# Using an Ontology to Evaluate a Large Rule Based Ontology: Theory and Practice

**William Jarrold**
Department of Educational Psychology
University of Texas at Austin
william.jarrold@alum.mit.edu

### Abstract

This paper surveys theoretical and practical issues associated with ontologies and performance evaluation of intelligent systems. The case is made for parallels between performance evaluation and human psycho-educational evaluation. Important ontological distinctions in testing concepts are reviewed. Finally an actual implemented evaluation system, the Cyc Testing System, is described. Consisting of an ontology and a software system, this system provides evaluative feedback during the development of Cyc, a large rule-based ontology. Four kinds of validation tests are described: challenge tests, regression tests, integrity tests, and rumination based tests. The ontology of these tests and its accompanying software package is intended for use with Cyc, a large rule based system with common sense and natural language capabilities.

## Keywords

ontologies, evaluation.

## Introduction

The role of ontologies in evaluation of intelligent systems has been a topic at prior PerMIS conferences. For example, at PerMIS '02 there was a discussion section titled, "General Discussion Panel 1 - What is the Role of Ontology in Performance Evaluation?" In this paper I first discuss some general issues associated with ontologies and performance evaluation. As a working example of these ideas put into practice, I next describe the Cyc Testing System, an ontology-based system for evaluating Cyc, a large rule based ontology.

## What is an Ontology?

An ontology may be viewed as a system of concepts partially ordered by a subsumption relation. The Linnean Taxonomy of organisms is an example of a simple ontology. Unlike the Linnean system, semantically rich ontologies contain a more diverse set of semantic relations between concepts than a single subsumption relation. Wordnet (http://www.cogsci.princeton.edu/ wn/) is an example of such an ontology which represents the English lexicon. Wordnet contain terms such as *Fruit* and *Apple* which are linked together via the *hypnonym* relationship, one of Wordnet's subsumption relationships. Being a relatively rich ontology, Wordnet also contains other types of relationships such as partonomic relationships relating, for example, *Apple* to *Stem*. Other types of relationships such as synset relationships are used to relate synonymous terms to one another. Richer still is a rule-based ontology. Such an ontology uses rules in addition to a diverse of semantic links as a means of linking concepts together. Cyc, described below, and the Knowledge Machine (http://www.cs.utexas.edu/users/mfkb/km.html) contain examples of this sort of ontology.



## Knowledge Base versus Ontology

The terms knowledge base (KB) and ontology are somewhat interchangeable. For the purposes, of this paper KB will refer to a more concrete entity - a datastructure which is supposed to serve as the actual instantiation of a given ontology. An ontology may be seen as an abstraction of a KB, as a scheme for carving up the world into concepts, relationships, and possibly rules about those concepts.

## Cyc

The points made in this paper are intended for anyone wishing to evaluate rule based ontologies. Although some of the points are made on the basis of general principles, the second half of this paper describes a specific testing system which I created during the last third of my years at Cycorp, from approximately 1997 though March of 2001. This system helped evaluate Cyc, a large rule-based system being constructed at Cycorp (www.cyc.com). As of late in 2001 Cyc contained over 1.2 million assertions which relate more than 100,000 concepts to each other. [4] Each such assertion is expressed in the representation language of Cyc, called CycL, a variant of first order predicate calculus with enhancements (see http://www.cyc.com/cycl.html for details).

## Metrics: Assertions versus Questions

There is a surprisingly pervasive tendency to evaluate rule based ontologies along dimensions such as the number of assertions or number of reified concepts. This is analogous to what has been known to happen in programmer communities. Certain programmers make boastful claims about how many thousands of lines of code they have written. In spite of this, all else being equal, the most efficient least buggy programs

tend to be the ones with fewer lines of code. The analogous is true for rule based ontologies. Like with computer programs, what really matters at the end of the day is not the number of assertions or the number of constants, but rather the overall functionality of the system.

How does one measure the performance of a rule based ontology? I submit that the number of questions that a given system is known to correctly answer, although incredibly crude, is a step in the right direction. Specifically, it is a step away from the trap of simply focusing exclusively on the number of assertions or the number of reified concepts.

Nonetheless, it almost goes without saying that a simple report of number of queries the system can answer begs certain questions such as how important are the queries and the answers? How correct are the answers? How robust are they to KB changes? To be sure, measuring this latter sort of functionality is no simple matter. However, the main point is that a focus on what inferences a rule based ontology can make is as least as important as quantifying the size of that system.

