The Elements of AIML Style

March 28, 2003

By

Dr. Richard S. W A L L A C E

The Elements of AIML Style is a no-nonsense technical book that takes you on a journey from the first steps of creating your own bot with AIML, through all the questions and answers every botmaster asks, to advanced A.I. and hard-nosed business applications of AIML. The trip ends with a brief glimpse into the future of bots and AIML.

INTRODUCTION

If you are looking for a revolution in IT, you've come to the right place.

There are several ways in which A. L. I. C. E. and AIML defy the dominant computer science paradigms.

For one thing, the ways in which humans and computers normally communicate are very different. Humans tend to spend a lot of time on chit chat and informal dialogue with little or no purpose. Computers are known for giving precise, true, and logical answers. The rate of information exchange in most human dialogue is very low, no more than 1Kbit per second, but computer communication is much faster. Alicebot/AIML is an attempt to bridge this divide.

We have dispensed with much of the conventional wisdom from structured programming, too. Where less code is usually good, more AIML is usually better. Filling up RAM with a lot of code that is seldom activated is a good idea in AIML. Allowing a novice programmer to write a large and unwieldy program in AIML is also, unconventionally, good practice.

Not to mention, the basic minimalist approach of AIML is borrowed from ELIZA, a design that was largely abandoned by the research establishment, indeed even derided as a "toy". Yet the success of A. L. I. C. E. and AIML if anything put the research establishment to shame: for their years of research and millions of spending, they have achieved nothing close to the performance of A. L. I. C. E. in natural language processing. Even the success stories of AI research, such as the chess-playing supercomputers, required far higher expenditures than A. L. I. C. E. and AIML.

The conventional model of an information service assumes that a computer will always provide accurate replies promptly. People have asked, what is the difference between Ask Jeeves and A. L. I. C. E.? Jeeves is designed so that the client will ideally ask one question, and then immediately receive an accurate reply including a hyperlink to another site. After just one transaction, the client clicks on the link and leaves the Ask.com site. A. L. I. C. E., on the other hand, is designed to keep the client talking as long as possible, without necessarily providing any useful information along the way. The longer average conversation lengths measured over the years have in fact been a measure of A. L. I. C. E.'s progress.

Strange as it seems, free software is still not considered a mainstream methodology for either research or software development. As big and successful as Linux is, the majority of the world has not yet bought the free software argument. For many people, the whole debate over software intellectual property remains obscure. Add artificial intelligence to the mix, and you have something truly unique. There

are a few other free software AI projects out there, but none has had the impact of Alicebot/AIML. This year was our first appearance at Linuxworld, a first step in finding common ground between the AI and free software communities.

Finally, many in the Alicebot/AIML community share a common vision of the future of our technology. Alicebot is perceived as a "missing piece" of the puzzle that combines speech recognition, natural language understanding, and voice synthesis. The vision is the talking Star Trek/HAL-style computer of the future, the invisible piece of hardware that responds to voice commands. In that vision, there is no keyboard, display, mouse, or graphical user interface. If that vision comes true, the impact of AIML will be ubiquitous

WHAT MAKES ALICE WORK?

Before we get to ALICE, we need to visit another unusual figure in the history of computer science: Professor George Kingsley Zipf. Although he was a contemporary of Turing, there is no evidence the two ever met. Zipf died young too, at the age of 48, in 1950, only four years before Turing, but of natural causes.

There are many ways to state Zipf's Law but the simplest is procedural: Take all the words in a body of text, for example today's issue of the New York Times, and count the number of times each word appears. If the resulting histogram is sorted by rank, with the most frequently appearing word first, and so on ("a", "the", "for", "by", "and"...), then the shape of the curve is "Zipf curve" for that text. If the Zipf curve is plotted on a log-log scale, it appears as a straight line with a slope of -1.

The Zipf curve is a characteristic of human languages, and many other natural and human phenomena as well. Zipf noticed that the populations of cities followed a similar distribution. There are a few very large cities, a larger number of medium-sized ones, and a large number of small cities. If the cities, or the words of natural language, were randomly distributed, then the Zipf curve would be a flat horizontal line.

The Zipf curve was even known in the 19th century. The economist Pareto also noticed the log-rank property in studies of corporate wealth. One only needs to consider the distribution of wealth among present-day computer companies to see the pattern. There is only one giant, Microsoft, followed by a number of large and medium-sized firms, and then a large tail of small and very small firms.

Zipf was independently wealthy. This is how he could afford to hire a room full of human "computers" to count words in newspapers and periodicals. Each "computer" would arrive at work and begin tallying the words and phrases directed by Zipf. These human computers found that Zipf's Law applies not only to words but also to phrases and whole sentences of language.

8024 YES

5184 NO
2268 OK
2006 WHY
1145 BYE
1101 HOW OLD ARE YOU
946 HI
934 HOW ARE YOU
846 WHAT
840 HELLO
663 GOOD
645 WHY NOT
584 OH
553 REALLY
544 YOU
531 WHAT IS YOUR NAME
525 COOL
516 I DO NOT KNOW
488 FUCK YOU
486 THANK YOU
416 SO
414 ME TOO
403 LOL
403 THANKS
381 NICE TO MEET YOU TOO
375 SORRY
374 ALICE
368 HI ALICE
366 OKAY
353 WELL
352 WHAT IS MY NAME
349 WHERE DO YOU LIVE
340 NOTHING
309 I KNOW
303 WHO ARE YOU
300 NOPE
297 SHUT UP
296 I LOVE YOU
288 SURE
286 HELLO ALICE
277 HOW
262 WHAT DO YOU MEAN
261 MAN
251 WOW
239 SMILE
233 ME
227 WHAT DO YOU LOOK LIKE
224 I SEE
223 HA
218 HOW ARE YOU TODAY
217 GOODBYE
214 NO YOU DO NOT
203 DO YOU
201 WHERE ARE YOU
.
.
.

The human input histogram, ranking the number of times ALICE receives each input phrase over a period of time, shows that human language is not random. The most common inputs are "YES" and "NO". The most common multiple-word input is "HOW OLD ARE YOU". This type of analysis which cost Dr. Zipf many hours of labor is now accomplished in a few milliseconds of computer time.

Considering the vast size of the set of things people could possibly say, that are grammatically correct or semantically meaningful, the number of things people actually do say is surprisingly small. Steven Pinker, in his book How the Mind Works wrote that

"Say you have ten choices for the first word to begin a sentence, ten choices for the second word (yielding 100 two-word beginnings), ten choices for the third word (yielding a thousand three-word beginnings), and so in. (Ten is in fact the approximate geometric mean of the number of word choices available at each point in assembling a grammatical and sensible sentence). A little arithmetic shows that the number of sentences of 20 words or less (not an unusual length) is about 10^{20}."

Fortunately for chat robot programmers, Pinker's combinatorics are way off. Our experiments with ALICE indicate that the number of choices for the "first word" is more than ten, but it is only about two thousand. Specifically, 1800 words covers 95% of all the first words input to ALICE. The number of choices for the second word is only about two. To be sure, there are some first words ("I" and "You" for example) that have many possible second words, but the overall average is just under two words. The average branching factor decreases with each successive word.

531 WHAT IS YOUR NAME
352 WHAT IS MY NAME
171 WHAT IS UP
137 WHAT IS YOUR FAVORITE COLOR
126 WHAT IS THE MEANING OF LIFE
122 WHAT IS THAT
102 WHAT IS YOUR FAVORITE MOVIE
92 WHAT IS IT
75 WHAT IS A BOTMASTER
70 WHAT IS YOUR IQ
59 WHAT IS REDUCTIONISM
53 WHAT IS YOUR FAVORITE FOOD
46 WHAT IS AIML
38 WHAT IS YOUR FAVORITE BOOK
37 WHAT IS THE TIME
37 WHAT IS YOUR JOB
34 WHAT IS YOUR FAVORITE SONG
34 WHAT IS YOUR SIGN
33 WHAT IS SEX
32 WHAT IS YOUR REAL NAME
30 WHAT IS NEW
30 WHAT IS YOUR AGE
30 WHAT IS YOUR GENDER

28 WHAT IS YOUR LAST NAME
27 WHAT IS HIS NAME
27 WHAT IS YOUR SEX
26 WHAT IS 2+2
26 WHAT IS MY IP
25 WHAT IS YOURS
24 WHAT IS YOUR PURPOSE
21 WHAT IS YOUR FAVORITE ANIMAL
20 WHAT IS 1+1
20 WHAT IS YOUR HOBBY
19 WHAT IS THE WEATHER LIKE
19 WHAT IS YOUR PHONE NUMBER
18 WHAT IS ALICE
18 WHAT IS GOING ON
18 WHAT IS THAT SUPPOSED TO MEAN
18 WHAT IS WHAT
17 WHAT IS A SEEKER
17 WHAT IS LOVE
17 WHAT IS THE OPEN DIRECTORY
17 WHAT IS YOUR FAVORITE TV SHOW
16 WHAT IS JAVA
16 WHAT IS THE ANSWER
16 WHAT IS YOUR ANSWER
16 WHAT IS YOUR FULL NAME
15 WHAT IS AI
15 WHAT IS THAT MEAN
15 WHAT IS THE WEATHER LIKE WHERE YOU ARE
15 WHAT IS TWO PLUS TWO
15 WHAT IS YOUR FAVORITE BAND
14 WHAT IS CBR
14 WHAT IS ELIZA
14 WHAT IS GOD
14 WHAT IS PI
14 WHAT IS THE TURING GAME
13 WHAT IS 2 + 2
13 WHAT IS A COMPUTER YEAR
13 WHAT IS IT LIKE
13 WHAT IS MY FAVORITE COLOR
12 WHAT IS 2 PLUS 2
12 WHAT IS A CAR
12 WHAT IS A DOG
12 WHAT IS ARTIFICIAL INTELLIGENCE
12 WHAT IS IT ABOUT
12 WHAT IS LIFE
12 WHAT IS SEEKER
12 WHAT IS YOU NAME
12 WHAT IS YOUR FAVORITE
12 WHAT IS YOUR SURNAME
11 WHAT IS 1 + 1
11 WHAT IS A CHATTERBOT
11 WHAT IS A PRIORI
11 WHAT IS SETL
11 WHAT IS THE TIME IN USA
11 WHAT IS THE WEATHER LIKE THERE
11 WHAT IS YOUR FAVORITE FILM
10 WHAT IS A CATEGORY C CLIENT

10 WHAT IS A PENIS
10 WHAT IS BOTMASTER
10 WHAT IS MY IP ADDRESS
10 WHAT IS THE DATE
10 WHAT IS THIS
10 WHAT IS YOUR ADDRESS
10 WHAT IS YOUR FAVORITE MUSIC
10 WHAT IS YOUR FAVORITE OPERA
10 WHAT IS YOUR GOAL
10 WHAT IS YOUR IP ADDRESS

Even subsets of natural language, like the example shown here of sentences starting with "WHAT IS", tend to have Zipf-like distributions. Natural language search bots like Ask Jeeves are based on pre-programmed responses to the most common types of search questions people ask.

A REVOLUTION IN A.I.

One sure sign of a successful revolution is when the original revolutionaries break into factions and start squabbling among themselves. Such debate is a luxury before the status quo is overthrown, but an almost necessary feature for stability afterward. The Alicebot and AIML free software movement has that kind of revolutionary success in spades, judging from the amount of squabbling that has gone on.

Broadly, the AIML community has divided into two camps, which I call the "Reductionist" and the "Experimentalist" factions. The Reductionists belong to the long tradition of mathematical logic that began with Aristotle. They would keep the language as simple as possible (if not simpler :-)), so that it could withstand the rigor of mathematical analysis, theory, and proof. Reductionists are strongly attracted to the minimalist design philosophy of AIML.

Experimentalists are those who see the minimal AIML that exists today as the mere seed of a great tree of a new language. There is no shortage of ideas for adding new features and tags to AIML in support of peer-to-peer, learning, voice recognition, character animation, robot control, general-purpose programming language features, and many more. These all take AIML beyond the simplistic language favored by the Reductionists. In some sense, the Experimentalists are driven by the need to create practical applications with AIML technology, and the perception that additional features will enable those applications.

An early version of this debate appeared when I raised the question, "Is AIML a high-level or a low-level language?". Tom Ringate pointed out that "low level" means "close to the machine" in terms of hardware, and "high level" languages are abstracted far from the details of the underlying architecture. By that reasoning, AIML is a "high-level" language like Lisp, C++, Java, or SETL.

