Adding Common Sense to Genesis

Robert McIntyre

Collaborators: Chris Calabrese, Dylan Holmes

May 12, 2011

Abstract

I worked together with Chris Calabrese, a fellow senior in Course 6, to integrate the wealth of commonsense knowledge from the OpenMind: Indoor Common Sense database into Genesis, a program which aims to read and understand stories the way that humans do. We were able to convert 10 separate tables into a form that can be consumed by Genesis, adding 26,055 commonsense rules. (Previously, Genesis had used 30-40 commonsense rules.) We tested Genesis under the stress of 5,000 rules and found that it could still process stories, and I also wrote a testing framework to confirm that the rules we added are indeed useful to Genesis.

The Common Sense Problem

Common Sense is an area of Artificial Intelligence that is still shrouded in mystery. How do we build systems that can practically reason about the world in almost any context? Expert systems of the past such as MYCIN could perform at human level or beyond over a very restricted domain of knowledge.(Buchanan and Shortliffe [1984]) But, the knowledge which "common sense" includes seems to go on forever, covering every social situation, knowledge of everyday items, basic physics, causes and effects, etc.. Yet, in order to build systems that think like we do, it is an absolute necessity that they have common sense. In this work, I greatly extend the number of rules that Genesis knows (by several thousand) by incorporating the wealth of knowledge contained in the Indoor Common Sense Database (`http://openmind.hri-us.com/`) . Genesis has a good shot at being able to use raw Common Sense knowledge, because it already has the ability to parse data into a meaningful variety of representations.

What is Genesis?

The Genesis system is an effort to understand and process stories the way humans do: by reasoning through analogy, finding high level plot elements, and extrapolating from rules learned in previous stories. Although every part of Genesis is rudimentary right now, it has shown promise; it can already find instances of revenge and other plot level events in simple stories. Genesis uses a set of rules expressed in normal English as a basis for its explanations of the events behind the stories it reads. For example, it might have a rule of the form "If xx harms yy, then yy may become angry at xx." After reading this rule, Genesis can find instances of anger in the stories it reads and offer up the explanation that that person was hurt and that that hurt was the source of their anger. Up until now, Genesis has used around 10-30 hard-coded rules relating

Algorithm 1 A sample story showing how Genesis can infer relationships based on previous rules it has seen, all in English.

```
Insert file Start experiment.
Start commonsense knowledge.
If xx is a toolshed then it may contain a spade.
Start story titled "Test".
The room is a toolshed. The room contains a spade.
The end.
```

to the particular story it is about to read.

Indoor Common Sense Database

The Indoor Common Sense website has been collecting simple commonsense rules and statements concerning all things related to indoor situations since August 2003.(Gupta and Kochenderfer [2004]) It focuses on on collecting commonsense information about "human desires, objects and their locations, and causality." A typical database entry might be something along the lines of: `{:vp1 are angry, :vp2 are mistreated}` which means that if a person is mistreated then they might become angry. The example in Figure 1 shows the output we would expect from Genesis if it read a rule generated from a `{:room toolshed, :obj spade}` entry in the locations table, which means that a room may contain a spade if it is a toolshed. The openmind website features an interactive prompt where users fill in blanks for various sentences which vaguely involve things that might happen indoors.

START parser

START is a parser written by Boris Katz that can both parse English sentences and also generate sentences given a programmatic description of the sentence to be generated (called *triples*). This piece of software makes it very easy to construct English sentences from the OpenMind tables for Genesis to consume.