This paper discusses ontological issues associated with a question- or functionality-focused approach (as opposed to a content focused approach) to evaluating a rule based ontology.

## White Box versus Black Box Tests

An important distinction for evaluating intelligent systems can be imported from the software testing community - the distinction between black box and white box tests. Black box tests are those motivated out of required module functionality. Black box tests ignore system internals and specify requirements for the mapping of interface inputs to outputs. White box tests, on the other hand, are motivated as a function of the targeted software module's internals. The designer of a white box test analyzes the innards of a system, identifies system weaknesses based on such an analysis and poses tests to check for buggy behavior arising from such weaknesses.

## Evaluation as Psychological Assessment

The difficulty of defining intelligence was alluded to in White Paper to PerMIS01. These difficulties apply to the evaluation of rule based ontologies as described below. To illustrate these difficulties, contrast the task of evaluating a physical map of France with a knowledge base representation of France. In the evaluating physical maps, one has access to physically verifiable measurements such as the vector from the actual Paris to the actual Lyon. One compares this vector with the vector from the map's Paris to the map's Lyon. A map may be evaluated on the basis of measurements such as these. By contrast, there is no such operational definition that instructs one how to evaluate the representation of France in a given KB. Any representation of France is an abstraction. An alternative evaluation model must be found.

One such alternative model can be found in human psycho-educational assessment. Test theory for this class of evaluation must cope with the problem that psychological constructs, unlike physical properties, can not be directly measured. Unlike physical scales, psycho-educational scales lack well-defined units. For example, if an examinee answers all questions incorrectly on a given skills test, does that indicate he or she has "zero" amount of that skill? Does equal spacing between different units on a psychological scale represent equal differences in the quantity being measured? Due to these problems, psycho-educational test theorists simply accept certain fundamental limitations. As a result, there is no universally accepted means of measuring any psychological construct [3]. Psychological constructs can not be defined in operational terms only. Relationships must be demonstrated to other constructs or other observables [3]. The prospect of applying specific lessons learned in the context of psycho-educational evaluation to evaluation of intelligent systems is worthy of further discussion at this conference.

## Vagueness, Ambiguity and Evaluation

Knowledge representation systems often need to represent and reason with vague and ambiguous concepts. Vagueness and ambiguity are less familiar in traditional software verification circles. In order to verify or validate a software system, one needs as complete and precise a specification of system requirements as possible. The more stringent the verification desired, the more precise the requirement specification needs to be. Such completeness and precision is problematic if one's application domain is commonsense reasoning. As the following use cases indicate, some commonsense reasoning requirements involve inherent vagueness or ambiguity.

For example,

*Fred loves France. (*A)

seems to correspond to a single coherent commonsensical assertion. A system with commonsense reasoning capability should be able to derive the following from (A)

*Fred would probably enjoy eating French food. (*B)

*Fred would probably enjoy visiting France. (*C)

*Fred would probably enjoy chatting with locals in France. (*D)

Yet, precisely what concept is referred to by the English word "France" in (A)? For example, does "France" denote (1) a body of land bordering on Spain, the North Atlantic and the Mediterranean, (2) an amalgam of regional lifestyles, culinary practices, artistic traditions etc or (3) the people who are citizens of France?

In addition to ambiguity, common sense systems should be able to reason with vague information. For example, given a fact of everyday life such as,

*Men prefer not to be bald.*

one should be able to use such an assertion in an explanation as to why a given bald man went to the store to buy medicine for his baldness. Being able to generate such an explanation is a requirement in spite of the fact that baldness is inherently vague or hard to operationalize.

As the prior use cases indicate, intelligent performance depends on the ability to handle ambiguous or vague information. For this reason, I claim that for intelligent systems with common sense performance evaluation must be less formal and less precise than traditional software verification in at least some cases.

## Ambiguity and Component Versus System Level Tests

Although ambiguity does impose limits on the precision of performance evaluation, it is not a complete show-stopper. This can be seen by introducing an ontological distinction made in the software testing community. The distinction I refer to is that between component tests versus system tests. Component level tests are intended to verify that a particular system component is behaving as intended. A system level test is intended to verify that an entire ensemble of components are interacting as specified.