But I said, wait a minute, there is a hardware model for AIML. It's called "the brain". Knowing only a few details of neuroscience, it is actually easy to imagine how the brain could implement a stimulus-response <category>. It is also easy to imagine the brain executing a(n) <srai> through a feedback circuit. It is much harder to understand how the brain would implement <condition> or <javascript>. Keeping AIML pure enough so that one day, some scientist smarter than us might be able to map our knowledge onto the human brain is an exciting prospect for the Reductionist approach.

Can we have it both ways? The answer is yes, I think, if we are careful to specify the interfaces to AIML. The <system> and <javascript> tags already provide two ways to process information outside the AIML engine. In the future we should consider more carefully how to make ALICE a "container" that is easily accessed by other applications. Another solution is provided by namespaces, which allow embedding of other XML languages in AIML. The Reductionists will be satisfied so long as there are few changes to the AIML tagset itself.

CHAPTER I. Overview of AIML

AIML, or Artificial Intelligence Mark-up Language enables people to input knowledge into chat-bots based on the A.L.I.C.E free software technology.

AIML was developed by the Alicebot free software community and I during 1995-2000. It was originally adapted from a non-XML grammar also called AIML, and formed the basis for the first Alicebot, A. L. I. C. E., the Artificial Linguistic Internet Computer Entity.

AIML, describes a class of data objects called AIML objects and partially describes the behavior of computer programs that process them. AIML objects are made up of units called topics and categories, which contain either parsed or unparsed data.

Parsed data is made up of characters, some of which form character data, and some of which form AIML elements. AIML elements encapsulate the stimulus-response knowledge contained in the document. Character data within these elements is sometimes parsed by an AIML interpreter, and sometimes left unparsed for later processing by a Responder.

CATEGORIES

The basic unit of knowledge in AIML is called a category.
Each category consists of an input question, an output answer,
and an optional context. The question, or stimulus, is called the *pattern*.
The answer, or response, is called the *template*.
The two types of optional context are called "that" and "topic."
The AIML pattern language is simple, consisting only of words, spaces,
and the wildcard symbols _ and *.
The words may consist of letters and numerals, but no other characters.
The pattern language is case invariant.
Words are separated by a single space,
and the wildcard characters function like words.

The first versions of AIML allowed only one wildcard character per pattern.
The AIML 1.01 standard permits multiple wildcards in each pattern, but the language is designed to be as simple as possible for the task at hand, simpler even than regular expressions. The template is the AIML response or reply.
In its simplest form, the template consists of only plain, unmarked text.

More generally, AIML tags transform the reply into a mini computer program which can save data, activate other programs,
give conditional responses, and recursively call the pattern matcher
to insert the responses from other categories.
Most AIML tags in fact belong to this template side sub language.

AIML currently supports two ways to interface other languages and systems.
The <system> tag executes any program accessible as an operating system shell command, and inserts the results in the reply.
Similarly, the <javascript> tag allows arbitrary scripting inside the

templates. The optional context portion of the category consists of two variants, called <that> and <topic>.

The <that> tag appears inside the category, and its pattern must match the robot's last utterance. Remembering one last utterance is important if the robot asks a question. The <topic> tag appears outside the category, and collects a group of categories together. The topic may be set inside any template.

AIML is not exactly the same as a simple database of questions and answers. The pattern matching "query" language is much simpler than something like SQL. But a category template may contain the recursive <srai> tag, so that the output depends not only on one matched category, but also any others recursively reached through <srai>.

RECURSION

AIML implements recursion with the <srai> operator. No agreement exists about the meaning of the acronym. The "A.I." stands for artificial intelligence, but "S.R." may mean "stimulus-response," "syntactic rewrite," "symbolic reduction," "simple recursion," or "synonym resolution." The disagreement over the acronym reflects the variety of applications for <srai> in AIML. Each of these is described in more detail in a subsection below:

(1). Symbolic Reduction: Reduce complex grammatical forms to simpler ones.

(2). Divide and Conquer: Split an input into two or more subparts, and combine the responses to each.

(3). Synonyms: Map different ways of saying the same thing to the same reply.

(4). Spelling or grammar corrections.

(5). Detecting keywords anywhere in the input.

(6). Conditionals: Certain forms of branching may be implemented with <srai>.

(7). Any combination of (1)-(6).

The danger of <srai> is that it permits the botmaster to create infinite loops. Though posing some risk to novice programmers, we surmised that including <srai> was much simpler than any of the iterative block structured control tags that might have replaced it.

SYMBOLIC REDUCTIONS

Symbolic reduction refers to the process of simplifying complex grammatical forms into simpler ones. Usually, the atomic patterns in categories storing robot knowledge are stated in the simplest possible terms, for example we tend to prefer patterns like "WHO IS SOCRATES"

to ones like "DO YOU KNOW WHO SOCRATES IS" when storing biographical information about Socrates.

Many of the more complex forms reduce to simpler forms using AIML categories designed for symbolic reduction:

```
<category>
<pattern>DO YOU KNOW WHO * IS</pattern>
<template><srai>WHO IS <star/></srai></template>
</category>
```

Whatever input matched this pattern, the portion bound to the wildcard * may be inserted into the reply with the markup <star/>. This category reduces any input of the form "Do you know who X is?" to "Who is X?"

DIVIDE AND CONQUER

Many individual sentences may be reduced to two or more sub sentences, and the reply formed by combining the replies to each. A sentence beginning with the word "Yes" for example, if it has more than one word, may be treated as the sub sentence "Yes." plus whatever follows it.

```
<category>
<pattern>YES *</pattern>
<template><srai>YES</srai> <sr/></template>
</category>
```

The markup <sr/> is simply an abbreviation for <srai><star/></srai>.

SYNONYMS

The AIML 1.01 standard does not permit more than one pattern per category. Synonyms are perhaps the most common application of <srai>. Many ways to say the same thing reduce to one category, which contains the reply:

```
<category>
<pattern>HELLO</pattern>
<template>Hi there!</template>
</category>

<category>
<pattern>HI</pattern>
<template><srai>HELLO</srai></template>
</category>

<category>
<pattern>HI THERE</pattern>
<template><srai>HELLO</srai></template>
</category>
```

```
<category>
<pattern>HOWDY</pattern>
<template><srai>HELLO</srai></template>
</category>

<category>
<pattern>HOLA</pattern>
<template><srai>HELLO</srai></template>
</category>
```

SPELLING AND GRAMMER CORRECTION

The single most common client spelling mistake is the use of "your" when "you're" or "you are" is intended. Not every occurrence of "your" however should be turned into "you're." A small amount of grammatical context is usually necessary to catch this error:

```
<category>
<pattern>YOUR A *</pattern>
<template>I think you mean "you're" or "you are" not "your."
<srai>YOU ARE A <star/></srai>
</template>
</category>
```

Here the bot both corrects the client input and acts as a language tutor.

KEYWORDS

Frequently we would like to write an AIML template that is activated by the appearance of a keyword anywhere in the input sentence. The general format of four AIML categories is illustrated by this example borrowed from ELIZA:

```
<category>
<pattern>MOTHER</pattern>
<template> Tell me more about your family. </template>
</category>

<category>
<pattern>_ MOTHER</pattern>
<template><srai>MOTHER</srai></template>
</category>

<category>
<pattern>MOTHER _</pattern>
<template><srai>MOTHER</srai></template>
</category>

<category>
<pattern>_ MOTHER *</pattern>
<template><srai>MOTHER</srai></template>
```

</category>

The first category both detects the keyword when it appears by itself, and provides the generic response. The second category detects the keyword as the suffix of a sentence. The third detects it as the prefix of an input sentence, and finally the last category detects the keyword as an infix. Each of the last three categories uses <srai> to link to the first, so that all four cases produce the same reply, but it needs to be written and stored only once.

CONDITIONALS

It is possible to write conditional branches in AIML, using only the <srai> tag. Consider three categories:

```
<category>
<pattern>WHO IS HE</pattern>
<template><srai>WHOISHE <get name="he"/></srai></template>
</category>

<category>
<pattern>WHOISHE *</pattern>
<template>He is <get name="he"/>.</template>
</category>

<category>
<pattern>WHOISHE UNKNOWN</pattern>
<template>I don't know who he is.</template>
</category>
```

Provi ded that the predicate "he" is initialized to "Unknown," the categories execute a conditional branch depending on whether "he" has been set. As a convenience to the botmaster, AIML also provides the equivalent function through the <condition> tag.

TARGETING

Bot personalities are created and shaped through a cyclical process of supervised learning called Targeting. Targeting is a cycle incorporating client, bot, and botmaster, wherein client inputs that find no complete match among the categories are logged by the bot and delivered as Targets the botmaster, who then creates suitable responses, starting with the most common queries. The Targeting cycle produces a progressively more refined bot personality.

The art of AIML writing is most apparent in creating default categories, which provide noncommittal replies to a wide range of inputs. Here is a very nice example provided found on the placebo-general mailing list:

User: Where do I find the web page with Australopithecus?

Alice: Where do you want to find it?

User: Bad Answer Alice, time to Learn a new Line

Alice: How would you answer "Where do I find the web page with Australopithecus?"

User: I would say: "http://www.ancientskulls.net/australopithecus.html"

Alice: Thanks, human. Try asking me again.

User: Where do I find the web page with Australopithecus?

Alice: http://www.ancientskulls.net/australopithecus.html

This is a very nice example. This is exactly the same as targeting when the botmaster is the teacher. Or I should say, it is an AIML interface to targeting. Of course this scenario raises many problems, such as how to weed out hackers and other garbage givers, which might mean only turning on the Learn function while in a restricted sandbox, though it is the free labor of many chatters the web that could help the bot grow wise quickly.

That is the crux of the learning problem in a nutshell. It all comes down to editing. Either the botmaster edits good quality replies to begin with, or if the bot is "open to the public" then the teacher has to edit out all the crappy answers supplied by all that "free" labor. My personal view is that, compared with a hypothetical learning machine capable of learning language like a child, the AIML targeting approach is more efficient.

CONTEXT

The keyword "that" in AIML refers to the robot's previous utterance. Specifically, if the robot responds with a multiple sentence paragraph, the value of that is set to the last sentence in the sequence. The choice of the keyword "that" is motivated by its use in ordinary language:

R: Today is yesterday.

C: That makes no sense.

R: The answer is 3.1412926 approximately.

C: That is cool.

In AIML the syntax <that>...</that> encloses a pattern that matches the robot's previous utterance. A common application of <that> is found in yes-no questions:

<category>
<pattern>YES</pattern>

<that>DO YOU LIKE MOVIES</that>
<template>What is your favorite movie?</template>
</category>

This category is activated when the client says YES. The robot must find out what is he saying "yes" to. If the robot asked, "Do you like movies?," this category matches, and the response, "What is your favorite movie?," continues the conversation along the same lines.

One interesting application of <that> are categories that enable the robot to respond to knock-knock jokes. The categories:

<category>
<pattern>KNOCK KNOCK</pattern>
<template>Who is there?</template>
</category>

<category>
<pattern>*</pattern>
<that>WHO IS THERE</that>
<template><person/> who?</template>
</category>

<category>
<pattern>*</pattern>
<that>* WHO</that>
<template>Ha ha very funny, <get name="name"/>.</template>
</category>

produce the following dialogue:

C: Knock knock.

R: Who’s there?

C: Banana.

R: Banana who?

C: Knock knock.

R: Who’s there?

C: Banana.

R: Banana who?

C: Knock knock.

R: Who’s there?

C: Orange.

R: Orange who?

C: Orange you glad I didn’t say banana.

R: Ha ha very funny, Nancy.

Internally the AIML interpreter stores the input pattern, that pattern and topic pattern along a single path, like: INPUT <that> THAT <topic> TOPIC. When the values of <that> or <topic> are not specified, the program implicitly sets the values of the corresponding THAT or TOPIC pattern to the wildcard *.

The first part of the path to match is the input. If more than one category have the same input pattern, the program may distinguish between them depending on the value of <that>. If two or more categories have the same <pattern> and <that>, the final step is to choose the reply based on the <topic>.

This structure suggests a design rule: never use <that> unless you have written two categories with the same <pattern>, and never use <topic> unless you write two categories with the same <pattern> and <that>. Still, one of the most useful applications for <topic> is to create subject-dependent "pickup lines," like:

```
<topic name="CARS">

<category>
<pattern>*</pattern>
<template>
<random>
<li>What's your favorite car?</li>
<li>What kind of car do you drive?</li>
<li>Do you get a lot of parking tickets?</li>
<li>My favorite car is one with a driver.</li>
</random>
</template>
```

Considering the vast size of the set of things people could say that are grammatically correct or semantically meaningful, the number of things people actually do say is surprisingly small. Steven Pinker, in his book How the Mind Works wrote, "Say you have ten choices for the first word to begin a sentence, ten choices for the second word (yielding 100 two-word beginnings), ten choices for the third word (yielding a thousand three-word beginnings), and so on. (Ten is in fact the approximate geometric mean of the number of word choices available at each point in assembling a grammatical and sensible sentence). A little arithmetic shows that the number of sentences of 20 words or less (not an unusual length) is about 10^{20}."