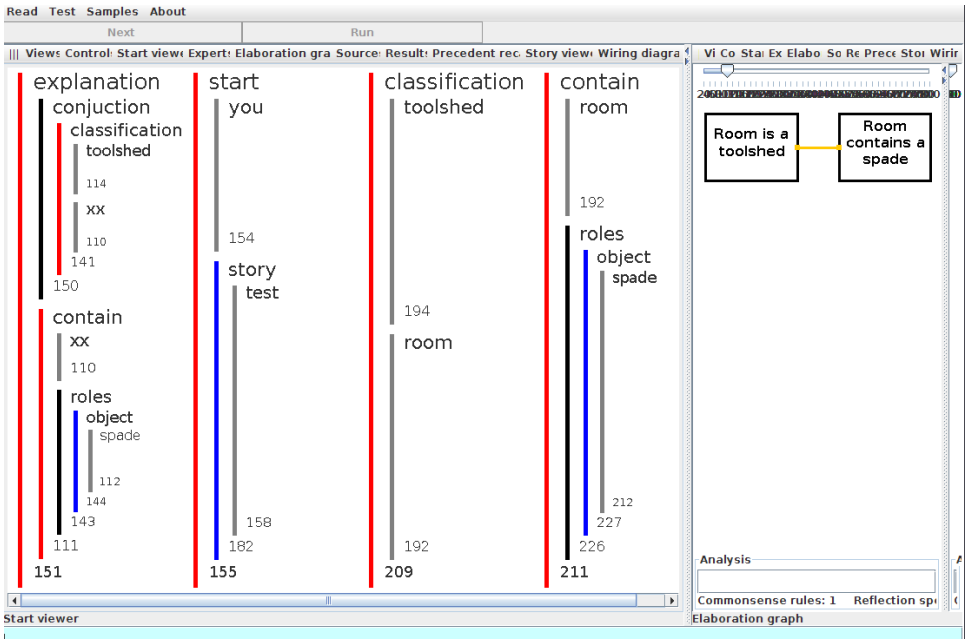

Figure 1: Genesis can construct representations of the stories it hears. In this figure, Genesis has read a rule that a toolshed may contain a spade and then after reading that there is a toolshed that happens to contain a spade, Genesis correctly forms an explanation relationship (seen on the far left) to show that the previous rule is has learned gives justification for the story. Seen to the right is an *elaboration graph,* which shows the basic plot units (events) which happen in the story, as well as the explanation inference represented by the yellow line.

Genesis itself uses the START parser to read its stories.

Steps

I take the SQL databases which comprise the totality of the Indoor Common Sense data and use START to write code which transforms each row in those tables into a proper English sentence that Genesis can understand. Thus, there are two parts to this exercise: generating proper English, and confirming that Genesis can draw the correct inferences from that English given the appropriate story.

Generating English

The system for generating English is divided into two parts: four classes that handle database connections and writing to files ,which is common to each table in the database, and 10 classes, one for each specific table in the database.

OpenMindTable.java

Each OpenMindTable object simply specifies where the data in the OpenMind database is located, and what to do with it once the data is obtained. The real meat of the entire package is contained in the getTriples() method, which does the work of actually taking a row of the particular table and transforming it into triples to be used by the START parser. Most of the databases follow a consistent template, and it is possible to simply parse one example sentence from the table and then just fill in the blanks appropriately. For example, the triple representation for

"John kissed Mary."

is:

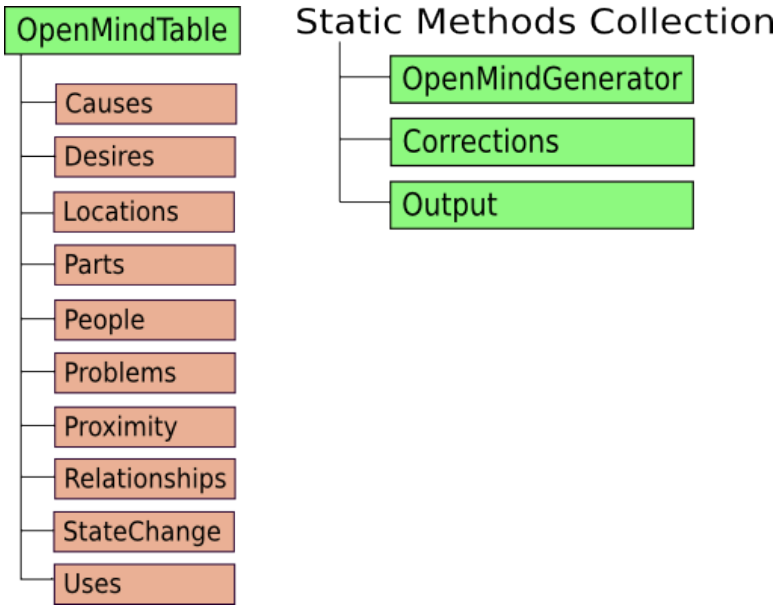

Figure 2: Class Structure of the OpenMind Genesis Package. It's just a set of 10 OpenMindTable classes which represent each table in the database, and some static methods to operate on said OpenMindTables.