In spite of the ambiguity of the term "France", we require of our system that upon being given (A) it should be able to derive (B), (C), and (D) as needed. Thus, if we are in a dialogue with our system, we should be able to tell the system

> *Fred loves France.*

and then ask it

> *Question, is it probable that Fred likes French cuisine?*

an intelligent system should answer

> *Yes.*

without resorting to requesting that the user disambiguate precisely what he or she meant by "France". This is a specification of a desired system level behavior and it is unambiguous.

The ambiguity in the above example involving Fred and France lies in interpreting *Fred loves France*. As the existence of interpretations (1), (2) and (3) suggest, there may not be a single coherent knowledge representation for that assertion. Thus, although at a system level performance evaluation is not ambiguous, ambiguity may be present at the component level.

## The Cyc Testing System: An Ontology-Based System for Evaluating Cyc

The main purpose of the Cyc Testing system is to evaluate the performance of the Cyc system so as to provide useful feedback to Cyc's developers. It consists of two main subcomponents: a declarative KB component and a procedural software component. The KB component consists of assertions about desired system behavior. These assertions are in the KB, articulated in CycL, Cyc's formal knowledge representation language. This paper describes the state of the testing system up to early spring 2001.

### What are the Testing Requirements?

What kinds of testing requirements must the Cyc Testing System support? Fundamentally, any knowledge based system is intended to make correct inferences. Thus, we desire the testing system to allow us to evaluate the correctness of these inferences. This broad requirement is made more specific via certain distinctions and desiderta described immediately below.

One such distinction has to do with the purpose of a given evaluation. There is comparative evaluation in which one desires to compare the performance of two or more systems. There is also developmental evaluation in which one seeks feedback in order to guide the process of system under construction. The Cyc Testing system is mainly intended for developmental evaluation. Nonetheless, many of the ontological distinctions represented in the testing system's KB may be applied to comparative evaluation.

Another requirement that the Cyc Testing System should meet is that it should ensure that system functionality never moves backward. What worked yesterday should continue to work today. This requirement is met by regression tests, a kind of black box test, as described below.

In addition, we desire to verify that constraints related to the internals of the system be upheld. Although there may not be an end user requirement directly motivating a given constraint, such tests may be said to enforce good KB housekeeping. KB Integrity Tests, a type of white box test described below, are intended to meet this requirement.

An additional desideratum is the ability to express desired system behavior and measure the ability of the system to meet or fail to meet a given set of specified behaviors. Challenge Tests, described below, are intended to meet this requirement.

Also, we desire the system to provide us with a means of measuring how well knowledge synergizes. We wish to detect unintended inferences, either good or bad, that might emerge when thousands or tens of thousands of rules interact. Rumination-Based Tests, described below, are intended to meet this requirement.

Finally, we desire the testing system to keep developers, managers and evaluators abreast of how the system performing and how its functionality is changing on a day to day basis. This should include an ability to aid in the diagnosis or system problems or bugs. How the Cyc Testing System meets these requirements is described below.

### System Components and Operation

The ontology-based Cyc Testing System has two components. A software component and a knowledge base component. The software component consists of procedural statements which act in response to KB test assertions (which comprise the knowledge base component) in the appropriate manner. For example, depending on the assertions associated with a particular test object, the Cyc testing system may make calls to the inference engine or to one of several natural language parsing systems. It may compare current to prior results along any of several performance dimensions such as number nodes traversed during inferencing, time elapsed, or results obtained. More details of the kinds of properties one may associate with KB test objects are described under the next subheading.

The Cyc Testing System can be invoked by a system user in a wide variety of ways. It is also automatically invoked in nightly testing runs. Results of particular tests and summaries of test suites can be automatically emailed to interested parties.

### Test Ontology

I submit that it is a good idea to use an ontology to evaluate an intelligent system such as Cyc. It is beneficial to declaratively assert evaluation guidelines to the knowledge base (KB) in a manner very similar to the way one ontologizes "normal" common sense or expert knowledge. As the power of knowledge engineer tools increases, as ontology-based natural

language interfaces become more usable, these gains leverage one's ability to communicate with the testing system.

## Should Tests Correspond to Assertions or Reified Terms?

One ontological issue that is fundamental to the testing system is the decision to a let a given test be denoted by a single reified KB term. Thus, test properties are articulated as assertions about that term. This section provides details on how that is done and then justifies why things were ontologized as they were.

