Brainhat Natural Language Processing

Kevin Dowd
Atlantic Computing Technology Corp.
Hartford, Connecticut, USA
Email: dowd@atlantic.com

***Abstract*—This paper describes Brainhat, a natural language processing platform.**

I. Introduction

BRAINHAT work began in 1996. I was testing my belief that it is possible to build a complete NLP environment. I retooled a FORTRAN profiler for use as a parser. I added disambiguation, output generation and the ability to answer questions and run inferences. In 1998, the code spoke. 'Hello!,' it said, too loudly for the quiet dark of early morning.

I was lucky to have a a little bit of money. I purchased tables at trade shows to demonstrate my talking machine. Brainhat would chat about our visitors, making observations, asking questions and commenting on their clothing. "That would be great for customer relationship management...," visitors would say, "did you ever consider having it do phone sex?"

Eventually, we did; we experimented with a variety of applications. We added the ability to save and recall conversations, a database and the ability to dynamically shift domains of focus as a conversation proceeded. We wrote interfaces for email, instant messaging, VoiceXML, speech engines (including the ability for Brainhat to feed vocabulary and grammar hints back to the engines), robotics, HTML, text, a set top box and other copies of Brainhat.

The tech economy soured. By 2004 we needed to find jobs. The project sat in pieces for a few years, save for a frenetic stab at an occasional university project or wild notion. Dedicated work began anew in 2009 with the objective to provide a scalable development and runtime environment for work based upon Brainhat, to incorporate new features and to test the belief that it must be possible to build a complete NLP environment.

II. Brainhat by Example

```
>> the ball is red
 the ball is red.
>> what color is the ball?
 The ball is red.
```

Brainhat can parse and interpret simple English. The words *ball* and *red* are in its vocabulary: a *ball* is a *toy*, which in turn is a *thing* (a noun). The word *red* is an attribute (adjective). Brainhat can attach adjectives to nouns. By saying that the ball is red, we cause Brainhat to combine these two simple *concept*s to create a *complex concept*: a red ball. Brainhat creates a copy of the ball concept and appends the attribute concept, red, like so:

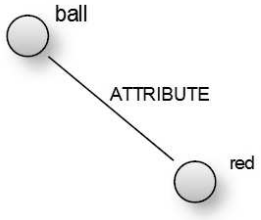

Fig. 1. a red ball

We can ask more questions:

```
>> is the ball blue?
 No. the ball is red.
>> what color is the ball?
 The ball is red.
>> do I say the ball is blue
 maybe. I do not know.
>> do I say the ball is red?
 Yes. You say the ball is red.
>> do I ask if the ball is blue?
 Yes. You do ask is the ball blue.
```

A. Inferences

The program can make inferences. *Inference templates* are derived from the dialog. The inference template below includes one condition: "I ask if the ball is blue", and one consequence: "I want a blue ball":

```
>> if I ask if a ball is blue then I want
    a blue ball
 if You do ask is a ball blue then You
  want a blue ball.
>> is the ball blue?
 Maybe. The ball. You want the blue ball.
>> why?
 You wants the blue ball because You do
  ask is the ball blue.
```

One can chain inferences; one consequence can cause others:

```
>> if I see the princess then I want
    the princess.
 If You see the princess then You want
  the princess.
>> if I want the princess then luigi
```

```
    is sad.
 If You want the princess then luigi
  is sad.
>> if luigi is sad then mario is happy.
 If he is sad then mario is glad.
>> if I want the princess then the
    princess wants me.
 If You want the princess then she
  wants You.
>> I see the princess
 You see the princess. You want the
  princess. The princess wants You.
  Luigi is sad. Mario is glad.
```

B. Taxonomy

Above, we described a taxonomic relationship when we said a *ball* is a *toy*, and a *toy* is a *thing*. Brainhat navigates taxonomies to identify specific references from more general references. Below, we refer to the *ball* as a *toy* and then as a *thing*.

```
>> I want the ball
 You want the ball.
>> do I want a toy?
 Yes. You want the ball.
>> if I want something then you are happy
 if You want something then I am glad.
  I am glad.
>> why are you happy?
 I am glad because You want the ball.
```