Algorithm 2 The entirety of OpenMindTable.java. This class defines a contract which each of the ten tables follow to interact with the rest of the system. Each table must specify the name of the table in the OpenMind database which it represents, the keys which it expects to process, and a function to generate triples from those processed keys.

```
package openmind.generate;
import java.util.HashMap;
public abstract class OpenMindTable {
        public abstract String getTableName();
        public abstract String[] getKeys();
        public abstract String getTriples(
                HashMap<String,String> data);
        public boolean verifyMap (
                HashMap<String,String> data){
                // confirm that the hash map has every
                //key we expect it to contain.
                for (String key : getKeys()){
                        if (!(data.containsKey(key)))
                                {return false;}}
                return true;} }
```

Algorithm 3 Tables only contain enough information to find the appropriate data in the OpenMind database and to construct a triple from a single row of the database.

```
public class Locations extends OpenMindTable{
  public String getTableName() {return "locations";}
  public static final String OBJECT = "obj";
  public static final String ROOM   = "room";
  public String[] getKeys() {
    return new String[]{OBJECT, ROOM};}
  public String getTriples(HashMap<String,String> data) {
    String room =
      Corrections.baseCorrections(data.get(ROOM)) + "+1";
    String object =
      Corrections.baseCorrections(data.get(OBJECT)) + "+2";
    // START triples for generating
    // "XX may contain :object if it is :room"
    String triples =
      "[xx is-a+3 " + room + "]" +
      "[xx contain+1 " + object + "]" +
      "[contain+1 if+2 is-a+3]" +
      "[contain+1 has_modal may]" +
      "[" + room + " has_det indefinite]" +
      "[" + object + " has_det indefinite]" +
      "[is-a+3 has_tense present]";
      return triples;}
}
```

("[john kiss+1 mary]" "[john has_number singular]" "[john is_proper yes]" "[john has_det null]" "[mary has_number singular]" "[mary is_proper yes]" "[mary has_det null]" "[kiss+1 is_main yes]" "[kiss+1 has_person 3]" "[kiss+1 has_tense past]") .

To change the verb to another verb, say "loved", it suffices to simply replace the instances of ''kiss+1'' with ''love+1''. This is how nine out of the ten tables operate.

The people table, however is more complicated than the other tables, in that it has various verb forms that don't all fall into a single template pattern. The solution for this table was to construct a preliminary triples string by creating

a basic English sentence, then going into those triples and changing the first verb to be modal instead of absolute. This requires two trips to START to process each row, but correctly processes around 15 out of every 16 entries. The `help` table is more complicated still and appears to be too irregular to reliably convert into English.

OpenMindGenerator.java

OpenMindGenerator takes care of connecting to the SQL database, retrieving tables, and mapping the `getTriples`() methods from the OpenMindTable objects over every retrieved row. Of primary importance is the `getRules()` method, which returns a List of Strings which represents each of the English rules generated by a particular table.

Corrections.java

Corrections exists to get around some of the flaws in both START, and the OpenMindDatabase itself. There are two main problems that are common. The first is that START can not handle spaces in its triples, so multi-word nouns must have their spaces replaced with underscores. The second is that in order to avoid SQL injection attacks, the OpenMind database simply replaces all apostrophes with spaces. The static methods in Corrections correct both of these common problems.

Output.java

The Output class deals with mobilizing all the other methods to generate a text file for each table that contains all the English rules for that table.

News

Of the 11 initial tables identified as being usefull to Genesis, Chris Calabrese and I were able to transform 10 into a usable state for Genesis. The total number of rules sums up to 26,055 rules, of which more than half are solid, usable rules that Genesis can directly use. This is several orders of magnitude that the ≈30 rules that Genesis currently processes with each story.

To stress test Genesis, I took the locations.txt file which contained 5635 separate rules and appended the entire text to the beginning of the Macbeth story which is the common test story used in Genesis. Whereas Genesis normally takes around 1 minute to process Macbeth, the (Macbeth + locations) story took 45 minutes to process, with much of that time being consumed in network transactions and wordnet lookups. Genesis did complete the story and achieve the correct inferences after that time. It also generated additional inferences from the locations common sense knowledge. It is impressive that Genesis can handle so many more rules than it usually does.

Testing

Because it's important that Genesis can actually use the English rules to generate inferences in the stories it reads, I devised a testing framework for determining whether Genesis is actually interpreting the rules the way we want it to. The testing system is written in Clojure and is around 168 lines of code and works by creating a local instance of Genesis and then dynamically adding a node to the Inferences propagator, which reports inferences that Genesis derives in the form of Sequences of Relations. For each rule, my code generates a minimal test story and then runs it through Genesis' gauntlet of propagators. The results from the Inference node are scanned for an appropriate explanation relation, which shows that Genesis has correctly interpreted the rule and drawn

the right inference. For example, for the rule "The room may contain a spade if it is a toolshed.", Genesis will generate the test story found in Algorithm 1, and then check the inferences derived from this test story to see if they contain the explanations shown in Figure 1.