Suppose one wished to test whether or not Cyc knows the common sense fact that *adult humans are capable of running*. One way to test knowledge of this fact is to pose a query to the inference engine using the CycL language and verify that the answer came back correctly. In other words, we wish to declare that when one poses the query "Can humans run?" in CycL to the Cyc inference engine, then the inference engine should return True.

In order to do this one reifies a term representing the test and makes assertions about that term using vocabulary in the Cyc test ontology. Suppose we wish to name the term `#$Test007`. Take it as a given that the means of specifying the question *Can humans run?* in CycL is as follows:

```
(#$typeBehaviorCapable
   #$HumanAdult #$Running #$doneBy)
```

Thus, in essence one needs to tell the system that when the above CycL query is made, true should be returned. Such a statement is made with two cycl assertions as show below. The first assertion associates a cycl query with the test.

```
(#%testQuestionCycL
   #$Test01
   (#$typeBehaviorCapable #$HumanAdult
   #$Running #%doneBy))
```

The next assertion associates the desired answer, true, with that test as shown below.

```
(#$testAnswersCycL
   #$Test01
   (((T T))))
```

The above presupposes that there should exist a reified term for each test. One might ask, could not the above specification of desired behavior be achieved more economically with a single assertion? For example, by creating *testQuestionAnswer* a hypothetical new addition to the test ontology, we could arguably make a more economical representation. In the former scheme we incurred the expense of reifying #$Test01 plus an assertion about the query and its answer. With the hypothetical proposal *testQuestionAnswer* we get by with a single assertion as shown immediately below.

```
(testQuestionAnswer
   (#$typeBehaviorCapable #$HumanAdult
   #$Running #%doneBy)
   (((T T))))
```

In the short run, such a proposal would indeed be more economical. However, it decreases expressiveness of the test ontology. There are other properties that we wish to assign to a given test. Some of these properties include which Cycorp employee should be notified if the test should exhibit a system failure; how frequently the test should be run; what sort inference engine settings should be in effect; what contextual assumptions should be present and so forth. Thus, if there were N test properties, an N-ary predicate would be needed to describe a given test. As the arity of a predicate increases, it becomes more ungainly for both humans and inference engines.[1]

Aside from ungainliness, there are additional arguments against using a single assertion to describe a given test. For example, we wish to optionally state these properties for some tests and not others. CycL is not capable of representing a value of 'not yet determined' for a given argument position. Finally, as the testing system evolves we may wish to ascribe new test properties to certain tests. If at time 1 we might have 4 properties (and a corresponding quaternary test specification predicate) and at time 2 we have 5 properties, we would need to transfer all prior test specification assertions to from the quaternary to the quintary predicate.

The testing ontology for the kinds of tests described below allows one to associate numerous types of properties with a given test. For example, one may wish to describe which knowledge engineers are responsible for which tests, what kind of inference parameters should be involved with a given test (such as depth of backchaining, time to spend before forcing a return of bindings collected so far, the context or "microtheory" which should be visible to the inference engine when a given test is run) past performance of a given test, and more.

## Types of Tests

**Regression Tests** Adopted from traditional software testing, regression tests are a type of test used to detect backsliding. In other words, they ascertain that functionalities that worked yesterday continues to work today.

These tests can be considered a kind of black box test. Motivated from user requirements, they check for specified outputs being generated in response to specified inputs being applied to a target functional module. For example, one might have a set of inferences or natural language parses that one has worked on for various demos, contracts, etc. One uses a regression test if one wishes to make sure that such inferences or parses continue to work correctly (and within optionally specifiable resource bounds). Thus, unlike Challenge Tests (see below) a Regression Test is, by definition, known to have worked correctly at some time in the past. Thus, if a given Regression Test fails, it most likely means that a KB editing operation or a patch was made to the inference engine code causing that test to fail.

We typically desire to run such tests frequently so as to be quickly informed of any breakage and to minimize the candidate set of KB/inference engine changes which could have caused the breakage. A corrolary to this is that knowledge engineers are more able to diagnose a problem when the cause of that problem is known to lie in activities which are more recent and therefore fresher in their minds.

[1] Technically speaking, a single N-ary predicate may not be necessary. However, one will need several predicates whose arity is higher than two. Having several assertions all representing the same test that do not hang off the same test descriptor object is still ungainly.

