Validation of Intelligence in Large Rule-Based Systems with Common Sense.

William Jarrold
Cycorp
3721 Executive Center Drive
Austin Tx 78759
Counseling Psychology Area
Department of Educational Psychology
University of Texas at Austin
billj@cyc.com

Abstract

The following describes work in progress on a testing system for a large common sense knowledge base. How this testing system is relevant to the requirements of validation of intelligence is described. Four kinds of validation tests are explained: challenge tests, regression tests, knowledge base 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.

Part I: Commonsense, Large Rule-Based Systems and Validation

This work starts with the postulate that for at least some application domains, an approach to validation that is less rigorous than traditional formal software verification techniques is appropriate. Cyc, I shall argue, is in such a domain.

Cyc is a large rule-based system being constructed at Cycorp (www.cyc.com). At current count Cyc contains over 1.2 million assertions which relate more than 100,000 concepts to each other. 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).

Intelligence Validation for Commonsense Systems

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, 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)

(b) Fred would probably enjoy eating French food .
(c) *Fred would probably enjoy visiting France* ,
(d) *Fred would probably enjoy chatting with locals in France.*

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 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) without asking a user of the system for more clarification about precisely what the term "France" means. On the other hand there are other circumstances in which desired inferences depend on automatically picking one of the above three disambiguations.

In addition, 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.

The above two use cases serve to address a point from the present conference's call for participation, i.e. that "it is not

even clear that we know how to specify validation criteria." I claim that for intelligent systems with common sense, we must settle for a less formal, less precise specification of requirements. As the prior use cases indicate, ambiguity and vagueness are necessary parts of requirement specifications for a commonsense reasoners. If requirements must contain ambiguity and vagueness then a formal requirement specification is impossible, making formal verification impossible. Given these limitations, one could argue that "validation of intelligence" should be based on a statistical inference made based on system performance on a sample of behaviors. In brief, I suggest that "validation of intelligence" for commonsense systems should be more like giving a psychological test than doing a proof of correctness.

Common Sense and Mission Critical Systems

There are a variety of mission critical applications in which commonsense reasoning capabilities are important. For example, a geopolitical crisis management system should be able to understand and reproduce human commonsensical reasoning patterns. An evacuation management system should possess the emotional intelligence required to help people remain calm in high pressure situations. A natural language interface for an avionics application should understand the pilot's colloquialisms. In general, the performance of any mission critical system which must interface with humans is likely to benefit greatly from commonsense capabilities.

Validation Issues and Large Scale Projects

Specification difficulties aside, another set of concerns derives from the sheer massive scale of the effort involved. Cyc is a project involving person-centuries of effort (Guha, Lenat 1990). There are currently about 80 full-time employees working for Cycorp, at least half of which send KB edit operations (KB edit operations are atomic KB editing events such as adding an assertion, removing an assertion, creating a term, giving an assertion a timestamp) to the main Cyc KB as part of their daily work. Recently the mean rate of sending operations to the main KB was calculated at 1900 per day (including weekends) over a 5 month period. Given the sheer volume of the effort, multiple inconsistencies are bound creep in.

In addition, from an efficiency standpoint it makes sense to use relatively incomplete but cheap to implement methods to find the easiest most glaring bugs early in a project's lifetime. Assuming sufficient "progress happens", i.e. Cyc nears completion and the formal model checking/model based validation techniques can be tractably applied to undecidable Cyc-like systems, then it becomes appropriate to proceed with such "big hammer" software verification techniques.

To sum up, I claim that for commonsense systems, validation of intelligence should not be a binary either/or decision. The ambiguity and vagueness inherent in specification of commonsense reasoning requirements forces us to follow a validation procedure based on performance across a possibly randomly sampled set of use cases. Secondly, even if that claim is incorrect in some way and advanced formal validation techniques could somehow be tractably applied to a large Cyc-like system (e.g. based on some fruitful hybrid of model based reasoning and model checking) it seems to make the most sense to only apply such relatively expensive techniques towards the end of a project when system change rate is lower or when hard to detect bugs are being sought out.