Future Work

- Move the SQL server serving the OpenMind tables from my home computer to a more reliable server in CSAIL.
- Pre-parse the 26,000 rules with START and store the triples in their own tables in the database to speed up processing.
- Extend the testing framework to cover the other 9 tables and use its results to prune away bad rules.
- Manually go through each table and remove obviously wrong rules as a sanity check.

Contributions

- Identified and classified 48 tables from the Indoor Common Sense Open Mind database.
- From these 48 tables, determined that 11 tables would be helpful to Genesis and completed an in-depth summary of their contents.
- Wrote code to convert 10 of the 11 tables identified into useful English rules for Genesis to use.
- Wrote a testing framework in Clojure to facilitate confirmation of the inferences generated by our new rules. (right now, the testing frame work

only covers rules from the locations table, but it is modular in the same way as the table transformation code, so I expect to add the other tables soon.)

- Analyzed the impact on Genesis when using 5,000+ rules.

Appendix

Summary of the tables in the OpenMind Indoor Common-Sense Database

There are 48 tables in the OpenMind Indoor Common Sense database; of these, 11 seem to have information that could be used by Genesis. They are:

- causes
  - 73459 entries
  - Data in the form (:obj1) has (:prop1) causes (:obj2) to have (:prop2).
  - {:prop1 interesting, :prop2 well written, :obj2 book, :obj1 book} → "The book is interesting if the book is well written."
- desires
  - 20258 entries
  - As a result of object (:obj1) having property (:prop1), one wants other object (:obj2) to have other property (:prop2).
  - {:obj1 trash can, :prop1 usable, :obj2 trash can, :prop2 empty} → "The trash can should be usable if the trash can is empty."
- locations
  - 5781 entries

  - Places (:room) and the things (:obj) you expect to find in those places.
  - {:room bathroom, :obj hot water} → "You expect to find hot water in a bathroom."

- parts
  - 27118 entries
  - A whole object, and part of the object.
  - {:obj computer, :part harddrive} → "A harddrive is part of a computer."

- people
  - 6204 entries
  - Describes the mental states of a person that an observer might infer given the actions of the person., i.e. if a person whines then they might want their way.
  - {:vp1 open a bottle, :vp2 want a drink of water} → "If a person opens a bottle then they might want a drink of water."

- problems
  - 23738 entries
  - Tasks to perform coupled with problems which might prevent one from accomplishing said tasks *alone*. For example, if you want to push something, you might be unable to do so because the object is too heavy.

  - {:problem the table is dirty, :task set the dining table} → ("If you want to set the dining table, you may be unable to do so if the table is dirty" OR "You can't set the dining table if the table is dirty.")
  - Out of 100 randomly selected entries, 5 were spurious and around 10 were duplicates. See also: help.

- help
  - 22861 entries
  - Like the opposite of :problems, but social. Describes a state that person1 is in and an action that person2 can take to help person1.
  - {:vp1 want to sleep, :vp2 keep the room quiet} → "If someone wants to sleep then you can help them by keeping the room quiet."

- proximity
  - 5593 entries
  - One object (:obj1) evokes mental images of another object (:obj2) due to "conceptual proximity."
  - {:obj1 tea cup :obj2 saucer}→ "XX may think about a tea cup if it sees a saucer."

- relationships
  - 9326 entries
  - One object (:obj1) could reasonably be found in this prepositional relation (:relationship) to another object (:obj2).
  - {:obj1 raincoat :obj2 closet :relationship in} → "XX may find a raincoat in a closet."

- statechange
  - 155547 entries
  - If one applies (:action) to an (:object1) having (:state1), it will change state into an (:object2) having (:state2).
  - {:object1 house plant :object2 house plant :state1 dry :state2 wet :action waters} → "The house plant becomes wet if the house plant is dry and XX waters it."
- uses
  - 10878 entries
  - The object/person/role(:obj) is generally used (to do)/(in the process of doing) activity (:vp).
  - {:obj chips :vp to have a snack} → "XX may use chips if it wants to have a snack. "

References

Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

Rakesh Gupta and Mykel J. Kochenderfer. Common Sense Data Acquisition for Indoor Mobile Robots. In *Nineteenth National Conference on Artificial Intelligence*, 2004.