As of March of 2001, the entire suite of approximately two hundred regression tests was run once per night. This seemed to be timely enough notification and also took advantage of the lull in computing needs that occurs at night when ontologists and programmers are out of the office. Each such run took the bulk of the night. The testing harness would boot up a fresh Cyc image. Next it would cause that fresh Cyc image to iterate through a transcript of KB edits. This transcript contained the sequence of several thousand edits to the knowledge base that had accumulated from the work of several score knowledge engineers. Once caught up on recent KB edits, the testing harness would iterate through the list of nightly tests.

**Challenge Tests** Roughly speaking, a challenge test is analogous to a problem assigned to a student. Challenge Tests are specific problems posed to the targeted system that test its ability to apply knowledge towards the resolution of that problem. Such tests can be in the form of sample tests or test questions. Sample tests, much like homework given to human students, allow a group of ontologists to modify the KB without many constraints. As human students doing homework are allowed access to books, relatively unconstrained time, and consultation with teachers, ontologists are allowed similar liberties. This work is done with an eye towards maximal generality so as to best prepare the system for answering test questions. Test questions, by contrast, are analogous to final exams given to humans. In some cases test questions are posed directly to a system. In other cases, ontologists are allowed to make KB changes in order to make the system answer a given test question, however access to the KB subject to time limitations.

In contrast with Regression Tests, a given Challenge Test is not necessarily expected to work successfully. Rather, the purpose of Challenge Tests are to characterize a milestone in Cyc's abilities or to focus and stimulate work on enhancing them. This kind of test is relevant to performance evaluators interested in questions such as the following. "Can Cyc perform temporal reasoning?" or "Does Cyc understand DNA transcription?" or "Is the Cyc recursive block parser able to correctly parse the phrase P?".

Likewise, performance measurers can use these tests to monitor the human costs associated with knowledge engineering. Suppose that a week's worth of a given knowledge engineer's activity was focused on implementing a general solution to the question battery Z (a "teach set"). How much did Cyc's performance on that battery improve? How well did that week's worth of knowledge engineering generalize to the "test set" battery Z' (of which the knowledge engineer was unaware)? Further, suppose that later a different knowledge engineer, was set to work on the challenge question battery W. How much of the early knowledge was able to be re-used in the later ontologization effort associated with W?

Performance evaluation issues need not always be framed in terms of specific instances of questions. Rather, a class questions can be identified. Thus, an evaluator might ask, what percentage of the instantiations of question type Y is Cyc able to correctly answer? The Cyc testing ontology distinguishes between questions instances and parameterized questions – whole classes of tests which vary along some specified set of parameters. See [1] for more on this topic.

All else being equal, a more flexible KB can be more quickly modified to meet a new set of challenges. Likewise, a more flexible knowledge base axiomatization should generalize to unseen "test set" cases. Lastly, more knowledge re-use should be attainable from a flexible knowledge base. For these reasons, performance data on challenge tests may be said to measure the flexibility and re-usability of a knowledge base.

**KB Integrity Tests** The third kind of test is termed a KB integrity test. The purpose of such tests are to evaluate certain structural properties of the knowledge base. Thus, the purpose of such tests is not so much to state desirable end user functionality but rather to enforce good "KB housekeeping". Because they check for in-felicities having to due with system internals rather than those having to do with input-output relationships these tests a kind of white box test. A KB with high integrity is easier to maintain and augment.

One type of KB integrity test checks the syntactic well-formedness of a given KB assertion. Although assertions can not be entered in the KB if they are syntactically well formed, inevitably some assertions become non-well formed over time. One way this change can occur when constraints that define a predicate become more restrictive. When this happens, an assertion using that predicate in the zeroth argument position that was previously well-formed could become non-well formed.

Another higher level KB integrity test checks for the appropriate relationship (i.e. the subset relationship) between argument types of predicates which are in a subsumption relationship. For example,

```
(#$implies
   (#$brothers ?X ?Y)
   (#$siblings ?X ?Y))
```

In English this could be described as

*If two people are brothers, then they are siblings.*

Thus the predicate `#$brothers` is subsumed by `#$siblings`. The argument type for brothers is `#$MalePerson`. The argument type for siblings is `#$Person`. The integrity constraint to be checked by this test is that the argument type of a subsumed predicate, e.g. `#$brothers` should be no more general than the argument type for the subsuming predicate, e.g. `#$siblings`. If `#$MalePerson` is asserted to be a specific kind of `#$Person`, then this integrity constraint is not violated.