Here's another example showing generalization; the program believes it wants everything it sees:

```
>> if you see a thing then you want
    a thing
 if I see a something then I want
  a something.
>> you see the princess
 I see the princess. I want the princess.
>> you see a ball
 I see a ball. I want a ball.
>> you see water
 I see water. I want water.
```

C. Verb Tenses

Brainhat can differentiate verb tenses:

```
>> i saw the princess in the water
 You did see the princess in the water.
>> i see mario
 You see mario.
>> what do i see?
 You see mario.
>> what did i see?
 You did see the princess in the water.
```

Two verb forms—the imperative and the hypothetical subjunctive—have special meaning for Brainhat:

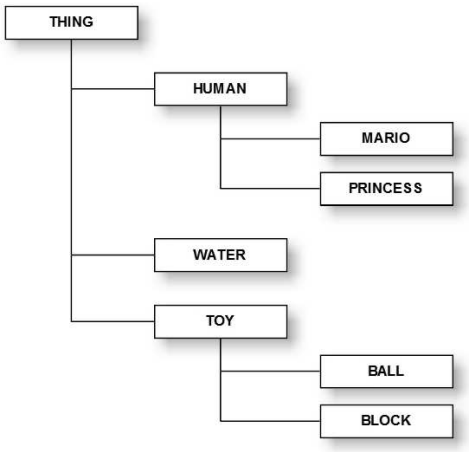

Fig. 2. taxonomy

```
>> tell me that you like me
 I do like You.
>> do you like me?
 maybe. I do not know.
```

The imperative is particularly important when the subject or object is associated with an external process or robot. One might first identify a robot—say, "Robby". Then one might instruct Brainhat to tell Robby to do something, e.g. "tell Robby to put the ball in the water."

The hypothetical subjunctive with modal verb "might" causes Brainhat to seek resolution of the hypothesis. If, for instance, one tells Brainhat "mario might want to see the princess", Brainhat will ask itself if this is true. If it doesn't know the answer, it may ask:

```
>> mario might want to see the princess.
 mario might want to see the princess.
  does mario want to see the princess?
>> yes
 mario wants to see the princess.
```

These subjunctive clauses, used in conjunction with inferencing, make it possible to progress a dialog in a goal-oriented fashion. For example, one could instruct Brainhat "if mario likes women then mario might want to see the princess". Once we learn that mario likes women, we have more to discuss.

D. Meme Shifting

Some versions of Brainhat code have the ability to segregate Brainhat English language programming into multiple domains or *memes*. Each meme can be a mix of propositions and inference templates like those we looked at in the examples above. Under the topological direction of a map (called a *meme map*), Brainhat can shift from one meme to another as dialog proceeds.

Inference templates and statements in the hypothetical subjunctive behave as honey pots in the meme-shifting algorithm; they make a particular meme more attractive when they become relative to the ongoing dialog. One might tell

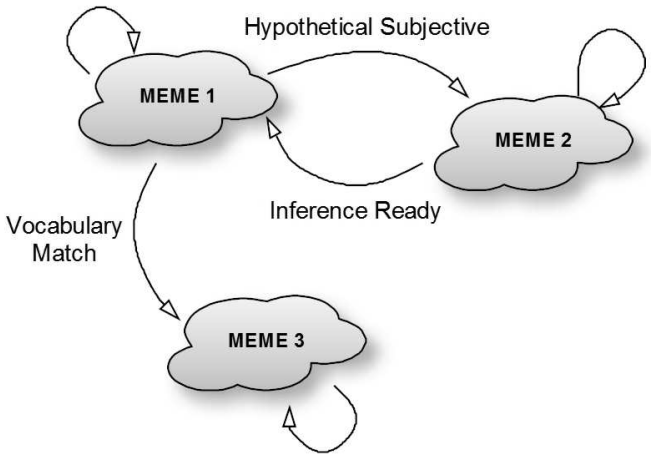

Fig. 3. meme shifting

Brainhat, for instance, “I might ask where is the princess.” If I subsequently ask where she is, the meme containing the statement will be a prime candidate for raised focus. Similarly, an inference template such as “if mario sees the princess then he might be happy” may precipitate a shift to the associated meme once brainhat learns that mario sees the princess.