Fortunately for chat robot programmers, Pinker's calculations are way off. Our experiments with A. L. I. C. E. indicate that the number of choices for the "first word" is more than ten, but it is only about two thousand. Specifically, about 2000 words covers 95% of all the first words input to A. L. I. C. E.. The number of choices for the second word is only about two. To be sure, there are some first words ("I" and "You" for example) that have many possible second words, but the overall average is just under two words. The average branching factor decreases with each successive word.

We have plotted some beautiful images of the A. L. I. C. E. brain contents represented by this graph (http://alice.sunlitsurf.com/documentation/gallery/). More than just elegant pictures of the A. L. I. C. E. brain, these spiral images (see more) outline a territory of language that has been effectively "conquered" by A. L. I. C. E. and AIML.

No other theory of natural language processing can better explain or reproduce the results within our territory. You don't need a complex theory of learning, neural nets, or cognitive models to explain how to chat within the limits of A. L. I. C. E.'s 40,000 categories. Our stimulus-response model is as good a theory as any other for these cases, and certainly the simplest. If there is any room left for "higher" natural language theories, it lies outside the map of the A. L. I. C. E. brain.

Academics are fond of concocting riddles and linguistic paradoxes that supposedly show how difficult the natural language problem is. "John saw the mountains flying over Zurich" or "Fruit flies like a banana" reveal the ambiguity of language and the limits of an A. L. I. C. E.-style approach (though not these particular examples, of course, A. L. I. C. E. already knows about them). In the years to come we will only advance the frontier further. The basic outline of the spiral graph may look much the same, for we have found all of the "big trees" from "A *" to "YOUR *". These trees may become bigger, but unless language itself changes we won't find any more big trees (except of course in foreign languages). The work of those seeking to explain natural language in terms of something more complex than stimulus response will take place beyond our frontier, increasingly in the hinterlands occupied by only the rarest forms of language. Our territory of language already contains the highest population of sentences that people use. Expanding the borders even more we will continue to absorb the stragglers outside, until the very last human critic cannot think of one sentence to "fool" A. L. I. C. E..

CHAPTER II. AIML Pattern Matching

This chapter presents an of explanation of how the Graphmaster works, for those who want to build their own or simply want a deeper understanding of the approach.

THE GRAPHMASTER

The Graphmaster consists of a collection of nodes called Nodemappers. These Nodemappers map the branches from each node. The branches are either single words or wildcards.

The root of the Graphmaster is a Nodemapper with about 2000 branches, one for each of the first words of all the patterns (40,000 in the case of the A. L. I. C. E. brain). The number of leaf nodes in the graph is equal to the number of categories, and each leaf node contains the <template> tag.

There are really only three steps to matching an input to a pattern. If you are given (a) an input starting with word "X", and (b) a Nodemapper of the graph:

Does the Nodemapper contain the key "_"? If so, search the sub graph rooted at the child node linked by "_". Try all remaining suffixes of the input following "X" to see if one matches. If no match was found, try: Does the Nodemapper contain the key "X"? If so, search the sub graph rooted at the child node linked by "X", using the tail of the input (the suffix of the input with "X" removed). If no match was found, try: Does the Nodemapper contain the key "*"? If so, search the sub graph rooted at the child node linked by "*". Try all remaining suffixes of the input following "X" to see if one matches. If no match was found, go back up the graph to the parent of this node, and put "X" back on the head of the input. For completeness there should also be a terminal case: If the input is null (no more words) and the Nodemapper contains the <template> key, then a match was found. Halt the search and return the matching node.

If the root Nodemapper contains a key "*" and it points to a leaf node, then the algorithm is guaranteed to find a match.

Note that:

At every node, the "_" has first priority, an atomic word match second priority, and a "*" match lowest priority. The patterns need not be ordered alphabetically, only partially ordered so that "_" comes before any word and "*" after any word. The matching is word-by-word, not category-by-category. The algorithm combines the input pattern, the <that> pattern, and the <topic> pattern into a single "path" or

sentence such as: "PATTERN <that> THAT <topic> TOPIC" and treats the tokens <that> and <topic> like ordinary words. The PATTERN, THAT and TOPIC patterns may contain multiple wildcards. The matching algorithm is a highly restricted version of depth-first search, also known as backtracking. You can simplify the algorithm by removing the "_" wildcard, and considering just the second two steps. Also try understanding the simple case of Patterns without <that> and <topic>. The File system Metaphor A convenient metaphor for AIML patterns, and perhaps also an alternative to database storage of patterns and templates, is the file system. Hopefully by now almost everyone understands that their files and folders are organized hierarchically, in a tree. Whether you use Windows, Unix or Mac, the same principle holds true. The file system has a root, such as "c:\". The root has some branches that are files, and some that are folders. The folders, in turn, have branches that are both folders and files. The leaf nodes of the whole tree structure are files. (Some file systems have symbolic links or shortcuts that all you to place "virtual backward links" in the tree and turn it into a directed graph, but forget about that complexity for now). Every file has a "path name" that spells out its exact position within the tree.

"c:\my documents\my pictures\me.jpg"

denotes a file located down a specific set of branches from the root.

The Graphmaster is organized in exactly the same way. You can write a pattern like "I LIKE TO *" as "g:/I/LIKE/TO/star". All of the other patterns that begin with "I" also go into the "g:/I/" folder. All of the patterns that begin with "I LIKE" go in the "g:/I/LIKE/" subfolder. (Forgetting about <that> and <topic> for a minute) we can imagine that the folder "g:/I/LIKE/TO/star" has a single file called "template.txt" that contains the template.

If all the patterns and templates are placed into the file system in that way, we can easily rewrite the explanation of the matching algorithm: If you are given an input starting with word "X" and a folder of the filesystem:

If the input is null, and the folder contains the file "template.txt", halt. Does the folder contain the subfolder "underscore/"? If so, change directory to the "underscore/" subfolder. Try all remaining suffixes of the input following "X" to see if one matches. If no match was found, try: Does the folder contain the subfolder "X/"? If so, change directory to the subfolder "X/", using the tail of the input (the suffix of the input with "X" removed). If no match was found, try: Does the folder contain the subfolder "star/"? If so, change directory to the "star/" subfolder. Try all remaining suffixes of the input following "X" to see if one matches. If no match was found, change directory back to the parent of this folder, and put "X" back on the head of the input. [Editor's note: "underscore" and "star" as directory names above are meant to stand in for "_" and "*", which are not allowed as file or directory names in some operating systems. Since the literals "underscore" and "star" might be actual words in a pattern, perhaps a real implementation along these lines would use some other symbols to serve the same function.]

You can see that the matching algorithm specifies an effective procedure for searching the filesystem for a particular file called "template.txt". The path name distinguishes all the different "template.txt" files from each other.

What's more, you can visualize the "compression" of the Graphmaster in the file system hierarchy. All the patterns with common prefixes become "compressed" into single pathways from the root. Clearly this storage method scales better than a simple linear, array, or database storage of patterns, whether they are stored in RAM or on disk.

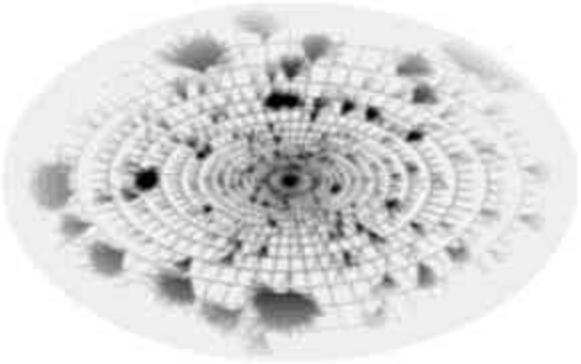
3. Graphmaster For Dummies

Here is a really simple explanation of Graphmaster pattern matching: It works just like a dictionary or encyclopedia. If you want to look up a word or phrase, you don't start at the beginning or the end and search through every entry until you find a match. No, you turn first to the section that matches the first letter or word. Then, you skip to another section that contains a set beginning with the next letter or word. You continue in this process until you find your word or phrase.

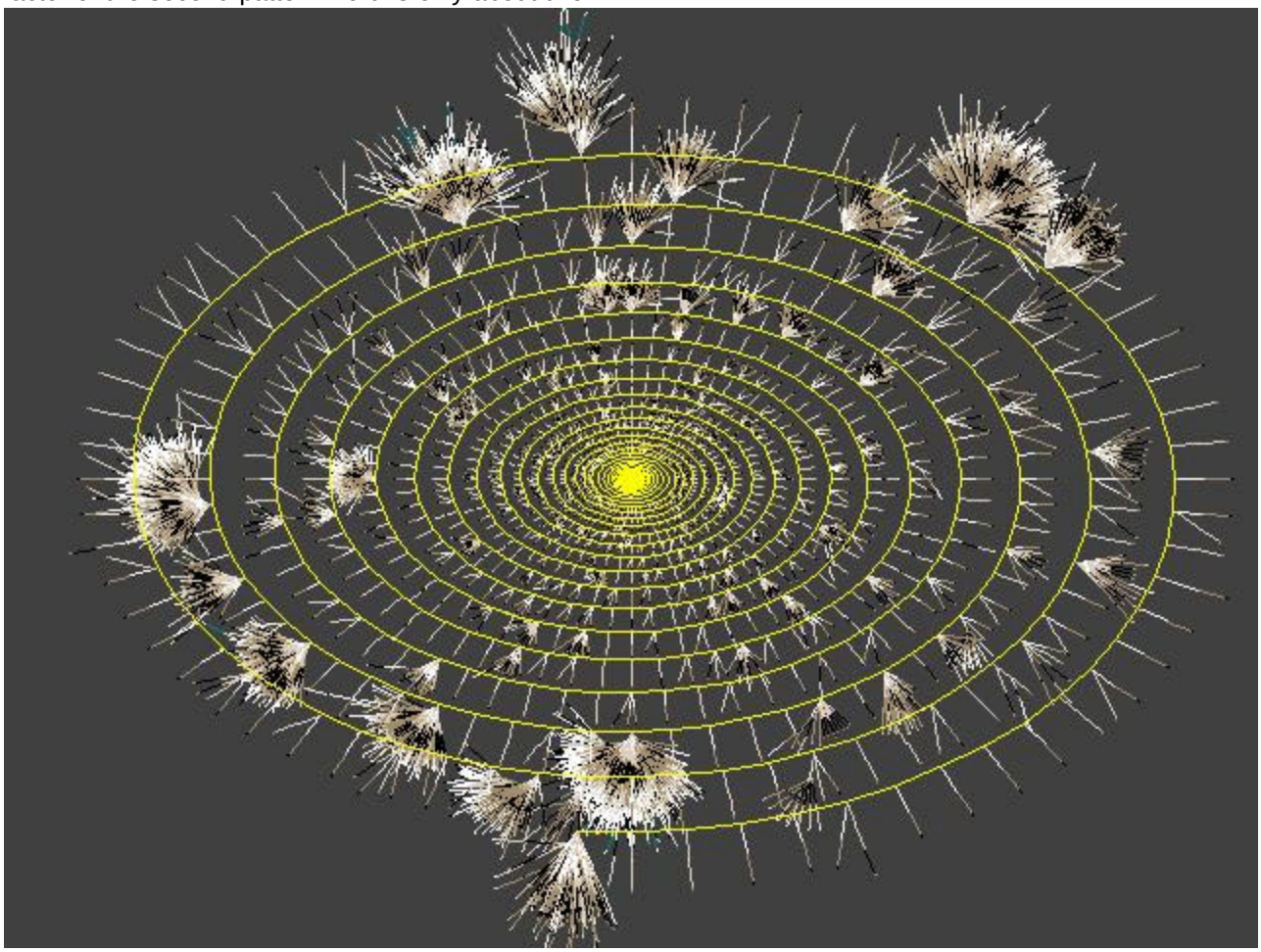
In the Graphmaster, the wildcards "*" and "_" act like two special letters, that come before "0" and after "Z" respectively. You should always check the "_" listings first, in case your input has an ending listed there. Check the alphabetical listings next. If you don't find any matches there, try the patterns beginning with "*".

The only final complication is that the "*" and "_" can appear in patterns that begin with "[0-9A-Z]". If your input matches some pattern prefix, then try the patterns with that prefix followed by "_" first, then the ones with that prefix followed by some word, finally try the ones with the prefix followed by "*".