**Rumination-Based Tests** The goal of rumination-based testing is to look for emergent intelligence in the knowledge base. As explained below the is achieved by having humans evaluate a representative random sample of the resource bound deductive closure of the knowledge base. This technique allows human knowledge engineers to measure how well disparate pieces of knowledge are synergizing. Discoveries can be welcome surprises about assertions comingling in unanticipated but desirable ways or undesirable interactions between incompatible assertions.

To be sure, the more such welcome surprises one finds, the better one's knowledge base is synergizing. The more undesirable interactions one finds, the worse the synergy is. Of course, some surprises might be more interesting than others. I have left as future work how to quantify the overall quality of a set of deductions.

The following anecdote of an exploratory foray into rumination-based testing methods shows how the technique works. The Cyc system was set to backchain on the following cycl query.

```
(#$feelsTowardsEvent ?agent ?emotion
?event-or-object ?level)
```

In English, the above CycL query could be phrased as follows.

*Does anyone feel any emotion to any degree about any event or any object.*

There were hundreds of conclusions, the vast majority of them were correct and even surprisingly interesting in some cases. For example, Cyc inferred that *Queen Elizabeth II feels loyalty towards England*, that various Cycorp employees loved their spouses and their children, and that *Abraham Lincoln felt fear at the time of his assassination*. In a microtheory devoted to the beliefs of Christianity the system even inferred that *God loves his son Jesus Christ.* Having worked at Cycorp for over ten years I had a sense of what projects people were working on and based on that knowledge found it highly unlikely that anyone had asserted rules for these specific deductions.

However, what was the following, a most amusing and probably incorrect deduction in light of scandals occurring towards the end of the Clinton presidency.

```
(#$feelsTowardsEvent
   #$HillaryClinton
   #$Dislike
   #$BillClinton
   #$None) (E)
```

An English rendering of the above assertion follows.

*Hillary Clinton feels zero dislike towards Bill Clinton.*

An ontologist should naturally wonder what were the grounds for such a seemingly erroneous conclusion and attempt to fix the KB appropriately. Delving into the problem further, I uncovered the following (translated into psuedo English) the givens as being responsible for the above deduction can be stated as follows:

- (G1) *Spouses love each other.*
- (G2) *Love and dislike are contrary emotions.*
- (G3) *If PERSON1 feels emotion E1 towards PERSON2 and E2 is contrary to E1, then the intensity level at which E2 is felt is zero.*
- (G4) *Bill Clinton is Hillary Clinton's spouse.*

Each one of the above givens may appear roughly correct standing on its own. However, when chained together they led to the conclusion (E) which I believe most people would consider a false deduction. These types of infelicitous combinations of rules are exactly the sort of thing that rumination based testing is intended to uncover.

Once a given infelicity is located ontologists can be tasked with its repair. There are several possible repairs for a deduction such as the above.

(1) Split apart the concept Love into at least two forms, one would denote a brief "in the moment" feeling inconsistent with dislike. The second would denote a long term affective disposition frequently felt by spouses which is not necessarily

(2) Add an exception to (G1) such that it reads (G1) Spouses love each other except in cases of infidelity. Then add a representation of the Monica Lewinsky affair to the Cyc KB.

In this way, a kind of introspection test or open-ended querying may be used to evaluate the overall synergy of a KB, locate problematic assertions or terms, and, thereby obviate particular improvements. A crude measure of this synergy or reliability could be simply by taking the ratio of deductions judged invalid vs. valid by some human judges based on their intuitions about what seems commonsensical.

Having introduced the general nature of rumination-based testing, a more complete rendering of this technique follows. First, one specifies a rather open ended inferencing task or series of tasks for the system to perform within (user specified) bounds of specified computational resources constraints (e.g. time limit, maximum deduction chain length, contexts in which to perform the deduction etc). Exactly what sorts of tasks may be performed are described below. Because rumination-based tests may take several hours on desktop PC machines circa 2001 they are best done at night.