Part II: Specifics of the Cyc Testing System and Techniques Validating the Intelligence of Cyc

This section describes a working system that can validate (or assess) Cyc's intelligence. For reasons described above, the system performs validation of Cyc's intelligence as a graded measure based on tests derived from use case type requirements.

System validity is established by measuring different aspects of Cyc's performance on a set of tests. The hope is that by beginning to measure this performance across many different dimensions with more and more tests, strides will be made in the direction of more comprehensively validated commonsense reasoners.

The Cyc Testing System has two main parts: a software part and a knowledge base (KB) part. The KB part contains a large number of test objects which make declarative assertions about system requirements. Each test object describes a desired system behavior in CycL. The software part operates on a given KB test object 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, 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

Typically tests are represented as a reified first class objects in the knowledge base (exception: rumination-based tests, described below, do not yet have any explicit knowledge base objects associated with them). 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, 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. One declaratively asserts this kind of information to the knowledge base (KB) in a manner very similar to the way one ontologizes "normal" common sense or expert knowledge. For example, one may use standard knowledge engineer interface tools, and eventually, free-form natural language.

Additional of the Cyc Test ontology shall be described under headings which outline four kinds of test which are important to validating Cyc.

Regression Tests

Regression tests derive their name from software engineering. 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 specified 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 inference engine code patch was done which caused 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.

The CycL name for one such test that has been in the system since October of last year is CST-Pratt6aRunMt. The assertion

```
(isa CST-Pratt6aRunMt  NightlyKBAskRegressionTest)
```

means that this test is one which is run once per night automatically in virtue of certain properties of NightlyKBAskRegressionTest omitted from this paper. Since this test, and 120 others like it is of the "nightly KB Ask" sort as opposed to, for example, the "nightly natural language parsing" sort.

it checks a particular call to the Cyc inference engine, rather than, e.g. a call to one of Cyc's natural language parsers.

An English representation of what CST-Pratt6aRunMt iis supposed to test can be gleaned from the following assertion:

```
(testQuestionEnglish CST-Pratt6aRunMt "Can human adults run?").
```

Together, the following two assertions

```
(testQuestionCycL CST-Pratt6aRunMt
        (typeBehaviorCapable HumanAdult Running doneBy))

(testAnswersCycL CST-Pratt6aRunMt (((T T))))
```

mean that when the inference engine is set to work on the query (typeBehaviorCapable HumanAdult Running doneBy) it is expected to return "true". (More precisely, we expect "true" as an answer to the CycL query only when a set of inference engine parameters are appropriately set. The parameters can either be explicitly asserted or inherited from other parts of the testing ontology. A detailed listing of what these parameters are would be beyond the scope of this paper.) Rending those two assertions in English, one could say that the English target answer to the question "Can human adults run?" is "yes."

Test objects which are of type NightlyKBAskRegressionTest should, by definition, be run automatically once per night. Such automatic testing has been happening since late Spring 2000. An added feature of the testing system is that the knowledge engineers can be automatically informed via email if any tests for which they are responsible have broken. The Cyc testing system can utilize KB information such as

```
(testCyclistsResponsible CST-Pratt6aRunMt RichardM)
```

and

```
(eMailAddressText RichardM "richardm@cyc.com")
```

in order to send email to the correct places.

If a given test stops behaving as desired, on a run of the regression tests, for example, a facility called the **Breakage Pinpointer** can determine which recent KB edit operation, if any, caused the test to stop working. 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.

In sum, use of regression tests can facilitate validation of intelligence by quantifying, giving timely notice of, and locating newly added system defects.