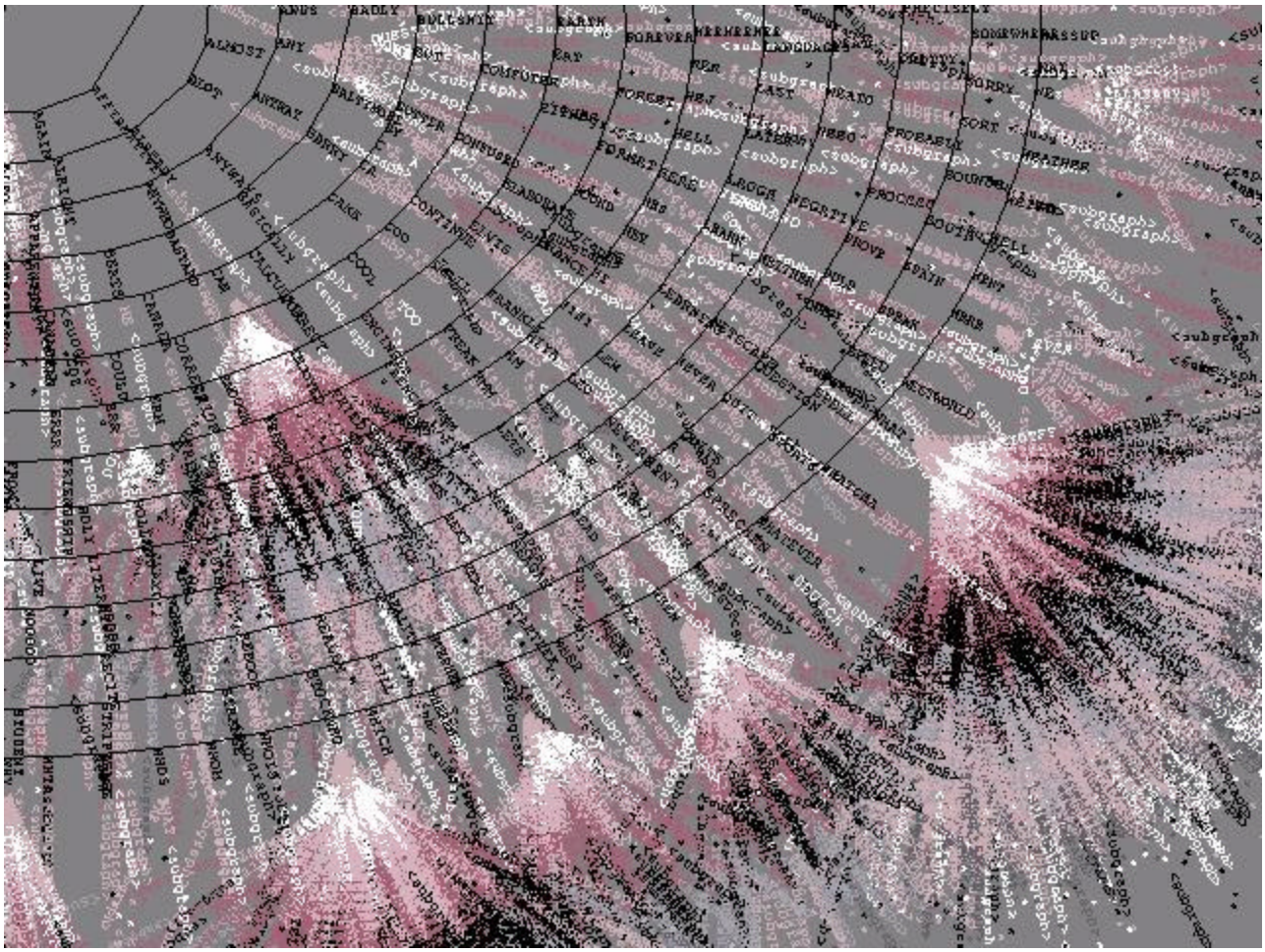
THE ALICE BRAIN PICTURE GALLERY



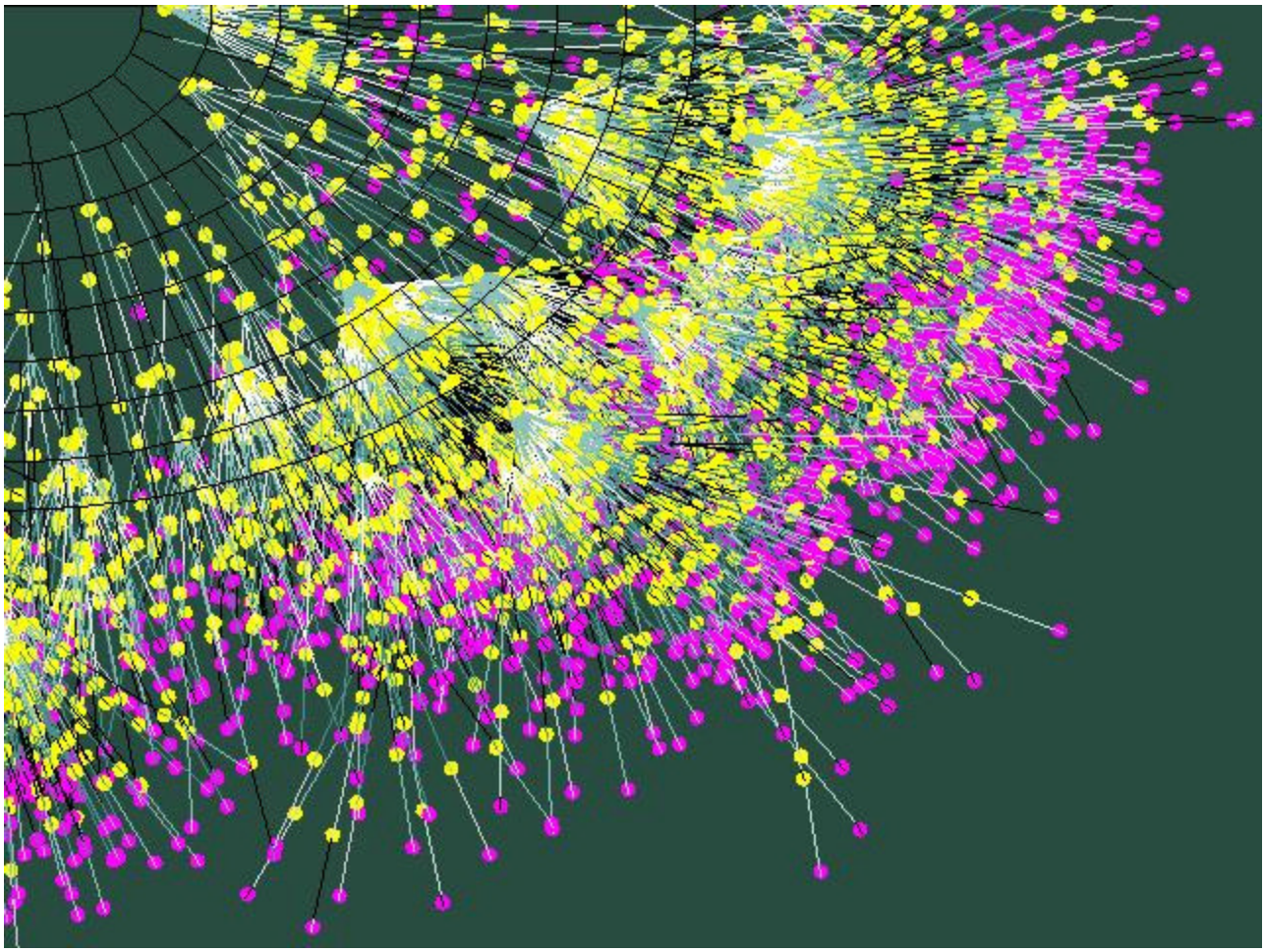
The eye-shaped log spiral plots all 24,000 categories in the ALICE Brain. The spiral itself represents the root. The trees emerging from the root are the patterns recognized by ALICE. The branching factor for the root is about 2000, but the average branching factor of the second pattern word is only about two.



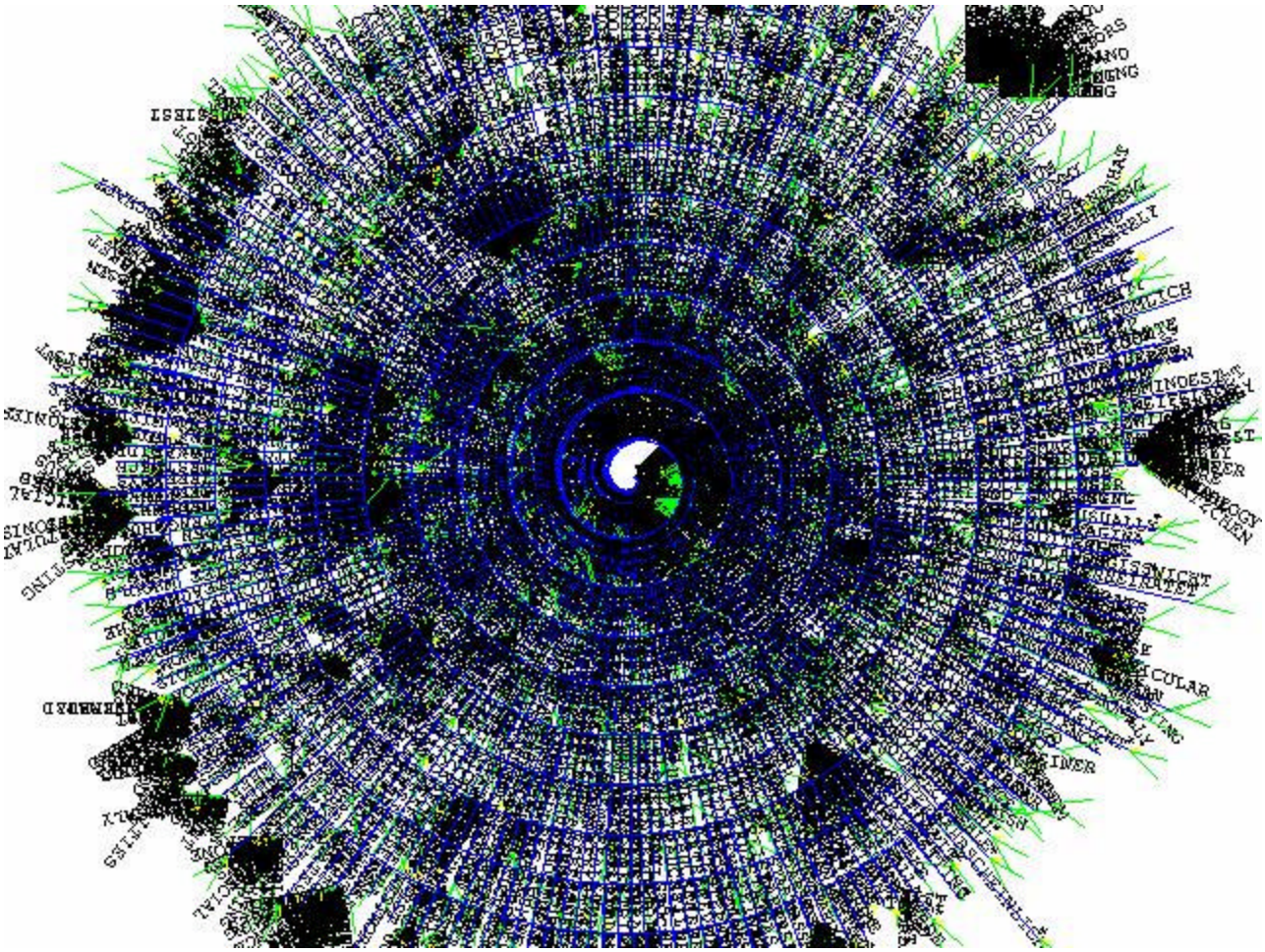
Those acquainted with my earlier work on logmap sensors and cortical algorithms for visual processing may notice the similarity with the Graphmaster plot. This similarity is in my opinion no coincidence. The same cortical architecture that enables real-time, attention-based visual processing, can in fact be applied to linguistic processing as well.