Secondly, once such a task is completed, knowledge engineers should perform a quality control analysis on the resulting deductions. Human evaluators look at the results of such a procedure and rate deductions along quality dimensions such as "Was the deduction common-sensical?" or "Did the deduction seem an interesting demonstration of Cyc's capabilities or was it merely a banal obvious fact?" In certain cases there are so many deductions made, that the best one can do is randomly sample the set of deductions and make statistical generalizations about the entire set. Thus, a disadvantage of this technique is that it can get quite labor intensive. Future work may identify heuristics and user interfaces which reduce human labor by filtering out banal deductions.

There are several techniques for generating a sampling of the resource bounded deductive closure of a given aspect of the KB. The simplest method was described above - that of backchaining on a predicate of interest. Several other methods described below remain to be tried as future work.

For example, one such relatively simple method involves forward chaining all rules in a given context or set of assertions. A third method involves simply backchaining on the consequent of every rule in the given assertion set. A fourth method involves repeatedly picking one rule at random, instantiating its antecedent, picking another random rule and backchaining on its consequent. After hundreds or thousands of iterations, such a technique (as of yet untried) is likely to yield numerous deductions.

A fifth method, called "Open Ended Asks", involves repeatedly picking an arbitrary n-ary predicate, P(X1,X2, ...Xn) and asking the inference engine for all bindings of X1, X2, ...Xn . On a run conducted in 2000, this latter technique, i.e. open ended asks, were performed on 634 arbitrarily chosen predicates. 6229 deductions were made over an 8 and a half hour test run. My informal quality control analysis suggested that the frequency of non-commonsensical to commonsensical deductions was between 1 in 10 and 1 in 1000. In addition, the

results were spot checked for interesting non-commonsensical deductions which could be easily described in this article. One such deduction which caught my eye showed up in natural language browsing mode exactly as follows: "Vienna is wet." Rendered in CycL viewing mode the assertion appears as follows:

```
(#$wetnessOfObject
#$CityOfViennaAustria #$Wet)
```

In order to maximize fidelity to the justification as it appeared when the above problem was found I have copied it directly from the interface below.

```
Argument :  Deduction #395917
```

- ```
  (#$implies
  (#$and
  (#$touches ?U ?X)
  (#$isa ?U ?TYPE)
  (#$genls ?TYPE
  #$LiquidTangibleThing))
  (#$wetnessOfObject ?X Wet)) in
  #$BaseKB
  ```
- ```
  (#$touches #$DanubeRiver
  #$CityOfViennaAustria)
  in #$WorldGeographyDualistMt
  ```
- ```
  :ISA (#$isa #$DanubeRiver
  #$LiquidTangibleThing)
  in #$InferencePSC
  ```
- ```
  :GENLS (#$genls
  #$LiquidTangibleThing
  #$LiquidTangibleThing)
  in #$InferencePSC
  ```

The above can be rendered in English as follows:
Vienna is wet because:
(1) *The Danube River touches the City of Vienna.*
(2) *The Danube River is a Liquid.*
(3) *If a liquid touches an object, then that object is wet.*

After identifying this infelicitous combination of assertions, it was decided to create a new attribute called `#$PartiallyWet` whose meaning was that at least some part of the object was wet but not necessarily the entire object. (3) was then re-written as:

(3) *If a liquid touches an object, then that object is at least partially wet.*

Of course, the above exploratory analysis of rumination base testing lacks metrical rigor. More experimentation and human effort are required in order to quantify the frequency and quality of felicitous and infelicitous synergies like the types described above. In spite of this, I believe that because these type of tests aim at uncovering the unexpected it is an important means and relatively untried means of evaluating the performance of an autonomous system.

### Using the Ontology to Give Evaluative Feedback

The Cyc Testing System does not only use the ontology to specify tests and how they should be run. It also leverages ontology in the service of giving feedback about system performance.

For example, certain ontologists are responsible for certain tests. Responsibility information is encoded in the knowledge base as exemplified below.

```
(#$testCyclistsResponsible #$Test01
#$Fred-TheOntologist)
```

The above assertion signifies that Fred, a hypothetical ontologist working for Cycorp, is responsible for #$Test01, a hypothetical test object. Because the Cyc KB is a general purpose knowledge base and since email addresses are a salient important fact about humans today, Cyc has means of specifying the email address(es) of a person as exemplified below.

```
(#$eMailAddressText
#$BillJ-TheOntologist
"billj@cyc.com")
```

Such information can be exploited by the Cyc Testing system. When a regression tests detects a problem, the Cyc Testing System uses the KB to determine the ontologists(s) responsible for that test and uses #$eMailAddressText, if specified, to sends an email reporting the breakage. This is a modest example of how general ontologized knowledge can be leveraged in the context of performance evaluation.