III. Brainhat Runtime

We begin with a vocabulary. Brainhat’s vocabulary contains simple *concept*s like a ball, or the color red. These concepts are connected hierarchically to others—e.g. balls are toys, and red is a color. Links between the elements define taxonomy’s structure. Everything is the child of something else, and some are the child or parent of many. The following shows how taxonomic vocabulary is defined:

```
define   woman-1
         label          woman
         child-of       human-1
         person         first
         related        man-1

define   human-1
         label          human
         label          person
         child-of       mammal-1
         wants          mood-1

define   mammal-1
         label          mammal
         label          creature
         child-of       animal-1

define   animal-1
         label          animal
         child-of       things
```

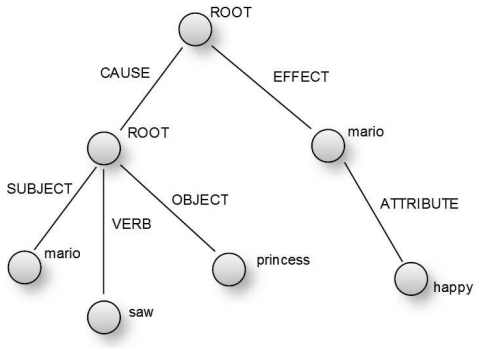

Fig. 4. mario is happy because he saw the princess

A. Complex Concepts

These simple concepts can be combined to form arbitrarily complex relationships. Within brainhat, these structures are called *complex concepts* (CC)—ideas made from other ideas. CCs can represent elementary assertions, e.g. “the ball is red.” They can be propositions, such as “mario sees the princess”, or inference templates, such as ”if the golden sun is shining then beautiful people are happy.” They can also be statements of cause-and-effect like ”mario is happy because he saw the princess.” CCs can even represent questions.

Complex concepts can be envisioned as inverted trees. The constituent concepts hang from their “roots”, like mobiles of ideas. The more abstract parts of the idea (e.g. cause-and-effect) live near the top. The actors and their attributes (golden sun, beautiful people) live near the bottom. The links between them define their relationships.

define	Root-8006	[999967245] (168811376)
	label	sent-action-10
	child-of	things
	auxtag	no-object-context
	hashval	377
	subject	mario-8007
	verb	tosee-8008
	object	princess-800c
	tense	past
	number	singular
	person	third

As processing proceeds, CCs (e.g. “mario saw the princess”) are assembled, destroyed, evaluated, compared and archived. Many live short lives as tendered (though sometimes incorrect) interpretations of something the user may have said. Others are the result of inferences. A few CCs survive to become part of the context of the conversation in progress, and to be added to the pool of things “known.” The detritus left by the process is garbage collected at the end of each input cycle.