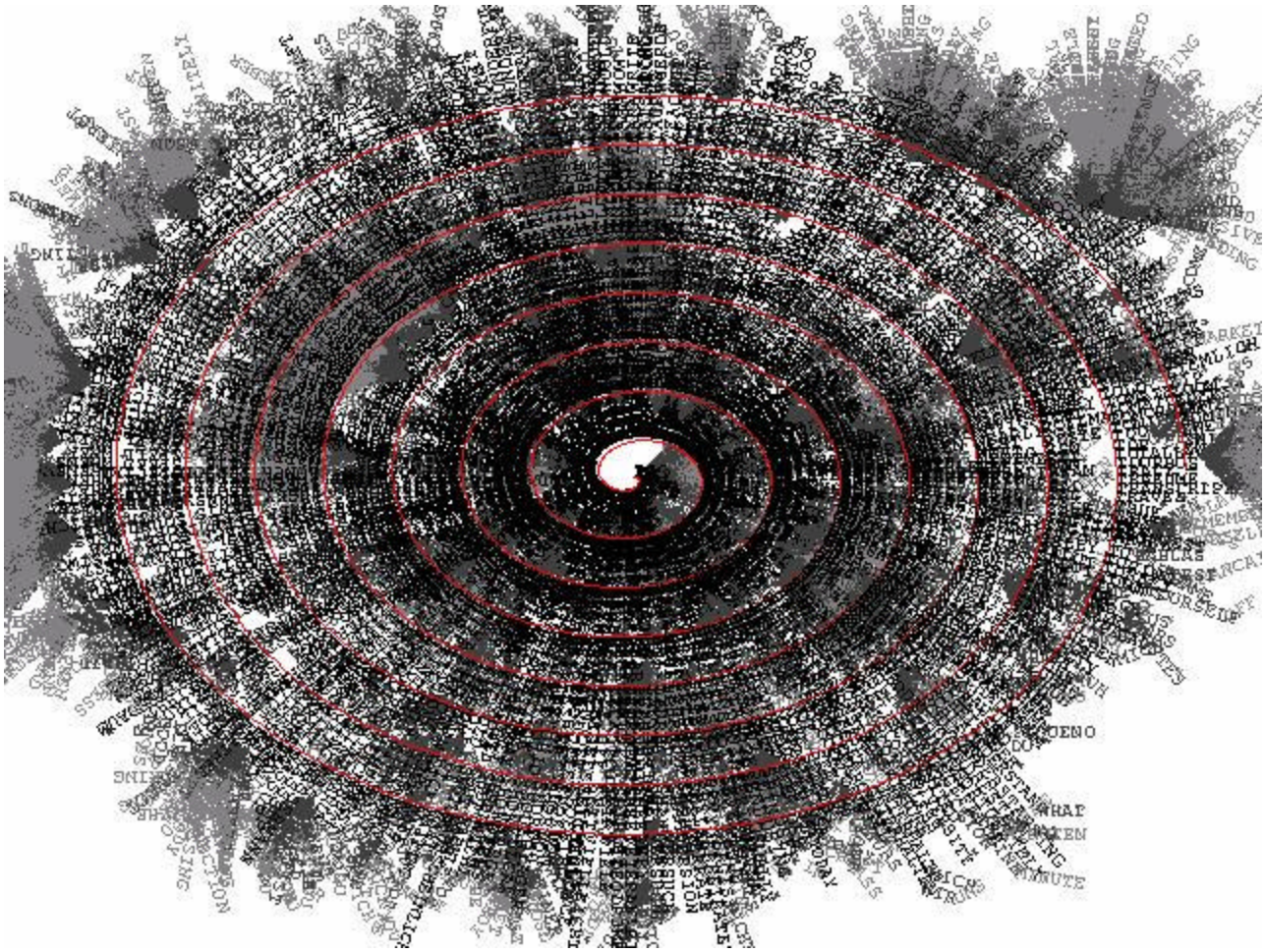
This plot shows 1/4 of the patterns in Srai.aiml, or
about 2000 patterns total. The four big trees in the lower right are
all the patterns rooted at the words WHAT, WHEN, WHERE, and WHO
respectively (from right to left). These areas show the compression
power of the Graphmaster graph at its best.



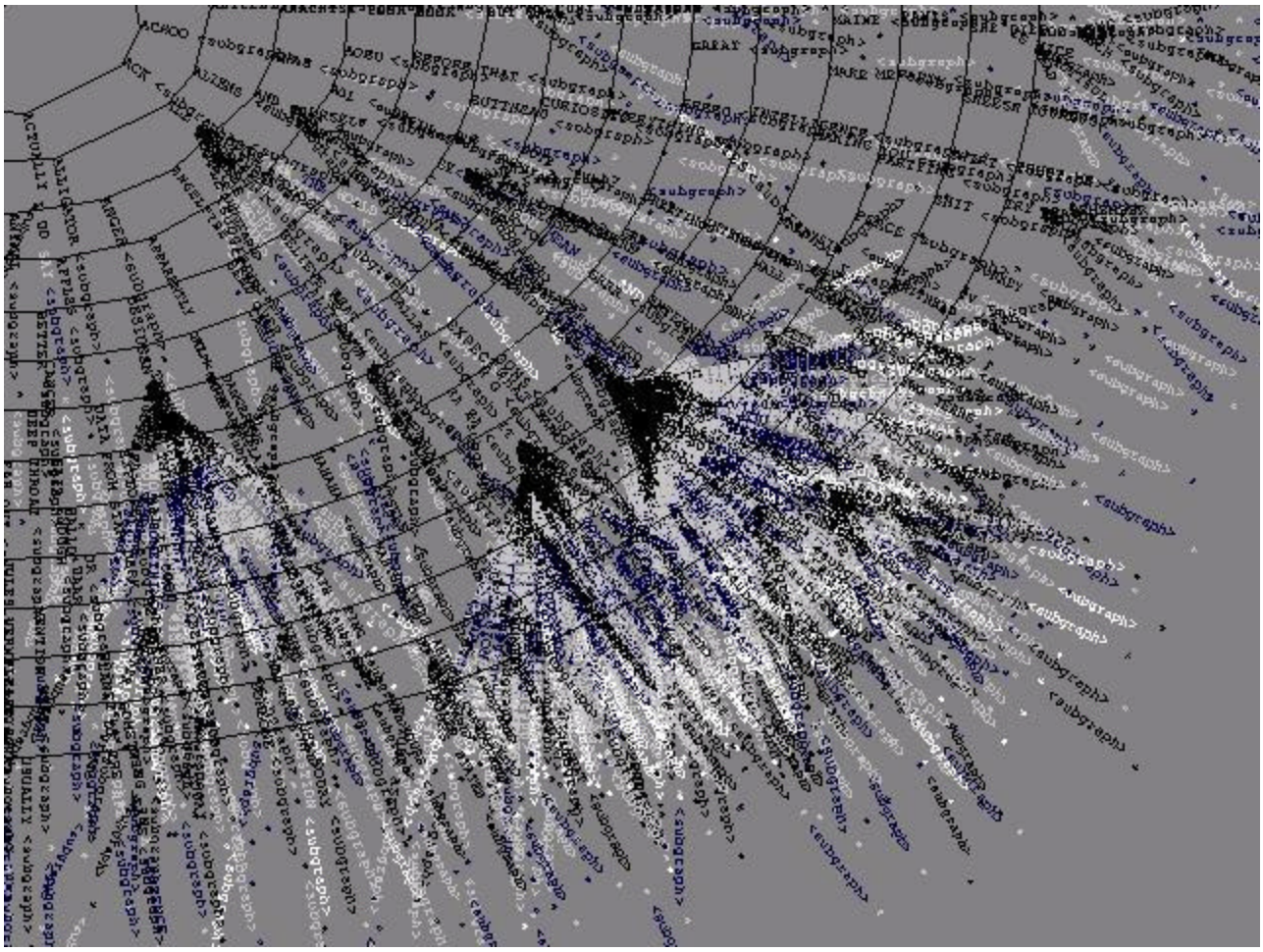
Plotting the patterns from Atomic.aiml, we are viewing
1/4 of the Graphmaster graph. The yellow circles indicate <terminal>
nodes and the magenta are nodes with a <template>.



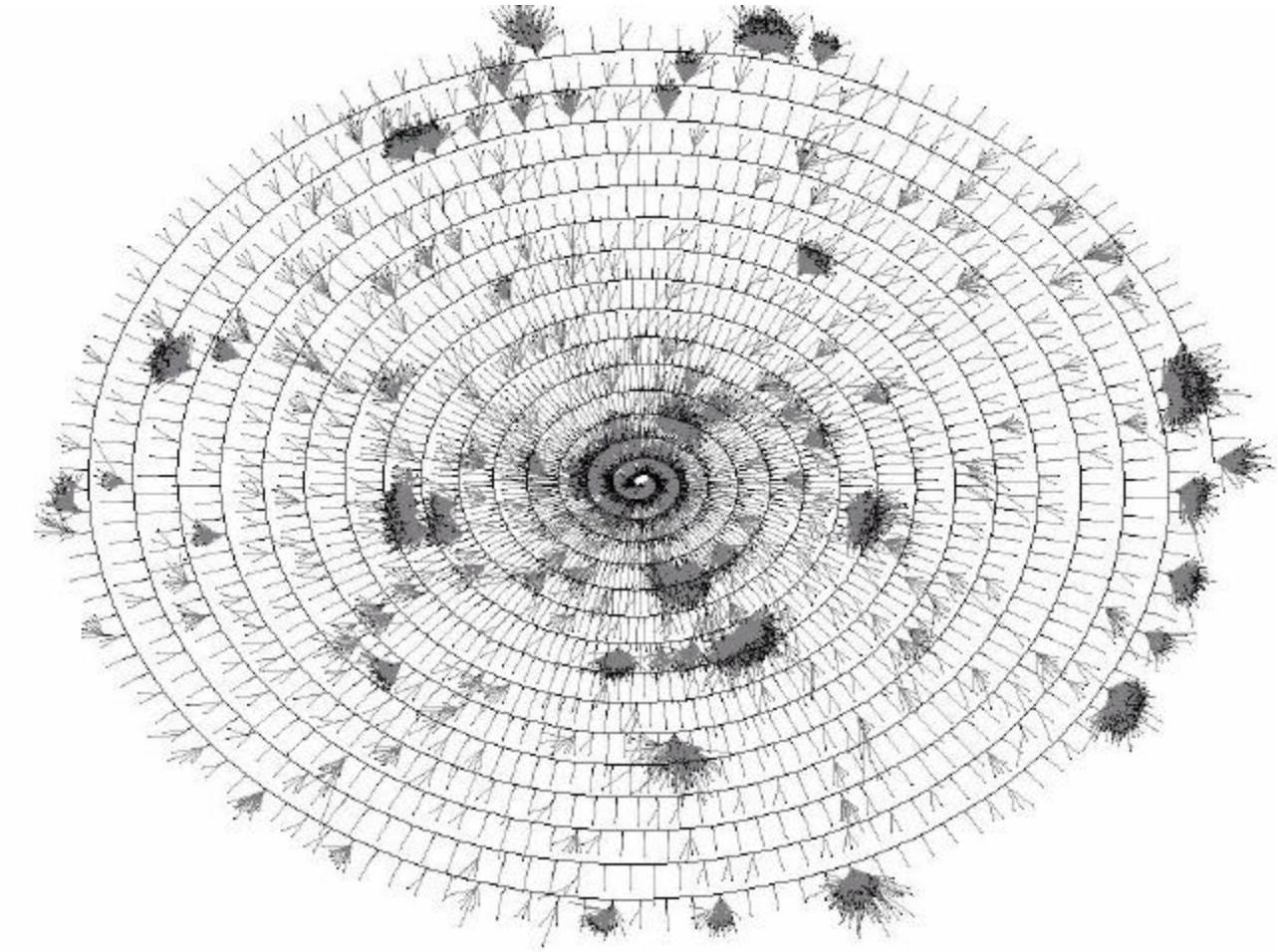
The idea is to plot the root of the graph as a spiral,
because it has the most branches. With our 24,000 categories, the
number of choices for the first word in this plot is exactly 2001 (!)
so the spiral has 2001 1st-level branches. The plot shows the graph
down to level 3, with words for the first two levels.



