT ?POLICY)
         (normProponents ?POLICY ?VICTIM)
         (preventsFromPlayingRoleInType
           ?ACT
           ?VICTIM
           keyParticipants
           ?ADOPT-TYPE))
       (agentHasMotiveForAction ?AGENT ?ACT))
```

[R1] means that an agent (*i.e.*, a person or an organization) has a motive for a particular action if the agent opposes a particular policy, and if the action prevents some proponent of the policy from being a key participant in its adoption.

Once the inference engine has selected [R1] as a relevant rule, it then transforms the original query into a new subquery formed from the antecedent of [R1]:

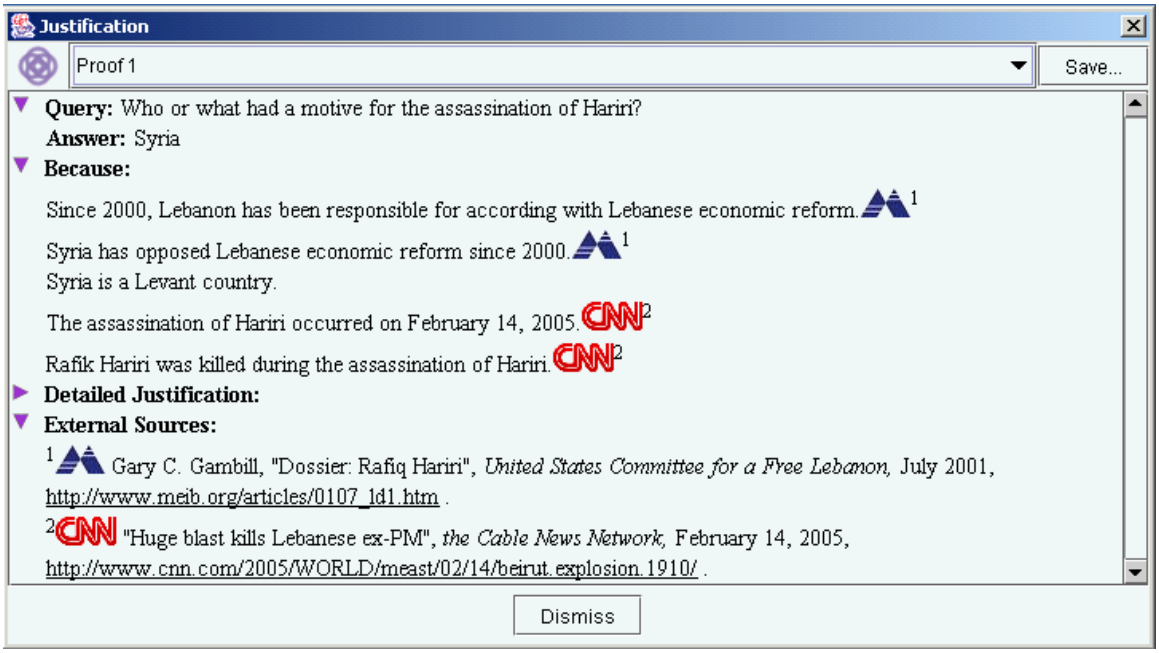

Figure 2: The justification display panel as it first appears to the user.

```
[Q2] (and
       (adoptionTypeOfNormByAgent
         ?POLICY ?ADOPTER ?ADOPT-TYPE)
       (policyForAgent ?POLICY ?ADOPTER)
       (isa ?AGENT IntelligentAgent)
       (negativeVestedInterest ?AGENT ?POLICY)
       (normProponents ?POLICY ?VICTIM)
       (preventsFromPlayingRoleInType
         TerroristAttack-February-14-2005-Beirut
         ?VICTIM
         keyParticipants
         ?ADOPT-TYPE))
```

The inference engine next considers how to solve [Q2]. At this point it could, in theory, address any one of the individual conjuncts. The most efficient tactic, however, would be to proceed with the conjunct that is likely to be most quickly solved without first having solved any of the other conjuncts. After analyzing [Q2], the inference engine determines that it would probably be most efficient to solve this sub-problem first:

```
[Q3] (adoptionTypeOfNormByAgent
       ?POLICY ?ADOPTER ?ADOPT-TYPE)
```

The inference engine's choice is based on its knowledge that the KB and all other accessible sources of data happen to contain comparatively few assertions formed with the predicate `adoptionTypeOfNormByAgent`. Indeed, simple look-up suffices for the inference engine to find this relevant fact in the KB:

```
[F1]  (adoptionTypeOfNormByAgent
        LebaneseEconomicReform
        Lebanon
        (AdoptionTypeOfNormByAgentFn
          LebaneseEconomicreform
          Lebanon))
```

This statement means that the nation-state Lebanon is the agent responsible for its own economic reform. The inference engine still must solve the remainder of [Q2], which, restricted by the values found in [F1], is:

```
[Q4] (and
       (policyForAgent
         LebaneseEconomicReform Lebanon)
       (isa ?AGENT IntelligentAgent)
       (negativeVestedInterest
         ?AGENT LebaneseEconomicReform)
       (normProponents
         LebaneseEconomicReform ?VICTIM)
       (preventsFromPlayingRoleInType
         TerroristAttack-February-14-2005-Beirut
         ?VICTIM
         keyParticipants
         (AdoptionTypeOfNormByAgentFn
           LebaneseEconomicreform
           Lebanon)))
```

After analyzing the structural properties of [Q4], the inference engine determines that this conjunctive query is now best approached as three independent problems between which no remaining variables are shared:

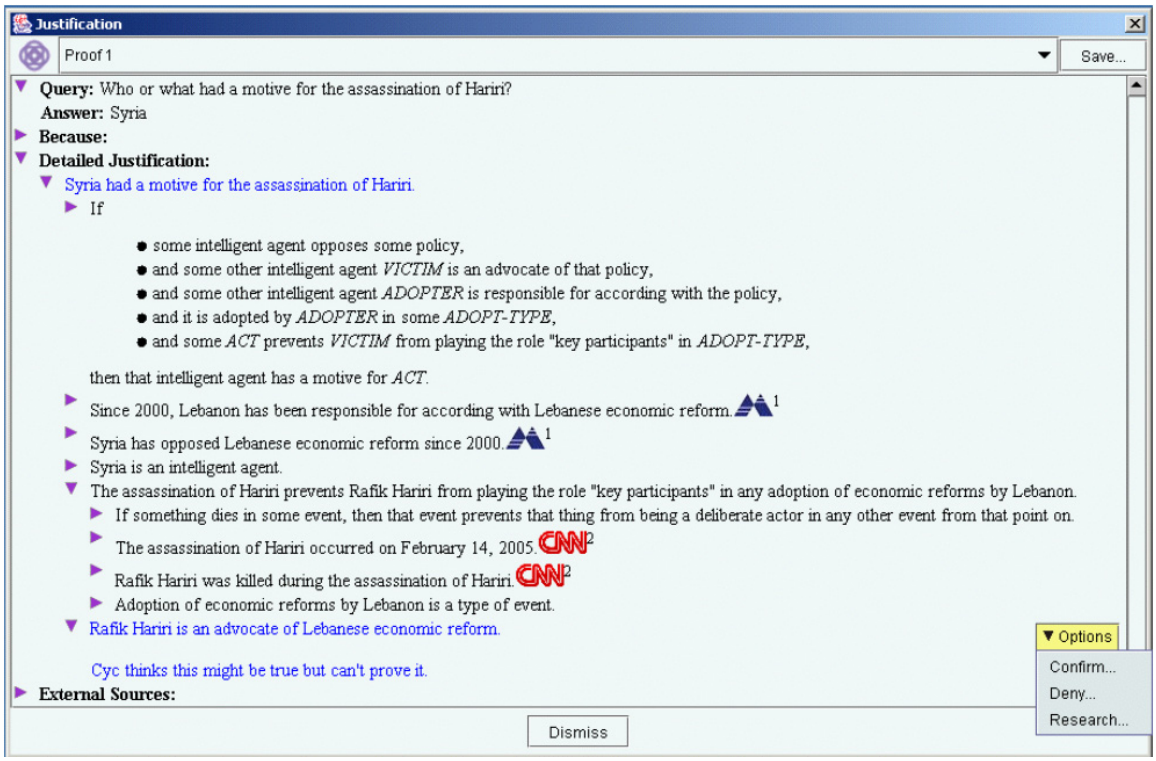

Figure 3: The justification display panel with the "Detailed Justification" section partially expanded.

```
[Q5] (policyForAgent
       LebaneseEconomicReform Lebanon)

[Q6] (and
       (isa ?AGENT IntelligentAgent)
       (negativeVestedInterest
           ?AGENT LebaneseEconomicReform))

[Q7] (and
      (normProponents
        LebaneseEconomicReform ?VICTIM)
      (preventsFromPlayingRoleInType
        TerroristAttack-February-14-2005-Beirut
        ?VICTIM
        keyParticipants
        (AdoptionTypeOfNormByAgentFn
          LebaneseEconomicreform
          Lebanon)))
```

It is possible to prove [Q5] via simple look-up, because this fact is directly asserted in the KB:

```
[F2] (policyForAgent
       LebaneseEconomicReform Lebanon)
```

The inference engine is also able to prove deductively that the term `Syria` is a valid substitution for `?AGENT` in [Q6]. That is, the inference engine is able to prove that Syria is an intelligent agent with a negative vested interest in Lebanese economic reform. It solves the latter part of the problem first, determining that Syria is opposed to Lebanese economic reform by finding this assertion in the KB:

```
[F3] (negativeVestedInterest
       Syria LebaneseEconomicReform))
```

The KB also contains this assertion:

```
[F4] (isa Syria LevantCountry)
```

The inference engine uses [F4] along with its class subsumption reasoning modules to prove that Syria is an intelligent agent. These modules allow the inference engine to quickly conclude from class subsumption assertions already in the KB that a Levant country is a country, a country is a political entity, a political entity is a geographical agent, a geographical agent is a multi-individual agent, and, finally, a multi-individual agent is an intelligent agent.

To solve [Q7], the inference engine splits it into these two sub-problems:

```
[Q8] (normProponents
       LebaneseEconomicReform ?VICTIM)

[Q9] (preventsFromPlayingRoleInType
      TerroristAttack-February-14-2005-Beirut
      ?VICTIM
      keyParticipants
      (AdoptionTypeOfNormByAgentFn
```

```
        LebaneseEconomicReform
        Lebanon))
```

It then performs a heuristic analysis to determine the relative cost of attempting to solve [Q8] before [Q9], and *vice versa*. As it happens, the inference engine can easily prove [Q9] because the KB contains the following forward rule (*i.e.*, all the rule's dependents are eagerly cached in the KB):

```
[R2] (implies
       (and
         (dateOfEvent ?EVENT ?DATE)
         (ist-Asserted ?MT
           (organismKilled ?EVENT ?ORGANISM)))
       (ist
         (MtSpace ?MT
           (MtTimeWithGranularityDimFn
             (IntervalStartedByFn ?DATE)
             TimePoint))
         (preventsFromPlayingRoleInType
           ?EVENT
           ?ORGANISM
           deliberateActors
           Event)))
```

[R2] means that if an organism is killed in an event, then forever after this event the organism is prevented from being a deliberate actor in any other (future) event. An immediately derived consequence of this rule is the following:

```
[F5] (preventsFromPlayingRoleInType
       TerroristAttack-February-14-2005-Beirut
       RafikHariri
       deliberateActors
       Event)))
```

The quaternary predicate `preventsFromPlayingRoleInType` is transitive through the collection-specialization relation in its fourth argument position, and is also transitive through the predicate-specialization relation in its third argument position. These constraints are expressed by the following assertions:

```
[F6] (transitiveViaArgInverse
       preventsFromPlayingRoleInType genls 4)

[F7] (transitiveViaArgInverse
       preventsFromPlayingRoleInType genlPreds 3)
```

Now, because the collection denoted by the non-atomic, functionally created term

```
(AdoptionTypeOfNormByAgentFn
        LebaneseEconomicReform Lebanon)
```

is a specialization of `Event`, and because the predicate `keyParticipants` is a specialization of the predicate `deliberateActors`, the inference engine can draw on [F5], together with specialized reasoning modules designed for predicates about which `transitiveViaArgInverse` assertions have been made, to prove this assertion:

```
[F8] (preventsFromPlayingRoleInType
       TerroristAttack-February-14-2005-Beirut
       RafikHariri
       keyParticipants
       (AdoptionTypeOfNormByAgentFn
         LebaneseEconomicReform
         Lebanon))
```

With [F8] as a solution for [Q9], Cyc is able to pose the following restricted version of [Q8] as the next pertinent problem to solve:

```
[Q10] (normProponents
         LebaneseEconomicReform RafikHariri)
```

At the time of the analyst's query, no knowledge sources accessible to the inference engine contain information about Rafik Hariri's advocacy of economic reform in Lebanon. Because no deductive tactic is successful at this point in inference, the inference engine falls back on abduction and posits a hypothetical support for [Q10]:

```
[H1] (normProponents
       LebaneseEconomicReform RafikHariri)
```

The introduction of conjectures generated through abductive reasoning is a tactic of last resort, employed by the inference engine when all purely deductive means of obtaining an answer have failed. This tactic becomes available to the inference engine only when the user specifies, at query time, that this sort of unsound but possibly fruitful reasoning is permitted. The use of abductive reasoning at this point in the proof allows the inference engine to return `Syria` as a plausible but strictly provisional answer to the top-level query, [Q1]. The validity of this answer hinges on the truth or falsity of [H1].

The chains of reasoning that enable the inference engine to return `Israel`, `UnitedStatesOfAmerica`, and `AlQaida` as additional answers to [Q1] rely on different facts and rules, but are equally complex. In the remaining sections of this paper, we will describe how Cyc reasons about the complex sets of supports generated in inference to produce understandable explanations of the inference engine's results.

The Display of Explanations

The QL interface provides several types of information for each answer returned by the inference engine. The first type appears as an extra column labeled "Speculation Level" in the tabular display of answers. This column is shown only when the user has permitted Cyc to resort, if necessary, to hypothetical (abductive) reasoning during a query session. The second type is information about the provenance (sources) of the facts used during inference. This informa-

tion is initially conveyed via source icons associated with each inference answer (Figure 1). The third type consists of detailed, interactive NL renditions of the proofs for each answer. This information is displayed in *justification display panels* that are launched when the user selects an answer and clicks the "Justify Answer" button (Figures 2 and 3).

Grouping Answers by "Speculation Level"

Cyc uses abduction to generate conjectures (hypothetical statements) during reasoning, which is one method for discovering plausible but uncertain explanations [Paul 1993]. As mentioned in the inference example above, Cyc's inference engine performs abduction by invoking reasoning modules that posit truth values for sentences, or bindings for variables, when these cannot be found via deductive search. Figure 1 shows the answers produced by the inference engine sorted into display groups by a "Speculation Level" metric. For answers supported by proofs with no conjectures, the displayed speculation value is "No Speculation". For all other answers, the speculation level is a relative measure determined by the ratio of conjectural supports (assumptions obtained via abduction) to non-conjectural supports (statements obtained via deduction and look-up) in each answer's least abductive proof. (A given answer may be justified by multiple proofs, some of which may require more use of abduction than others.) Thus, for example, the abductive proof for the answer "Syria" is deemed only "Mildly Speculative" because, as indicated in the inference example section above, the proof for the answer "Syria" contains only one abductive support among several deductive supports.

While primitive in its current implementation, the display of speculation level values gives the user an indication of the number of guesses the system has made to obtain a given answer. The ranking of abductive inference results is possible because the inference engine keeps track of all proof components in the interest, ultimately, of marshalling them to explain proofs to the user.

Displaying Source Icons with Inference Answers

When presenting inference answers, the QL also displays icons that denote information sources. The motivation for showing the icons along with the initial presentation of the answers is to provide the user with a simple, mnemonic means of "credibility assessment" based information provenance. For example, in Figure 1, "the United States" and "Israel" are included among the answers displayed for the query "Who or what had a motive for the murder of Rafik Hariri?", and these answers are marked as being provable with "No Speculation". Here, Cyc's reasoning is based on information for which the ultimate provenance is Syrian state-run newspapers, reprinted by the Al-Jazeera news agency. The icons shown in the "Sources" column, on the far right of the QL interface's answer table, indicate immediately that the information used to obtain the answers "Israel" and "the United States" was distributed by Al-Jazeera. Prominent display of provenance information with each answer makes the inference results more immediately comprehensible to a user who may already be overloaded with information. The user need not memorize the meanings of all the icons, because simply hovering over any icon with the mouse pointer causes a pop-up window to display the full source citation. The QL interface shows as many source icons as are pertinent to the answer displayed in the corresponding table row.

To find the sources for a given answer, the inference engine iterates over all of the assertions that contribute to (*i.e.*, are part of) the justification (proof) for the answer. (An answer may be supported by multiple, independent proofs.) For each such assertion $\mathbf{a}_i$, the inference engine executes the following query:

```
[Q11] (sourceOfAssertion-NonTrivial ?SOURCE a_i)
```

If the inference engine finds a binding for `?SOURCE`, then that binding is considered a contributing, citable source for the answer supported by $\mathbf{a}_i$. Whenever users add knowledge to the KB, they are actively encouraged by Cyc's knowledge-entry interfaces to include explicit source information that can be retrieved by the query shown above. The types of entities that may be referenced as sources include people, news articles, web pages, books, magazines, encyclopedias, almanacs, databases, and many others.

To retrieve the correct icon for a specific source, such as an individual journal article, the inference engine tries to find an entity that can be considered the source's provenance (both in the sense of "place of origin", and in the sense of "indictor of credibility/reliability"). In general, provenance entities are distributors, repositories, or points of access for individual information source objects, such as articles or web pages. Examples of provenance entities include organizations such as the United States Department of Energy, CNN, Reuters, and Al-Jazeera. Cyc does not directly associate every newspaper article, encyclopedia article, or web page with a distinctive icon. Rather, when possible, the more general provenance entities are associated with icons via assertions in the KB. Thus, given that an assertion in the KB is explicitly linked to a particular encyclopedia entry or newspaper article, Cyc must then call on inference to determine the provenance (publisher, issuer, originating entity) of the assertion. Specifically, to retrieve an appropriate display icon for a given information object $<\texttt{SOURCE}_i>$, the inference engine tries to find bindings for `?PROVENANCE` and `?ICON` by running the following query:

```
[Q12] (and
        (tinyIconTermImagePathname
          ?PROVENANCE
          ?ICON)
      (or
        (issuerOfCW <SOURCE> ?PROVENANCE)
        (subWorks ?PROVENANCE <SOURCE>)
        (thereExists ?EDITION
          (thereExists ?DATE
```