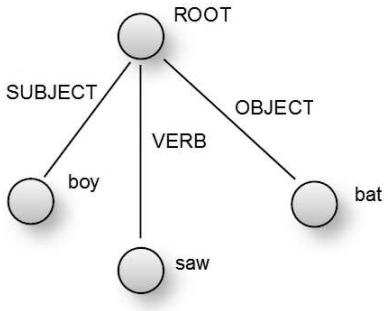

Fig. 5. boy saw bat

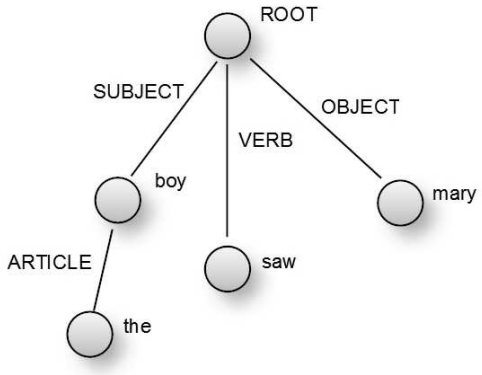

Fig. 6. the boy saw mary

B. Input Processing

Brainhat's ability to understand, learn, answer questions and infer are the product of creation and transformation of CCs. Parsing and pattern matching grammars tell brainhat how to cast fragments of input into CCs or how to recognize the knowledge content within a CC. Processing routines manipulate the CCs to change their meaning, or combine them to make new.

Brainhat attempts to match user input against a set of grammar patterns, one at a time, until it finds a fit. The "fit" is a parts-of-speech match; it does not presuppose the meaning of the matched text. Rather, many permutations may be generated, with many different meanings. "Boy saw bat," for instance, might generate CCs that represent "bat" as a winged mammal and as an wooden mallet. "Saw" could mean "viewed" or it could mean "cut in half."

A rule that matches "boy saw bat" might look like this:

```
define xxx
   label   sentence
   rule    $c‘things‘0! $c‘actions‘1!
            $c‘things‘2
   map     SUBJECT,VERB,OBJECT
```

Pattern elements corresponding to *boy*, *saw* and *bat* appear in the corresponding locations. The `$c‘`*parent*`‘n` construct says that brainhat should attempt to match a word that is a child of *parent*, and assign it to the nth position listed in the *map* directive. For instance, `$c‘`*things*`‘n` would match any taxonomic child of *things* and assign it to the 0th position in the map, making it the SUBJECT. Each potential interpretation will be threaded onto a concept chain. The number of permutations generated will depend on the number of vocabulary definitions for each word in the input, the number of dirty [1] copies of each word in circulation and the complexity of the input.

The map directive describes what the resulting CCs should look like. There will always be a root node. From that, components hang down, one level deep.

CCs are typically constructed from other CCs. Matching descends and rises, striving to build multi-level CCs from the bottom up. Expanding the previous example, we might want to match more complicated utterances such as "the boy saw the bat," or "the boy saw mary" using the patterns below:

```
define   xxx3
   label   sentence
   rule    $r‘subobj‘0! $c‘actions‘1!
            $r‘subobj‘2
   map     SUBJECT,VERB,OBJECT

define   zzz
   label   subobj
   rule    [$c‘article‘0! ]$c‘things‘1
   map     ATTRIBUTE,ROOT
```

Rules can invoke other rules. The `$r‘`*subobj*`‘x` construct instructs Brainhat to attempt sub-rules of the type *subobj* and assign matches to the SUBJECT position. By virtue of delegation the construction of individual components (subject, object, etc.) to other rules, we can construct multi-level CCs.

Upon making a successful match, Brainhat passes candidate CCs onto post-processing routines. These routines may change the shape of the CCs, eliminate a few, or use them for speech or to direct further processing. Each *mproc* is executed in turn, starting from the bottom and working upward in the list.

```
/* Where is x? */

define   sent-where
   label     question
   rule      where $c‘tobe‘0! $r‘csubobj‘1
   map       VERB,SUBJECT
   mproc     SPEAK
   mproc     CHOOSEONE
   mproc     PULLWHERES
   mproc     TOBECOMPACT
   mproc     PUSHTENSE
   mproc     REQUIREWHERES
```

The rule above is taken from Brainhat's distribution input grammar. It would match questions such as "where is the boy?" *mproc* routines `REQUIREWHERES`, `PUSHTENSE`, `TOBECOMPACT` and `PULLWHERES` change the shape of the

[1] a *dirty* concept is one that has been in the course of processing, possibly being linked to other concepts