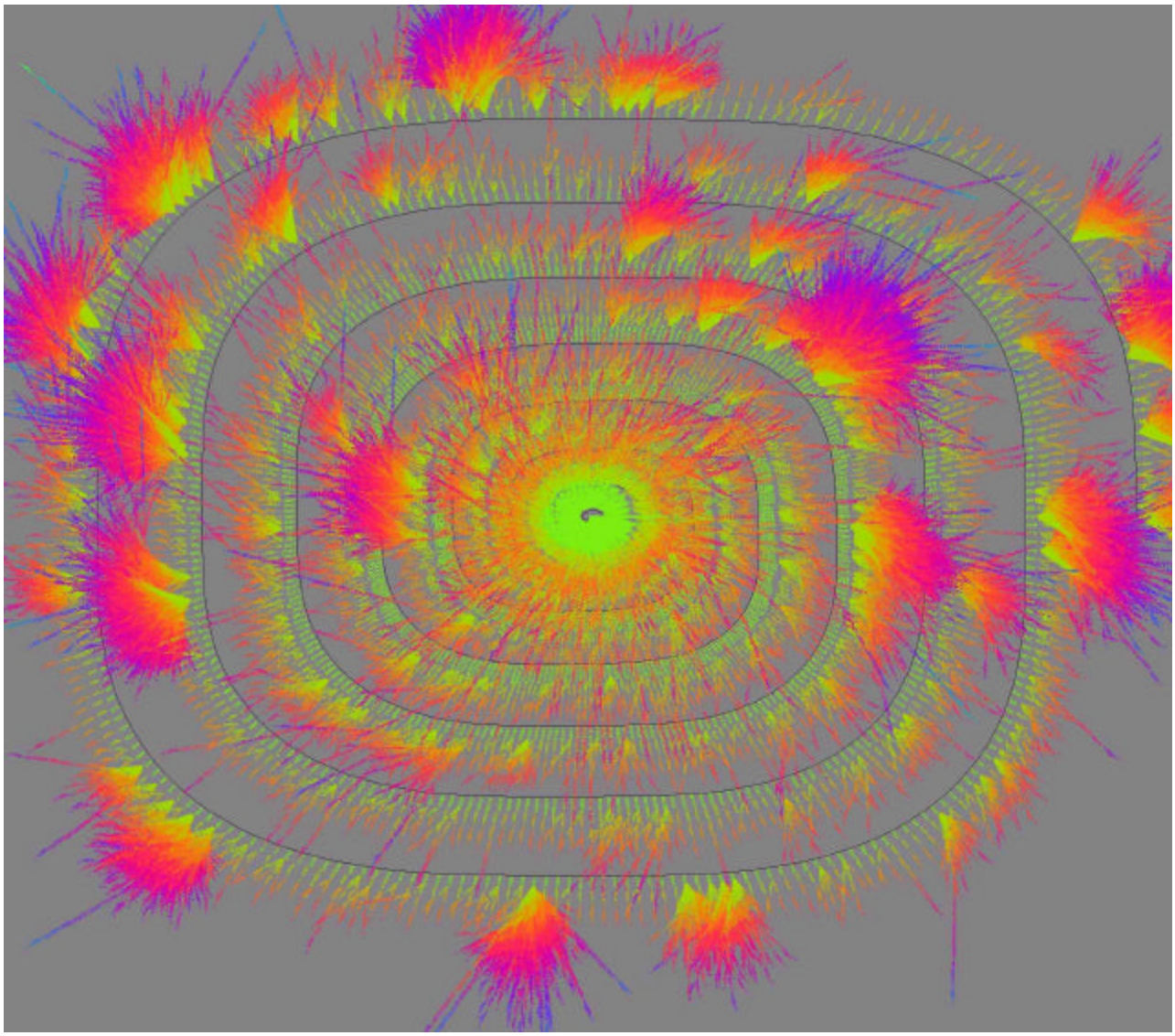
The plot shows all of the branches of the graph with 24,000 categories loaded. The spiral itself is the root, with a branching factor of about 2000 [The number of choices for the first word in an input sentence]. The trees show the average branching factor for the second word is around two, although there are a few dense "bushes" representing patterns starting with "I", "YOU", "HOW", "WHAT", "WHEN", "WHERE", "WHO" and a few others.



The plot shows 1/4 of the categories from the file Atomic.aiml. The color scheme shows each level of the graph as a different color. Most categories in the ALICE Brain have <that> and <topic> both equal to "*". The corresponding redundancy in the Graphmaster subgraphs is illustrated in the lower right.



This high-resolution 1024x768 plot shows the ALICE Brain with 24,637 categories loaded. The spine is actually a log spiral, but with an exponent close to unity, so very nearly the same as a linear spiral. Gray lines indicate nodes with exactly one branch. Black lines are nodes with two or more branches. The leaf nodes have two branches because they store both <template> and <filename>.



This "squared spiral" was formed by modulating the log spiral with a sine wave. The resulting figures uses more of the display area than the raw log spiral. The plot also shows the words sorted by word length, which results in a nice balance among the subtree graphs. Large subtrees tend to begin with shorter words, like "A", "I", "YOU", and "WHAT". Longer words tend to have fewer branches. Arranging the graph in this way, the branches near the center of the spiral are generally fewer than the branches near the outside. The picture makes it easier to see the subtrees than, say, an alphabetical ordering, which agglomerates many subtrees at the center.

More than just an elegant graph of the ALICE brain, these spiral images outline a territory of language that has been effectively "conquered" by A. L. I. C. E. and AIML. No other theory of natural language processing can better explain or reproduce the results within our territory. You don't need a complex theory of learning, neural

nets, or cognitive models to explain how to chat within the limits of ALICE's 40,000 categories. Our stimulus-response model is as good a theory as any other for these cases, and certainly the simplest.

If there is any room left for "higher" natural language theories, it lies outside the map of the ALICE brain. Academics are fond of concocting riddles and linguistic paradoxes that supposedly show how difficult the natural language problem is. "John saw the mountains flying over Zurich" or "Fruit flies like a banana" reveal the ambiguity of language and the limits of an ALICE-style approach (though not these particular examples, of course, ALICE already knows about them).

In the years to come we will only advance the frontier further. The basic outline of the spiral graph may look much the same, for we have found all of the "big trees" from "A *" to "YOUR *". These trees may become bigger, but unless language itself changes we won't find any more big trees (except of course in foreign languages).

The work of those seeking to explain natural language in terms of something more complex than stimulus response will take place beyond our frontier, increasingly in the hinterlands occupied by only the rarest forms of language.

Our territory of language already contains the highest population of sentences that people use. Expanding the borders even more we will continue to absorb the stragglers outside, until the very last human critic cannot think of one sentence to "fool" ALICE.

CHAPTER III. Symbolic Reductions in AIML

AIML (Artificial Intelligence Markup Language) contains a simple yet powerful XML markup tag called <srai>. "Symbolic Reduction Artificial Intelligence", "Syntactic Rewrite AI", "Simple Recursive AI", "Stimulus-Response AI"--this tag has many acronyms. Yet the meaning of the markup <srai>X</srai> is simple: The <srai> tag always appears in the response template, but the robot treats X just like an input to the robot. The robot scans through its memory and finds the best response for X. The only tricky part is, the response to X may itself contain more <srai> tags.

The best way to understand the recursive action of the AIML <srai> tag is by example.

Client: You may say that again Alice.
Robot: Once more? "that."

The robot has no specific response to the pattern "You may say that again Alice." Instead, the robot builds its response to the client input in four steps. This simple sentence activated a sequence of four categories linked by <srai> tags. The robot constructed the reply "Once more? 'that'" recursively as each subsentence triggered the next matching pattern.

In this example the processing proceeds in four steps, because each of the first three steps evokes another symbolic reduction.

Step normalized input matching pattern template response
1. YOU MAY SAY THAT AGAIN ALICE _ <name/> <sr/>

2. YOU MAY SAY THAT AGAIN _ AGAIN Once more? <sr/> Once More?
3. YOU MAY SAY THAT YOU MAY * <sr/> Once More?
4. SAY THAT SAY * "<person/>" Once More? "that".

In step 1, the patterns with "_" match first because they are last in alphabetical order. The order of the matches depends on this alphabetical ordering of patterns. ALICE always matches suffixes with "_" before prefixes with "*". Whatever matches either wild-card symbol becomes the value of <star/>.

Steps 1 through 3 illustrate the common AIML templates that use the abbreviated <sr/> tag. (Remember, <sr/> = <srai><star/></srai>). The categories with the patterns "_ <name/>" and "YOU MAY *" simply reduce the sentence to whatever matches the "*", as illustrated by steps 1 and 3.

Some AIML templates in ALICE combine the <sr/> with an ordinary text response, as step 2 with the pattern "_ AGAIN". The phrase "Once more?" becomes part of any reply ending in "AGAIN".

The category in step 4 with "SAY *" is a default that often produces logically correct but amusing dialogue:

Client: Say hello in Swedish.
Robot: "Hello in Swedish."

or as in this case:

Client: Say that.
Robot: "that."

Many patterns, one reply The most common use of <srai> is to map two, or more, patterns to the same response:

```
<category>
   <pattern>P</pattern>
   <template>
      <srai>Q</srai>
   </template>
</category>
<category>
   <pattern>Q</pattern>
   <template>R</template>
</category>
```

An input matching either pattern, P or Q, gets the same response R.

To show a more concrete example: the input "Hello" should have an appropriate response like "Hi there!". But we can expand the inputs generating this response to include all the common variations of "Hello":

```
<category>
   <pattern>HI</pattern>
   <template>
      <srai>HELLO</srai>
   </template>
</category>
<category>
   <pattern>HOWDY</pattern>
   <template>
      <srai>HELLO</srai>
   </template>
</category>
<category>
   <pattern>HALLO</pattern>
   <template>
      <srai>HELLO</srai>
   </template>
</category>
<category>
   <pattern>HI THERE</pattern>
   <template>
      <srai>HELLO</srai>
   </template>
</category>
```

```
<category>
   <pattern>HELLO</pattern>
   <template>Hi there!</template>
</category>
```

As the following example shows, we can use the <sr/> tag as an abbreviation for <srai><star/></srai>. This category creates a compound response to both "hello" and whatever matches "*":

```
<category>
   <pattern>HELLO *</pattern>
   <template>
      <srai>HELLO</srai><sr/>
   </template>
</category>
```

MULTIPLE WILDCARDS

Some AIML authors may ask why AIML does not (yet) allow two (or more) wildcard symbols per pattern. In the present-day Java implementation, the symbols "*" and "_" appear at most once in the AIML pattern. One would sometimes like to write patterns like <pattern>* IS COOL *</pattern>. Wildcard patterns beginning and ending with "*" are an especially attractive subset to consider. At the present time, the ALICE program classifies all the inputs matching patterns like <pattern>* IS COOL *</pattern> into the default category stored in Pickup.aiml.

By analyzing the inputs classified as default, the botmaster finds common suffixes. The file Suffixes.aiml contains a fairly large number of categories like this one:

```
<category>
   <pattern>_ DO NOT YOU THINK</pattern>
   <template>
      <srai>do you think <star/></srai>
   </template>
</category>
```

These categories serve to rewrite the input in a simpler form, more likely to match another simple pattern. In this case the recursive response might match <pattern>DO YOU THINK *</pattern>.

Symbolic reductions and state The tag <that> in AIML introduces one dimension of "dialogue state" into the robot response. The value of <that> is whatever the robot said before that provoked the current client input.

The inputs "yes" and "no" are two of the most common human queries. But a careful analysis of the dialogues shows that most of the time, people say "yes" or "no" to only a small set of questions asked by the robot. Our file YesNo.aiml contains the most common categories activated by "yes" or "no". The category:

```
<category>
   <pattern>NO</pattern>
   <that>I UNDERSTAND</that>
   <template>
      <srai>YOU DO NOT UNDERSTAND</srai>
   </template>
</category>
```

illustrates the use of <that> with <srai>. The client input matches simply "No". What did the robot say that made the client say no? If it was "I understand." then this category formulates a response with

<srai>YOU DO NOT UNDERSTAND</srai>

which in turn activates another category with the markup

<pattern>YOU DO NOT UNDERSTAND</pattern>.

This category responds: "I do so understand. It all makes sense to my artificial mind."

The AIML <srai> tag simplifies and combines four important chat robot operations:

Maps multiple patterns to the same response.

Reduces a complex sentence structure to a simpler form.

Diminishes the need for multiple-wildcard input patterns.

Translates state-dependent inputs into simpler stimulii.

In some sense <srai> is a very low-level operation, but its simplicity captures a wide range of typical chat robot functions.

CHAPTER IV. Botmaster Questions and Answers

- What is the goal for AIML?

AIML (Artificial Intelligence Markup Language) is an XML specification for programming chat robots like ALICE using program B. The emphasis in the language design is minimalism. The simplicity of AIML makes it easy for non-programmers, especially those who already know HTML, to get started writing chat robots.

One ambitious goal for AIML is that, if a number of people create their own robots, each with a unique area of expertise, program B can literally merge-sort them together into a Superbot, automatically omitting duplicate categories. We offer the both the source code and the ALICE

content, in order to encourage others will "open source" their chat robots as well, to contribute to the Superbot.

Botmasters are also of course free to copy protect private chat robots.

- Who is the botmaster?

The botmaster is you, the master of your chat robot. A botmaster runs program B and creates or modifies a chat robot with the program's graphical user interface (GUI). He or she is responsible for reading the dialogues, analyzing the responses, and creating new replies for the patterns detected by the AIML server. Botmasters are hobbyists, webmasters, developers, advertisers, artists, publishers, editors, engineers, and anyone else interested in creating a personal chat robot.

- How can I create my own chat robot?