```
(and
  (subWorks ?EDITION <SOURCE>)
  (editionOfPeriodicalByDate
    ?PROVENANCE
    ?DATE
    ?EDITION))))
(publisher <SOURCE> ?PROVENANCE)))
```

In other words, the inference engine performs a heuristic search to determine whether the source has as a provenance entity a known "issuer" (*e.g.*, a quote with a known speaker), or a containing work of which it is a part (*e.g.*, an article in an encyclopedia), or an agent (person or organization) that is its publisher.

If the KB contains no icon reference (URI) for the particular provenance entity retrieved, then the inference engine determines the type of the specific source in question (*e.g.*, web page, book, journal article), and the QL interface displays a default icon for that type of source (*e.g.*, a picture of a generic book, or a picture of a generic magazine).

The Justification Display Panel

Clicking on the "Justify Answer" button in the QL interface launches a justification display panel for the highlighted answer. This panel shows NL versions of CycL proofs that have been constructed by the inference engine. Each proof comprises an entire chain of reasoning that supports the answer, and a single answer may be supported by multiple proofs. A pull-down list at the top of the panel allows users to toggle the display between different proofs.

The justification display panel contains four sections, labeled: "Query/Answer," "Because," "Detailed Justification," and "External Sources". Each section can be expanded or collapsed by clicking on the small triangular icon to its left. When the panel is first displayed, all of the sections except "Detailed Justification" are fully expanded. Figure 2 illustrates what the justification display panel looks like when it is opened to display the proof for the answer "Syria" in our inference example.

The "Query/Answer" Section. This section of the justification display panel provides context for the other sections by showing an NL version of the query posed, together with an NL version of the answer the justification supports. In the example in Figure 2, the query portion of this section reads: "Who or what had a motive for the murder of Rafik Hariri?" The justification displayed is for the answer "Syria". In each of these cases (and, indeed, in every case where NL is produced in the QL interface), the NL is generated heuristically from CycL by the Cyc system, using paraphrase techniques that we will describe below in the section entitled "NL Paraphrases of CycL Assertions".

The "Because" Section. This section displays ground statements (closed, non-rule supports) in the proof that are either facts obtained from the KB or other sources accessible to the inference engine, or conjectures generated via abduction. This section is supposed to provide the user with a minimal explanation: a quickly comprehensible list of the basic facts used to support the inference answer. When an external published source is the origin of a ground statement displayed in this section, an icon for the source appears to the right of the assertion, and footnote number links the assertion to a full citation in the "External Sources" section of the display.

The "Detailed Justification" Section. This section is supposed to provide the user with rich explanations that include sufficient information to review, understand, and validate Cyc's proofs (Figure 3). When the user expands this section of the justification panel to reveal its content, it provides most of the detail of a complete *proof tree*, which is a complex Cyc datastructure that contains the entire justification for an inference result, including sub-proofs of intermediate results. Cyc stores proof trees in memory until explicitly told to destroy them, so they are readily available for rendering in NL in this display.

When first expanded by the user, the detailed justification section displays the proof tree at one level deep. The first level of depth is constituted by the first rule used to transform the top-level problem (thus, in our example, [R1]) together with all of the supports used to prove the conjuncts of the antecedent of the first rule (from our example, supports [F1], [F2], [F3], [F4], [F5] and [H1]). When completely expanded by the user, the "Detailed Justification" section reveals almost the complete structure of the proof tree for the relevant inference results, one level at a time, including

Figure 4: What the "Because" section would look like if no supports in the proof tree were suppressed. "Duh!" arrows point at trivial facts that users do not need to see. "Huh?" arrows point at supports that would only confuse users if included in the display.

rules and omitting only uninteresting details.

When preparing to show a detailed proof, Cyc deems certain classes of supports uninteresting and, therefore, worthy of suppression. Suppressed supports are not available to the user through drill-down: they are not displayed even when the "Detailed Justification" section is fully expanded.

The justification display panels of the QL interface routinely suppress three kinds of inference supports. The first two kinds encompass supports that are judged too trivial for users to see, and those that are judged too confusing for users to see. The third kind consists of "structural links" in proofs.

Figure 4 illustrates what the "Because" section of the justification display panel would look like if the interface did not suppress trivial and confusing supports. Without such suppression, the effect on the "Detailed Justification" section of the display would be even more extreme, with myriad trivial and confusing sentences.

The process of determining at runtime which elements of a proof tree to expose in the "Detailed Justification" section, and which elements to completely suppress, is guided by inference using knowledge stored in the KB. For example, Cyc deems uninteresting any support that the inference engine can prove is trivial for justification paraphrase. Unary predicates such as

`ruleTrivialForJustificationParaphrase`

(which applies to whole assertions) and

`predTrivialForJustificationParaphrase`

(which applies to predicates) encode this sort of knowledge. For example, assertions that state trivial translations between synonymous CycL expressions are tagged with `ruleTrivialForJustificationParaphrase`:

```
[F9] (ruleTrivialForJustificationParaphrase
       (implies
         (trueSentence ?SENT)
         (sentenceTruth ?SENT Truth)))
```

Also, if a term mentioned in a support in a proof tree is labeled in the KB with the unary predicate `keIrrelevant`, then that support is suppressed in the justification display. For example, in Figure 4, which shows what the display would look like with no supports suppressed, one of the confusing supports displayed is: "Argument 4 of `#$preventsFromPlayingRoleInType` is transitive with respect to the inverse of `#$genls`", and another is "Argument 3 of `#$preventsFromPlayingRoleInType` is transitive with respect to the inverse of `#$genlPreds`". These supports are the NL renderings of [F6] and [F7] in the example inference. Because the KB contains this assertion,