Challenge Tests

Roughly speaking, a challenge test is analogous to a "quiz" given to human students. In contrast with Regression Tests, an arbitrary Challenge Test is not necessarily expected to work successfully. Rather, the purpose of Challenge Tests are to characterize Cyc's abilities or to focus and stimulate work on enhancing Cyc's abilities. This kind of test is relevant to system validators would be interested in issues such as:

Is Cyc able to answer question X?

Is the Cyc recursive block parser able to correctly parse the phrase P?

What percentage of the instantiations of question type Y is Cyc able to correctly answer?

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 used in the later ontologization effort associated with W?

In Cyc the testing ontology distinguishes between questions nstances and parameterized questions -- whole classes of tests which vary along some specified set of parameters. See (Cohen, Chaudri, Pease, and Schrag 1999) 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 of a knowledge base. Since common sense systems should accommodate new knowledge and new problems efficiently, flexibility is arguably an aspect of the definition of validity.

KB Integrity Tests

The third kind of test is termed a kb integrity test. With such tests, one declaratively describes structural properties of the knowledge base which one wishes to monitor. The purpose of such tests is not so much to state desirable end-user functionality. Rather, such tests are akin to measures of good "KB housekeeping". A KB with high integrity is easier to maintain and augment.

One very fundamental KB integrity test checks the syntactic well-formedness of a given KB assertion. Another higher level KB integrity test looks for the appropriate 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.

The Quality Control Group at Cycorp is responsible for preventing too many integrity violations from cropping up. Email messages are automatically sent to the members of this group by the same nightly testing mechanism that the above mentioned nightly kb ask regression tests use.

Rumination-Based Tests

The goal of rumination-based testing is to allow discovery of particular deductions that Cyc can make that were otherwise unanticipated. Discoveries can be welcome surprises about assertions comingling in unanticipated but desireable ways or undesirable interactions between incompatible assertions. An overall quality measure of such

deductions suggests how well knowledge base content synergizes.

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).

("Does anyone feel any emotion to any degree about any event or any object.")

Results of the query were cached and the procedure was repeated several times. There were hundreds of conclusions, the vast majority of them were correct, even surprising 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.

But, what was most amusing and not very intelligent seeming was the following deduction.

(D) (feelsTowardsEvent HillaryClinton Dislike BillClinton None)
Hillary Clinton feels zero dislike towards Bill Clinton.

Translating these axioms into psuedo English the givens 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 it, then the 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 (D) which most people would rank as false.

There are several possible repairs for such a deduction:

(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 specificies 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 current standard desktop PC machines at Cycorp they are best done at night.

Secondly, once such a task is completed, knowledge engineers 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.

There are several techniques for generating a sampling the resource bounded deductive closure of a given aspect of the KB. The simplest method involves forward chaining all rules in a given context or set of assertions. Another method involves simply backchaining on the consequent of every rule in the given assertion set. A third method involves repeatedly picking one rule at random, instantiating its antecedent, picking another rule and backchaining on its consequent. A fourth 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 recent run, 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)
```

The deduction, shown exactly as it appears in the interface is shown 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
```

In colloquial english the above can be rendered 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.

More rigorous measurements using this technique are in order.

Conclusion

Although commonsense reasoning capability is arguably a desirable property of mission critical systems, difficulties associated with the specification of commonsense reasoning requirements suggest that the validation of the "intelligence" of such systems should be done in a way which differs from traditional software systems. Because of such limitations, rather than creating a precise, general and complete specification, the best one can hope for is that validation criteria be delineated by performance across a comprehensive set of salient use cases. Intelligence validity dimensions of regression, flexibility, internal consistency and synergy were sketched. With further deployment of the ontology based testing system and thorough analysis of results obtained, there is hope for validity measures that go beyond the impressionist formulations outlined above.

Acknowledgements

Thanks to Jim Zaiss for many helpful suggestions.

Bibiography

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

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

Guha, R. V. and D. B. Lenat. (1990) "Cyc: A Midterm Report." AI Magazine, Fall .