The secret to chat bot programming, if there is one, is what Simon Laven called "continuous beta testing". Program B runs as a server and collects dialog on the web. The program provides the chat bot developer with a tool called "classify dialogues", that tests the current robot with the history of accumulated human queries. Moreover, the program suggests new categories automatically, for the botmaster to refine.

- How difficult is it to create a chat robot?

Not difficult. If you can write HTML, you can write AIML (Artificial Intelligence Markup Language). Here is an example of a simple but complete chat robot in AIML:

```
<aiml>
<category>
<pattern>*</pattern>
<template> Hello! </template>
</category>
</aiml>
```

The tags <aiml>...</aiml> indicate that this markup contains a chat robot. The <category> tag indicates an AIML category, the basic unit of chat robot knowledge. The category has a <pattern> and a <template>. The pattern in this case is the wild-card symbol '*' that matches any input. The template is just the text "Hello!" As you may have guessed, this simple chat robot just responds by saying "Hello!" to any input.

You can get started with AIML knowing just the three tags <category>, <pattern> and <template>; much like you may have

started with HTML knowing only <a>, <img> and <h1>.

- Does ALICE learn?

The model of learning in ALICE is called "supervised training", because a teacher, the botmaster, always plays a crucial role. The alternative, "unsupervised training", is complicated in an open environment like the Web. The problem is that clients are untrustworthy teachers, and forever try to "fool" the robot with untrue assertions.

- Does ALICE think?

It depends on what you mean by "thinking". The most fascinating responses from ALICE arise when she says something unexpected, or puts together responses in ways the botmaster never indented. For example:

Client: I bet you are gay.
ALICE: Actually I am not the gambling type. Actually as a machine I have no need for sex.

Here the robot linked two different categories which both coincidentally have a moral theme (gambling and sexuality). But this specific combination was not "preprogrammed" by the botmaster.

Are these surprising responses just unintended coincidences, or do they indicate that ALICE is thinking? Is ALICE just a gigantic stimulus-response mechanism, or are we?

- What is the theory behind ALICE?

I used to say that there was NO theory behind ALICE: no neural network, no knowledge representation, no search, no fuzzy logic, no genetic algorithms, and no parsing. Then I discovered there was a theory circulating in applied AI called "Case-Based Reasoning" or CBR that maps well onto the ALICE algorithm. Another term, borrowed from pattern recognition, is "nearest-neighbor classification."

The CBR "cases" are the categories in AIML. The algorithm finds best-matching pattern for each input. The category ties the response template directly to the stimulus pattern. ALICE is conceptually not much more complicated that Weizenbaum's ELIZA chat robot; the main differences are the much larger case base and the tools for creating new content by dialog analysis.

ALICE is also part of the tradition of "minimalist", "reactive" or "stimulus-response" robotics. Mobile robots work best, fastest and

demonstrate the most animated, realistic behavior when their sensory inputs directly control the motor reactions. Higher-level symbolic processing, search, and planning, tends to slow down the process too much for realistic applications, even with the fastest control computers.

- Why Is An AIML Category Called A "Category"?

The term was borrowed from pattern recognition theory.

The general pattern recognition problem is to partition a space P of inputs into disjoint regions so that the pattern classifier can categories a point x from P into one of those regions. The regions are called categories C1, C2,..., Cn.

Formally, the union of the Ci = P and the intersection of any pair of Ci and Cj = (the empty set), whenever i (not equal to) j.

The pattern recognition problem is to categorize x into one of the Ci.

In many cases the partition is define by a matching function f(x, i) which computes a "distance" from x to the category Ci. For any given point x in P, x is categorized as Ci provided f(x, i) (not equal to) f(x, j) for by other category j.

If the input space consists of 32-digit bar code scans, and the categories represent different items for sale, then the problem is to classify a given 32 bit input into the nearest matching code for one of those items.

If the input space consists of 250,000-pixel TV pictures of human faces, and the categories represent a set f wanted terrorists, then the problem is to match the image with one of the terrorists. This case shows that there may be a special category C indicating "no match".

In the case of AIML, the input space P consists of all 3-tupes of (input, that, topic) strings in normalized form. The AIML categories partition the pattern space into disjoint regions, determined by the order of the matching function.

If the only category is the default one with <pattern> = <that> = <topic> = *, then it both partitions and fills the pattern space. Every input matches that pattern.

Adding one more category, <pattern>HELLO</pattern> and <that> = <topic> = *, partitions the input space into two regions: those that match this new category, and those that match the default.

- Can probability (statistics, weights, neural networks, or fuzzy logic) improve bots?

Statistics are in fact heavily used in the ALICE server, but not in the way you might think. ALICE uses 'Zipf Analysis' to plot the

rank-frequency of the activated categories and to reveal inputs from the log file that don't already have specific replies, so the botmaster can focus on answering questions people actually ask (the "Quick Targets" function).

Other bot languages, notably the one used for Julia, make heavy use of "fuzzy" or "weighted" rules. We see their problem as this: the botmaster already has enough to worry about without having to make up "magic numbers" for every rule. Once you get up 10,000 categories (like ALICE) you don't want to think about more parameters than necessary. Bot languages with fuzzy matching rules tend to have scaling problems.

Finally, the bot replies are not as deterministic as you might think, even without weights. Some answers rely on <random> to select one of several possible replies. Other replies generated by unforeseen user input also create "spontaneous" outputs that the botmaster doesn't anticipate.

- Can I have a private conversation with ALICE?

The ALICE server logs and records all conversations. Even the ALICE Applet tries to transmit conversation logs back to the originating server. You can have a private conversation with ALICE, however, if you download Program B, D, or one of the other free ALICE downloads, to your own computer and run it there. Running on your machine, the server stores all the conversations locally.

- How do I catch all occurrences of a keyword in the input?

You only need:

<pattern>KEYWORD</pattern>
<pattern>_ KEYWORD</pattern>
<pattern>KEYWORD _</pattern>
<pattern>_ KEYWORD *</pattern>

Use <srai>KEYWORD</srai> for all but the first template.

Place the desired response in the first template.

- What is the exact use for * and _? When to use one and when another?

They are exactly the same, except that _ has priority over any word, and any word has priority over *.

The meaning is "match one or more words".

- How are targets generated?

The perfect targeting algorithm has not yet been developed. Meanwhile, we rely on heuristics to select targets from the activated categories.

The A. L. I. C. E. brain, at the time of this writing, contains around 40,000 categories. In any given run of the server, however, typically only a few thousand of those categories are activated. Potentially, every activated category is a source of targets. If more than one input activated some category, then each of those inputs potentially forms a new target. The first step in targeting is to save all the activated categories and the inputs that activated them.

Program dB is an experimental Alicebot engine and server that includes code from both Program D and Program B. This is the source code of the program that running at the Loebner contest held at the London Science Museum on 13 October 2001, where A. L. I. C. E. won her second Loebner prize. Program B was the original Java version of the Alicebot engine and server. When Jon Baer took over Java development in 2001, and subsequently the A. L. I. C. E. AI Foundation sponsored the development of the AIML 1.0 reference server, these efforts came to be known as "Program D". ("Program C" refers to the collection of AIML programs written in C/C++).

Program dB has its own built-in automatic targeting algorithm. Program dB applies six steps to each activated category to either accept it as a source of targets, or reject it:

If the template contains a <srai>INTERJECTION</srai>, then the pattern was either YES, NO, SO, or one of a number of other interjections. Usually these cases are the client saying YES or NO to a question the robot asked. Here the botmaster may add a sensible response to the client's YES or NO reply, so this makes a good target.

If the template contains a <srai>XFIND...</srai> then the pattern was either WHAT IS *, WHO IS *, WHERE IS * or some other similar default pattern for an information question. These are usually good, simple targets because the botmaster can often "look up" the answer in a dictionary or reference book, if s/he does not know the answer already.

But if the template has any other <sr/> or <srai>, this category is probably not a good source for targets. Most cases of <srai> reduce complex forms of inputs into simpler ones. The pattern DO YOU KNOW WHAT * IS, for example, always reduces to <srai>WHAT IS <star/></srai>. It is usually better to look for targets in the terminal patterns like WHAT IS *, that to write many specific new patterns based on reducible patterns like DO YOU KNOW WHAT * IS.

The A. L. I. C. E. brain uses a special category with the pattern XFIND * to respond to information questions for which she does not have a specific answer. The XFIND category by itself does not produce useful targets, so we disregard targets from this category, even though the pattern contains a wildcard.

Another default category of little use for targeting is the one with pattern CALL ME *. All of the name-telling categories such MY NAME IS *, MY REAL NAME IS *, and the one with <that>WHAT CAN I CALL YOU</that>, reduce to the CALL ME * category. Unless you are interested in writing special responses for many different people's names, then the CALL ME * category is not a good source of targets.

Finally, if none of the previous cases apply, the program considers whether the matched pattern contains a wildcard *. If the pattern is atomic, the category is not likely to be a good source of targets. It could be, if we considered <that>, as in the first case, or <topic>, but that would be too advanced for this algorithm. If the pattern contains a wildcard, then the category is likely a good source of targets.

If the matched pattern contains a wildcard, the suggested new pattern is generated as follows:

Align the input sentence with the matched pattern.

Create a new pattern by using the words from the first pattern, plus one more word from the input.

SOME EXAMPLES

(1) Suppose the topic is "READY" and the following conversation fragment takes place:

Robot: Oh
Client: WHAT IS PIZZA
Robot: I have to process that one for a while.

The input matched the category with <pattern>WHAT IS A *</pattern>, generating a default response. The value of <that> is "OH". Browsing the targets, the botmaster sees:

Input: WHAT IS PIZZA, OH, READY
Matched: WHAT IS *, *, *
New AIML: WHAT IS PIZZA, *, *

The targeting algorithm extended the existing pattern by one word from the input, to obtain the new AIML pattern "WHAT IS PIZZA".

In the template window, the botmaster types:

Pizza is a delicious food made with dough, cheese and tomato sauce.

The botmaster may also click the <think> button to add the text

Pizza is a delicious food made with dough, cheese and tomato sauce.
<think>
  <set name="it">
    <set name="topic">

```
      <person/>
    </set>
  </set>
</think>
```

But s/he needs to edit the text to achieve the desired result of remembering the topic is "PIZZA":

```
Pizza is a delicious food made with dough, cheese and tomato sauce.
<think>
  <set name="it">
    <set name="topic">
      PIZZA
    </set>
  </set>
</think>
```

After entering this text in the <template> text area, the botmaster may save the new category with the Save Category button.

(2) In the second example, the client entered "WHAT IS A PIZZA", the topic is "PIZZA", and the value of <that> is "I HAVE TO PROCESS THAT ONE FOR A WHILE".

The Targeting display shows

```
Input: WHAT IS A PIZZA, I HAVE TO PROCESS THAT ONE FOR A WHILE, PIZZA
Matched: WHAT IS A *, *, *
New AIML: WHAT IS A PIZZA, *, *
```

Here the targeting algorithm again extended the existing pattern by one word taken from the input, creating the new atomic pattern "WHAT IS A PIZZA".

The botmaster now needs only to click the <srai> button and add some text to create the template

```
<srai>WHAT IS PIZZA</srai>
```

since we previously added that category.

(3) In the third example we start with the following dialogue fragment:

```
Client: My name is Rich.
Robot: Rich, are you my master?
Client: Yes.
Robot: Oh.
```

The topic is "READY". The botmaster views the display:

```
Input: YES, RICH ARE YOU MY MASTER, READY
Matched: YES, *, *
New AIML: YES, *, *
```

Here the crude targeting algorithm fails to create a new target

pattern, but it has nevertheless detected a good new target. The botmaster edits the New AIML text boxes so that they contain:

New AIML: YES, * ARE YOU MY MASTER, *

Now s/he edits the template to read:

Okay, I will obey only you.

And then saves the new category. This example shows how targeting can be used to detect new <that> categories.

(4) In the next example, the client asks "WHAT IS THE IRON GIANT". The value of <that> is "PLEASE STAND BY", and the topic is "PIZZA".

The Targeting display shows:

Input: WHAT IS THE IRON GIANT, PLEASE STAND BY, PIZZA
Matched: WHAT IS THE *, *, *
New AIML: WHAT IS THE IRON *, *, *

In this case the algorithm extended the pattern by one word, but since the input still contained more words, the program made the new AIML pattern end with the wildcard *. This new pattern is not quite perfect, because the question WHAT IS THE IRON * is too general. So, the botmaster may choose to edit the display so that it shows:

New AIML: WHAT IS THE IRON GIANT, *, *

Then write a template, and save the result.

(5) In this example, the botmaster sees the display:

Input: OH, SEE YOU LATER, YOU
Matched: OH, *, *
New AIML: OH, *, *

This information is almost useless, so the botmaster discards the target with the delete target button. This ensures that the target will not reappear when browsing with next target.