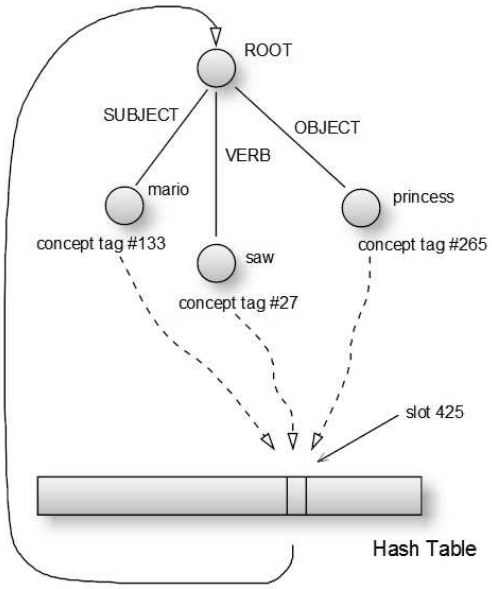

Fig. 7. simple additive hash

candidate CCs, dressing them up so that they become potential answers to the question. Routine CHOOSEONE eliminates all of the candidates but one. The result is passed onto SPEAK for output.

C. Ambiguity Resolution and Choice

Brainhat navigates through ambiguity in language by evaluating each CC against itself (vertically), to see whether it makes sense alone, and against a context buffer (horizontally), to see how it fairs against ideas that came before. Reduction in the number of permutations of potential interpretations, pronoun disambiguation and handling of anaphors proceed as input is shaped and processed. Brainhat further looks for *orthogonality* in attributes to differentiate between actors. For instance, Brainhat can detect a difference between a red ball and a blue ball based on the orthogonality of the attributes red and blue.

D. Hash Tables

Execution is in part serial—driven by input, and part associative—driven by the affinity of data for other data. To accommodate associative processing, Brainhat keeps hash tables. Hashing is useful when one is interested in matching concepts by their content without doing an expensive component-by-component pattern match. It is one of the mechanisms by which questions are answered, by which inferences are triggered and by which meme shifting is initiated.

The components of CCs that contribute to the hash are the parts of speech. For instance, in a simple proposition like "Mario sees the princess," we might choose the SUBJECT, OBJECT and VERB as components that we care about in a hash. Each vocabulary entry is assigned a tag number. The tag numbers for *mario*, *to-see* and *princess* will take part in the hash calculation.

In some cases, the hashes are built explicitly; the code does a pattern match against a CC, pulls out the most interesting parts and manually constructs the hash. In other cases, *auto-hashing* discovers the most interesting components. With the interesting concepts in hand, we create an integer from their tag numbers. This integer becomes the hash storage location, modulo the size of the hash table. A pointer to the CC that created the hash is stored along with a list of tags representing the concepts that make up the hash. This allows us to check if the hash actually corresponds to the components that caused a successful fetch, or whether the match was a hash collision.

IV. Conclusion

There is quite a bit more to Brainhat. There is also a lot to do. As with other dialog systems, conversation with Brainhat can be brittle; once the thread is lost or the system comes to misunderstand some input, it can be difficult to continue. Furthermore, creating a system that is conversant over a wide array of domains is a challenge.

We approached the domain challenge with meme-shifting. We are going to pursue another approach as well—parallel brainhat communities. This would be many copies of Brainhat communicating in an ad-hoc configuration, each greedily gathering the information that interests it and sharing their results with the rest of the community.

Among the most interesting late developments is the elimination of language from the saved context, in part to support meme-shifting and brainhat communities. We have the ability to compute with knowledge that the language left behind, without the language!

The laundry list of projects also includes:

- Look into a pattern matching front-end for processing idiomatic language into forms that can be processed more cheaply. AIML comes to mind.
- Hone the robot interface.
- Implement some kind of temporal tagging so that CCs can age out.
- Burden each CC with its lineage so that if CC that led to its existence is be found to be invalid or false, the descendant CC can be discarded as well.
- Collect and use statistical information for dialog.
- Improve memory management.
- Document more.
- Provide some kind of language subroutine capability for logical and mathematical computations.