```
[F10] (keIrrelevant transitiveViaArgInverse)
```

[F6] and [F7] are omitted from the standard justification display panel in both the "Because" and the "Detailed Justification" sections.

Also not included in the NL justification display are structural links that indicate the order in which the inference engine has solved the sub-problems of a proof. These structural links are useful to Cyc's developers for debugging the process of inference. The detailed justification presented in the justification panel, however, is intended to allow end users to examinine the proofs that result from inference, rather than to debug the inference process itself. Structural links encode procedural information that is not relevant for assessing a proof's validity, and therefore is generally not pertinent to the task-directed interests and concerns of most end-users. For instance, a structural link contained in the full justification of our inference example, above, captures the fact that the inference engine deemed it appropriate to split problem [Q4] into the independent problems [Q5], [Q6] and [Q7]. Interfaces to the inference engine designed for debugging inference must make this structural link explicit, but it is suppressed in the justification display panel.

In the "Detailed Justification" section, the ground statement, "Rafik Hariri is a proponent of Lebanese economic reform", is displayed in blue font to indicate that it is a conjecture obtained by the inference engine via abduction. Other ground facts in that section are displayed in black font. In addition, the abduced conjecture has a drop-down menu to its right that reads "Options". If the user clicks on this menu, it expands to reveal three choices: "Confirm", "Deny", or "Research". Selecting "Research" causes a complex Boolean web query generated from the CycL version of the abduced conjecture to be passed to an online search engine (such as AltaVista or Google), and also causes a web browser to display the search results. Selecting "Confirm" causes Cyc to enter the assertion into the Cyc KB. If the justification contains no additional hypothetical statements it will, thenceforth, be considered a valid deductive proof for the answer, and the blue font of the displayed conjecture will change to black. Selecting "Deny" causes the statement to be entered into the KB with an explicit truth-value of false, after which the displayed justification will disappear, since it can no longer possibly be a valid proof for the answer.

Any supports that depend on conjectures will also be displayed in blue, which makes it easy for the user to distinguish those parts of the explanation that rely on abduction from those that are purely deductive. Thus, in the "Detailed Justification" section in Figure 4, both the sentence "Rafik Hariri is a proponent of Lebanese economic reform" and the sentence "Syria had a motive for the assassination of Hariri" display in blue font, because the latter depends for its support on the former, which has only abductive support.

The "Detailed Justification" section of the justification panel presents the results obtained by specialized inference modules, but does not automatically show the sub-proofs constructed by the modules. We adopted this tactic after finding, in general, that the results of specialized Cyc inference modules are intuitive and rarely beg for user drill-

down. In most cases, however, Cyc can be prompted by the user to provide arbitrary levels of detail. For example, in Figure 3 the interface displays this fact as part of the detailed justification: "Syria is an intelligent agent". This fact is not explicitly asserted in the KB, but rather is calculated based on the explicit assertion that "Syria is a Levant country" and the further fact that in Cyc's hierarchy of classes, "Levant country" is a specialization of "intelligent agent". Indeed, if a user clicks on the triangle to the left of the claim that Syria is an intelligent agent, Cyc will show these supports: "Syria is a Levant country" and "A Levant country is a kind of intelligent agent". The latter claim can be further expanded to show the explicit chain of subsumptions that enables Cyc to conclude that a Levant country is a kind of intelligent agent.

NL Paraphrases of CycL Assertions

To illustrate how Cyc generates English from the CycL structures that are the actual nodes of a proof tree, we will explain step-by-step how Cyc paraphrases the following rule from our example inference:

```
[R1] (implies
       (and
         (adoptionTypeOfNormByAgent
           ?POLICY ?ADOPTER ?ADOPT-TYPE)
         (policyForAgent ?POLICY ?ADOPTER)
         (isa ?AGENT IntelligentAgent)
         (negativeVestedInterest ?AGENT ?POLICY)
         (normProponents ?POLICY ?VICTIM)
         (preventsFromPlayingRoleInType
           ?ACT
           ?VICTIM
           keyParticipants
           ?ADOPT-TYPE))
       (agentHasMotiveForAction ?AGENT ?ACT))
```

Ultimately, Cyc paraphrases [R1] as:

[PR1] "If

- **some intelligent agent opposes some policy;**
- **a second intelligent agent, `?VICTIM`, is an advocate of the policy;**
- **a third intelligent agent, `?ADOPTER`, is responsible for according with the policy;**
- **the policy is adopted by `?ADOPTER` in some `?ADOPT-TYPE`; and**
- **some `?ACT` prevents `?VICTIM` from playing the role "key participants" in any `?ADOPT-TYPE`;**

then the first intelligent agent has a motive for `?ACT`."

To paraphrase [R1], Cyc proceeds through the following sequence of steps. First, it checks to see if a special paraphrase template has been written for [R1] as a whole. In this case, there is no such rule-specific template, so Cyc proceeds to the next step, which is to determine the specific type of object it is paraphrasing.

In the case of [R1] Cyc recognizes that it needs to paraphrase an *assertion* (as opposed to, *e.g.*, a single first-order reified term, such as `Syria`). Next, Cyc makes explicit any hitherto implicit quantification of the assertion. All rules in Cyc are implicitly universally quantified, and [R1] is a rule. Making explicit the implicit universal quantifiers in [R1] therefore yields [R1′]:

```
[R1′]
(forAll ?AGENT
  (forAll ?POLICY
    (forAll ?VICTIM
      (forAll ?ADOPTER
        (forAll ?ADOPT-TYPE
          (forAll ?ACT
            (implies
              (and
                (adoptionTypeOfNormByAgent
                  ?POLICY
                  ?ADOPTER
                  ?ADOPT-TYPE)
                (policyForAgent ?POLICY ?ADOPTER)
                (isa ?AGENT IntelligentAgent)
                (negativeVestedInterest
                  ?AGENT
                  ?POLICY)
                (normProponents
                  ?POLICY
                  ?VICTIM)
                (preventsFromPlayingRoleInType
                  ?ACT
                  ?VICTIM
                  keyParticipants
                  ?ADOPT-TYPE))
              (agentHasMotiveForAction
                ?AGENT
                ?ACT)))))))))
```

The next step is for Cyc to identify the type of [R1′], which in this case is "universal sentence", since the outermost quantifier of [R1′] is a universal quantifier. The first action Cyc performs on universal sentences is to determine the most specific types of things over which their universally quantified variables can range. These types are determined by the semantic argument constraints that pertain to the predicates in the rule. For example, the KB contains the following constraint:

```
[F11] (arg2Isa policyForAgent IntelligentAgent)
```

This assertion means that any term occurring as the second argument to the predicate `policyForAgent` must be an instance of `IntelligentAgent`.

Upon examining all the relevant semantic argument constraints in [R1′], Cyc registers `?AGENT`, `?ADOPTER`, and `?VICTIM` as most narrowly constrained to instances of `IntelligentAgent`, `?ADOPT-TYPE` as constrained to instances of `AdoptingANorm`, `?POLICY` as constrained to

instances of `Policy`, and `?ACT` as constrained to instances of `Action`.

Cyc next registers all the variables in [R1′] as universally quantified, and removes the clauses from [R1′] that serve merely to type the variables, since Cyc has just registered variable types independently. Thus, all of the quantifiers and the sole `isa` clause in [R1′] are removed, yielding [R1′′]:

```
[R1′′] (implies
         (and
           (adoptionTypeOfNormByAgent
             ?POLICY
             ?ADOPTER
             ?ADOPT-TYPE)
           (policyForAgent ?POLICY ?ADOPTER)
           (negativeVestedInterest ?AGENT ?POLICY)
           (normProponents ?POLICY ?VICTIM)
           (preventsFromPlayingRoleInType
             ?ACT
             ?VICTIM
             keyParticipants
             ?ADOPT-TYPE))
         (agentHasMotiveForAction ?AGENT ?ACT))
```

During all this rephrasing (from [R1] to [R1′] to [R1′′]) Cyc's paraphrase code keeps track of how argument positions in the new formula correspond to argument positions in the original formula, since at the end it must be able to determine how individual terms inside the formula were paraphrased.

Next, Cyc tries to use variable-typing clauses to paraphrase variables as typed noun phrases. Thus `?AGENT` (which, recall, has been registered as constrained to instances of `IntelligentAgent`) is now paraphrased as "some intelligent agent", based on these two assertions in the KB:

```
[F11] (multiWordString
          (TheList "intelligent")
          Agent-TheWord
          CountNoun
          IntelligentAgent)

[F12] (singular Agent-TheWord "agent")
```

Although logical quantification is universal, Cyc knows that in the "if" section of an implication universally quantified variables should generally be paraphrased as existentially quantified, thus here generating the English quantifier "some".

The other intelligent agents mentioned in the "if" section of [R1′′] are referred to when they first occur in the paraphrase as "some other intelligent agent `?VICTIM`", and "some other intelligent agent `?ADOPTER`", to emphasize that they are distinct from one another and from `?AGENT`. When they occur subsequently in the paraphrase, they are referred to simply as "`?VICTIM`" and "`?ADOPTER`", because their types have already been mentioned.

Cyc keeps track of the occurrences of each variable in the rule and generates a phrase for the variable that is appropriate to the context in which the variable occurs. For example, `?POLICY` is referred to three times: first as "some policy", then as "that policy", and finally as "the policy". `?AGENT` is referred to in the consequent (the "then" section of the implication) as "that intelligent agent", whereas, as mentioned above, it had been referred to in the "if" section of the implication as "some intelligent agent".

The text surrounding the variables is generated from NL template assertions on the predicates that occur in the initial (0^{th}-argument) positions of each paraphrased sub-sentence. For example, the KB contains this generation template for the predicate `negativeVestedInterest`:

```
[F13] (genTemplate
        negativeVestedInterest
          (ConcatenatePhrasesFn
            (BasicTransitiveSentenceFn
              (TermParaphraseFn-NP :ARG1)
              Oppose-TheWord
              (TermParaphraseFn :ARG2))))
```

This assertion means that any CycL assertion with `negativeVestedInterest` in the initial position should be paraphrased as a basic transitive sentence with "oppose" as the verb, and the terms in the 1^{st}- and 2^{nd}-argument positions as the subject and direct object, respectively, of the verb.

In the final step of paraphrase the paraphrased elements of the rule are assembled into a string:

> If some intelligent agent opposes some policy, some other intelligent agent ?VICTIM is an advocate of that policy, some other intelligent agent ?ADOPTER is responsible for according with the policy, it is adopted by ?ADOPTER in some ?ADOPT-TYPE, and some ?ACT prevents ?VICTIM from playing the role "key participants" in any ?ADOPT-TYPE, then that intelligent agent has a motive for ?ACT.

This string is displayed in the "Detailed Justification" section of the justification display panel for the example inference (Figure 3).

The "External Sources" section. Often, the KB will contain explicit knowledge that specific assertions were derived or extracted from the information found in a named, published source. In this section of the justification panel, footnoted citations link such assertions to the relevant sources. If the footnote text is the title of a web source and Cyc knows the relevant URL, it will be displayed as a "live" link, which, if clicked on by a user, will open the source document with the default web browser.

Related Work

Cyc's interactive justification presentation functionality grew out of a static, one-shot explanation generation system

developed for the HALO project [Friedland *et al.* 2004]. The ability to drill down into different parts of the justification for which greater detail is desired greatly enhances the effectiveness of the current interface over the earlier system, in which explanations – often several pages long for a single chemistry problem – were printed out and judged by human evaluators. An alternative approach, used in the *P.rex* system for explaining mathematical proofs [Fiedler 1999], is to model the user's domain knowledge and predict which parts of the proof will require more detailed explanations.

Rather than displaying an entire proof at once, the *P.rex* system uses a chat-like dialog system, showing one step in the proof at a time, allowing the user to indicate if she has understood that step or would prefer more (or less) detail, and then moving on to the next step. This approach undoubtedly facilitates the system's updating of its user model, but also imposes linearity on the explanation process that leaves the system in control. The user cannot easily see all the top-level supports for the conclusion at once glance, nor can she jump from a "later" part of the proof to an "earlier" part. In Cyc's approach, by contrast, the user decides which parts of the justification to expand or collapse, and can see the entire proof at once, or can focus on parts of it in any order. However, the Cyc system currently does little modeling of the user's knowledge or preferences.

Another system used in the HALO project, described in [Barker *et al.* 2002], used hand-written explanation templates for each rule, specifying a gloss of the rule, a list of "dependent facts" (corresponding generally to the antecedent of the rule), and a template for paraphrasing the conclusion. This system was able to produce very readable results, but imposed an extra burden on rule authors. In the Cyc system, the templates used to paraphrase facts and rules for the QL interface's justifications are the same ones used in all other applications of Cyc's natural language generation capabilities, including Cyc's interfaces for browsing KB content, creating fact sheets, and paraphrasing query sentences. The approach used in the Cyc system ensures that English inference explanations faithfully conform to the actual proofs generated by the inference engine, because the explanations are compositionally generated directly from the underlying CycL of each proof.

One could imagine a hybrid system in which explanation templates are automatically generated by a system like Cyc's justification generation system and then hand-edited for naturalness. A possible problem with this approach would be that pre-formulated templates cannot be modified in the context of a particular proof or user, whereas justifications generated on demand always have that possibility.

Concluding Remarks

The Cyc system provides an interactive interface for displaying understandable, NL explanations of queries, answers, and complex chains of inference. Cyc reasons about the structure and content of its own proofs to filter trivial, confusing, or merely "structural" supports from the explanations provided to users. In addition, Cyc deduces optimal ways of describing and displaying the provenance of supports, the most suitable placement of supports (paraphrases) in explanation interfaces that allow the user to choose between varying levels of detail, and the most appropriate manner of paraphrasing CycL supports in English.

Acknowledgments

Support for development of the capabilities described in this paper was provided by the HALO project, and by ARDA's NIMD and AQUAINT programs.

References

Barker, Ken, *et al.* 2002. A Question-Answering System for AP Chemistry: Assessing KR&R Technologies. *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*. Whistler, 488-497.

Fiedler, Armin 1999. Using a Cognitive Architecture to Plan Dialogs for the Adaptive Explanation of Proofs. In T. Dean, ed., *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 358-363. Morgan Kaufmann, San Francisco, CA.

Friedland, N S., *et al*. 2004. Project Halo: Towards a Digital Aristotle. *AI Magazine* 25(4): 29-47.

Lenat, D. B. and Guha, R.V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley.

Lenat, D. 1995. Steps to Sharing Knowledge. In *Toward Very Large Knowledge Bases*, ed. N.J.I. Mars. IOS Press.

Masters, J. and Güngördü, Z. 2003. Semantic Knowledge Source Integration: A Progress Report. *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS '03)*, 562-566. Piscataway, N.J.: IEEE Press.

Paul, G. 1993. Approaches to Abductive Reasoning: An Overview. *Artificial Intelligence Review* 7:109-152.