### Localizing the Cause of a KB Bug

Suppose a given test, such as an integrity test or regression test, run on a given night starts detecting a failure that was not detected during the prior evening's run. Such a scenario was quite common, an informal analysis suggested an average of at least one new problem was detected per day during the last several months of nightly runs. Most likely, the bug was introduced as a result of the knowledge editing activity of the intervening day. A facility called the Breakage Pinpointer can help diagnose the source of the problem. In particular, this facility determines which KB edit operation was the one after which the given test began to signal a new failure point. The breakage pinpointer does this simply by looping through the following procedure:

(0) Start up a new Cyc image at the beginning of yesterday's sequence of KB edit operations.

(1) Given yesterday's sequence of KB edit operations, execute the next N of them in order.

(2) run the test (3) if the test works, do (1) again else return the information that the breakage occurred between the operation just run and the Nth -1, inclusive. By setting N to 1 (for brief tests) or using larger N (for time consuming ones) coupled with a binary search technique one can automatically narrow down on the exact operation which caused the breakage.

## The Testing System and Technology Readiness

There are different means by which the Cyc Testing System can be used to measure and ensure technology readiness.

One means by which technology readiness can be measured is by way of regression tests. If a given set of supported functionality can be identified and cast in terms of a regression test suite, then any version of the Cyc KB that passes these tests can be considered certified according to the functionality articulated by the given test suite.

Integrity tests provide another means by which technology readiness can be quantified. All things being equal, a version of the Cyc KB which has fewer violations of a given integrity test exhibits a higher level of readiness. In addition, the rate of change in the number of violations of a given integrity test can help answer questions such as, is the organization of ontologists keeping up with KB defects as they occur? This information can inform best practices. In management terms, the ratio of human resources required to maintain integrity of existing knowledge versus that expended to add new knowledge can, over time, be monitored. This ratio can provide feedback to an organization in identifying best practices to ensure a given level of technology readiness for a given cost.

Challenge tests can help one quantify the re-usability of knowledge or the amount of effort required to meet novel requirements. In addition, the progress of a team of ontologists can be periodically tracked by means of challenge question progress tracking vocabulary created for this purpose. For example, one of the first stages in adapting the Cyc KB to a new set of challenge problems is to create the vocabulary needed for representing the question. A second step is sketching out an inference path in English. A third step is identifying any already existing KB rules that can be used as part of the inference path. Ontologists can indicate progress against these different milestones for a given challenge problem by means of challenge question progress tracking vocabulary. In some cases this progress can be automatically inferred (e.g. if a #$testQuestionCycL is asserted, it then follows that question vocabulary has been formulated).

Finally, rumination tests can provide measures of knowledge synergy and autonomous or emergent behavior. By providing frequencies of felicitous and infelicitous deductions in the resource bounded deductive closure, they allow one to estimate how well the KB may perform in novel circumstances.

### Future Work

Note a given instantiation of Cyc tests itself. This could be problematic. If a given instantiation contained a KB defect which affected the test specification of the KB, then the results of the testing system could compromised. Thus, in the future, it is important to have a stable testing well-verified testing system that stands apart from the system undergoing development. Nonetheless, in practice, this limitation was not noticed to have caused any problems while I was putting the system to use.

## Acknowledgements

Thanks to Jim Zaiss for many helpful suggestions. Also, although he has not reviewed this paper, I am indebted to Ken Murray for many useful ideas.

## References

[1] Cohen, Chaudhri, Pease and Schrag (1999), Does Prior Knowledge Facilitate the Development of Knowledge Based Systems, proceedings of AAAI-99.

[2] Cohen, Schrag, Jones, Pease, Lin, Starr, Gunning, and Burke (1998), The DARPA High Performance Knowledge Bases Project, AI Magazine, Vol. 19 No.4, Winter.

[3] Crocker, L. and Algina, J. (1986) *Introduction to Classical and Modern Test Theory* Harcourt Brace Jovanovich College Publishers, New York.

[4] Jarrold, W. (2001). Validation of Intelligence in Large Rule-Based Systems with Common Sense. *Model-Based Validation of Intelligence: Papers from the 2001 AAAI Symposium* (AAAI Technical Report SS-01-04).