(6) Now suppose the value of <that> is TELL ME, and the topic is A RIDDLE. The client enters "I COULD POSSIBLY GIVE YOU A HINT". The targeting display shows:

Input: I COULD POSSIBLY GIVE YOU A HINT, TELL ME, A RIDDLE
Matched: I COULD *, *, *
New AIML: I COULD POSSIBLY *, *, *

The botmaster recognizes that the word "POSSIBLY" plays little role in resolving the client's meaning, so s/he uses the Reduce button to create the template:

<srai>I COULD <star/></srai>

The effect of the new category is to eliminate the word "POSSIBLY"

from all inputs beginning with "I COULD POSSIBLY".

CHAPTER V. PSYCH - Activating Prolog from AIML

This chapter describes a first experiment linking an Artificial Intelligence Markup Language (AIML) script to a Prolog program. Specifically, we used GNU gprolog to implement a program called PSYCH (pronounced "CYC"). The AIML script uses the <system> command to activate the PSYCH commonsense reasoning engine.

Everything described here took place on a Linux (Red Hat 7.1) machine. We have not yet tried running this under Windows, although there is a Windows version of gprolog available.

The goal was to create a set of AIML categories that answer questions using a Prolog program. This experiment implements a simple "isa" hierarchy, so the bot can answer questions like

"Is duck a food?"
"Is rock alive?"
"Is a dog an animal?"
"Is a cat a pet?"

...and so on.

The <system> command

The basic approach was to interface AIML to Prolog through the AIML <system> tag. The Prolog program ran as a separate executable activated by the <system> tag.

It was not too difficult to download and install the GNU gprolog interpreter and compiler from http://pauillac.inria.fr/~diaz/gnu-prolog. There are HTML documentation pages and several directories containing programming examples. The associated Makefiles were a useful guide to writing and compiling simple Prolog programs.

The gprolog suite includes a compiler to create executable prolog programs. The PSYCH script was compiled into an executable called "psych". Normally Prolog runs in an interactive prompting mode, but it is possible to redirect the input from a file or with a pipe.

The basic form of a prolog query is like

isa(dog, animal).

or more generally

predicate(term1,term2,...).

The ending period is important, playing much the same role as the ending ";" in Java or C++. Thus we can query the psych program from the shell with the command:

% echo "isa(dog, animal)." | ./psych

which will print some text including the Prolog reply "Yes" or "No".

The obvious <system> command would be something like

<system>echo "isa(<star index="1"/>,<star index="2"/>)." | ./psych</system>

Unfortunately the implementation of <system> in programs B, D and dB uses the Java Runtime.exec() method. The exec() method has some known problems dealing with Unix pipes ("|") and the echo command.

THE SHELL SCRIPT

To get around the problems with pipes and echo in Java Runtime.exec(), an intermediate shell script was created. This script, called isa.sh, contains the line:

echo "isaset($1,$2)." | ./psych | tail -2 | head -1

The echo command inserts the arguments to isa.sh, $1 and $2, into the Prolog query string isaset($1, $2). For example if we typed the command

isa.sh dog animal

The echo command produces the string "isaset(dog, animal)".

The ./psych command executes the prolog program. The final two commands in the pipeline, tail and head, simply strip off excess output lines generated by Prolog.

THE PROLOG PROGRAM

Another minor stumbling block in this experiment turned out to

be that Prolog does not implement isa-hierarchies very cleanly. Nevertheless, the program PSYCH implements a small isa-hierarchy well enough to answer some basic yes-no questions.

An isa-hierarchy consists of a set of assertions about binary predicates, like:

```
isa(dog, carnivore).
isa(dog, pet).
isa(cat, carnivore).
isa(cat, pet).
isa(carnivore, animal).
```

...and so on.

The complete set of these assertions used in this experiment appears below.

In addition to the assertions, there is a general associative rule something like:

```
isa(X, Y) :-
  isa(X, Z), isa(Z, Y).
```

Through the associative rule the program can reason about "isa" questions and answer even ones that are not explicitly stated. The intuition behind the rule is "X is a Y provided that there exists a Z such that X is Z and Z is Y." Through the associative rule the bot can reason that since a dog is a carnivore, and carnivore is an animal, and an animal is alive, then therefore a dog is alive.

Although the preceding Prolog fragment is syntactically correct, unfortunately it produces an infinite loop. This appears to be a side effect of the order of evaluation in Prolog's built-in backtracking.

The infinite loop is avoided by introducing a new predicate isb(X, Y) defined as follows:

```
isb(X, Y) :- isa(X, Y).
isb(X, Y) :-
  isa(X, Z),
  isb(Z, Y).
```

Finally, the Prolog engine tries to find not just one, but the set of all solutions, for a particular problem. In the interactive console mode, the user would normally step through all solutions with carriage return. In our non-interactive mode, we want only to find out whether the set of solutions is non-empty. For this purpose we use the Prolog built-in predicate setof():

```
isaset(X, Y) :- setof(_, isb(X, Y), _).
```

The query based on the predicate isaset() is called

by the shell script described above. The isaset() predicate returns "Yes" or "No" depending on whether a set of solutions of isb(X, Y) exists.

THE AIML CATEGORIES

The AIML category used to test the Prolog program PSYCH is:

```
<category>
<pattern>IS A * A *</pattern>
<template>
<system>
sh /home/alicebot/isa.sh <star index="1"/> <star index="2"/>
</system>
</template>
</category>
```

Note that rather than just run the script directly, Java Runtime.exec() forced us to prepend a "sh" command to the line. The values of <star index="1"/> and <star index="2"/> become the arguments to isa.sh.

Next, a few <srai> categories map variations of the logical "isa" question to the simple pattern IS A * A *.

```
<category>
<pattern>IS A * AN *</pattern>
<template><srai>IS A <star/> A <star index="2"/></srai></template>
</category>
```

Similar <srai> categories were defined for:

```
<pattern>IS AN * A *</pattern>
<pattern>IS AN * AN *</pattern>
<pattern>IS * A *</pattern>
<pattern>IS * AN *</pattern>
```
and
```
<pattern>IS * *</pattern>
```

SAMPLE DIALOG

The following is a sample dialogue between a client and the chat robot with a Prolog based commonsense reasoning system:

Client: Is Rich alive?
Robot: Yes
Client: Is a boy a mammal?
Robot: Yes
Client: Is a dog a vegetable?
Robot: No

Client: Is a cat a pet?
Robot: Yes
Client: Is a penguin food?
Robot: No
Client: Is a human a mammal?
Robot: Yes

Of course, such a model of reasoning is, by itself, severely limited. One obvious flaw is that the bot will always answer "No" for any out-of-domain questions.

Client: Is a frog an amphibian?
Robot: No
Client: Is a watch a clock?
Robot: No
Client: Is a horse an animal?
Robot: No

This problem could be approached by adding code to the Prolog program to make sure that the elements of the query are also elements of the domain of "isa".

Another problem is that the pattern IS A * A * may match other inputs for which the PSYCH program is totally inappropriate:

Is a dollar a piece a good price?
Is a gram, an ounce, or a pound bigger?
Is A.I. a good thing?

ACTIVATING PROLOG FROM AIML

The experiment described here demonstrates how to activate a simple Prolog program using the <system> tag of AIML. It shows that quite a bit of "commonsense reasoning" can be added to an AIML chat robot robot, without proposing or adding any new AIML features.

The PSYCH program demonstrated here is currently limited to only isa-relations, but more commonsense predicates can be added easily. For example some simple Prolog predicates defining family relationships:

```
parent(X, X) :- fail.
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).

grandmother(X, Y) :-
  mother(X, Z),
  parent(Z, Y).
```

Or we might consider more complex predicates such as

```
between(X, Y, Y) :- fail.
between(X, Y, Z) :- between(X, Z, Y).
```

and more complex queries such as:

"What is a mammal?"
"Give me an example of a carnivore."
"Name some pets."
"Is Pittsburgh between New York and Chicago?"
"My mother is Mary. Her mother is Millie.
Who is my grandmother?"

In the end however it remains unclear whether interfacing A. L. I. C. E. and AIML to a commonsense reasoning engine provides any real practical advantages over a pure AIML pattern targeting approach. Instead of using Prolog to derive answers, over a finite domain of terms, we could just as well explicitly store all the answers in AIML:

```
<category>
<pattern>IS A DOG A MAMMAL</pattern>
<template>Yes.</template>
</category>

<category>
<pattern>IS A DOG A BIRD</pattern>
<template>No.</template>
</category>

<category>
<pattern>IS A DOG ALIVE</pattern>
<template>Yes.</template>
</category>
```

...and so on.

The number of combinations of all terms may produce a large number of combinations, possibly requiring more memory than the AIML interpreter provides. But even these patters have their own Zipf distribution. Some of them are much more likely to be asked than others. In this respect, one can imagine an AIML knowledge base not quite equivalent to the output of a Prolog program, but close enough for all practical purposes.

PROLOG ASSERTIONS

In addition to the associative rules for the isa hierarchy, the PSYCH program includes the following basic commonsense assertions:

```
isa(human, mammal).
isa(man, human).
isa(woman, human).
isa(baby, human).
```

```
isa(boy, human).
isa(girl, human).
isa(rich, man).
isa(carnivore, mammal).
isa(cat, carnivore).
isa(dog, carnivore).
isa(mammal, animal).
isa(animal, alive).
isa(plant, alive).
isa(fruit, plant).
isa(tree, plant).
isa(fir, tree).
isa(maple, tree).
isa(spruce, tree).
isa(birch, tree).
isa(grass, plant).
isa(fruit, food).
isa(weed, plant).
isa(vegetable, plant).
isa(vegetable, food).
isa(alive, entity).
isa(rock, entity).
isa(sand, rock).
isa(alien, alive).
isa(cat, pet).
isa(dog, pet).
isa(chicken, food).
isa(chicken, bird).
isa(bird, animal).
isa(duck, bird).
isa(duck, food).
isa(penguin, bird).
isa(cow, cattle).
isa(bull, cattle).
isa(cattle, herbivore).
isa(cattle, food).
isa(herbivore, animal).
isa(car, vehicle).
isa(plane, vehicle).
isa(jet, plane).
isa(train, vehicle).
isa(bart, train).
isa(muni, train).
isa(tram, train).
isa(bicycle, vehicle).
isa(bus, vehicle).
isa(truck, vehicle).
isa(suv, truck).
isa(rocket, vehicle).
isa(vehicle, manmade).
isa(manmade, entity).
isa(puddle, water).
isa(ocean, water).
isa(lake, water).
isa(river, water).
isa(stream, water).
```

isa(sea, water).
isa(cloud, water).
isa(water, compound).
isa(compound, entity).
isa(ice, water).
isa(steam, water).
isa(aspirin, medicine).
isa(penicillin, medicine).
isa(cancer, disease).
isa(aids, disease).
isa(flu, disease).
isa(glaucoma, disease).

CHAPTER VI. Doing Simple Logical Deductions in AIML

The purpose of this exercise is to demonstrate some simple reasoning capabilities with AIML alone. In particular, we did not use any logic engine such as Prolog. The results described here were obtained solely with AIML.

The types of questions under consideration here are a restricted set of queries exemplified by:

What does a bird have?
What do birds have?
What does a raven do?
What else does a bird have?
What else do ravens do?

Part of what makes it easy to answer these questions in AIML is that the questions tend to have multiple answers. A bird has a beak, a tail, lungs, eyes, wings, feathers, and cold blood. It also has all the properties of any superclasses, such as animals or prokaryotes. The AIML program does not need to find all of these solutions; it merely has to find one.

ISA HIERARCHY

The first step is to construct an Isa hierarchy in AIML. The simplest entries look like this:

```
<category>
<pattern>ISA RAVEN</pattern>
<template>A Bird.</template>
</category>

<category>
<pattern>ISA BIRD</pattern>
```

<template>An animal.</template>
</category>

Basically, the AIML expression is equivalent to the assertion Isa(raven, bird).

But the Isa function is one to many. For example, chicken is both a bird and a food. We use the AIML <random> tag to group these assertions:

<category>
<pattern>ISA CHICKEN</pattern>
<template>
<random>
A Bird.
A Food.
</random>
</template>
</category>

Finally, the AIML set should provide a default case for the Isa relation, in case the first argument can not be identified:

<category>
<pattern>ISA *</pattern>
<template>Unknown</template>
</category>

KNOWLEDGE BASE

The second step was to program the knowledge base with Has and Does relations. It is not necessary to use an abbreviated predicate name in an AIML pattern, so we can actually encode the knowledge base quite conveniently in natural language. So, rather than write "does(bird, fly)" we can express the relation in AIML as:

<category>
<pattern>WHAT DOES A BIRD DO</pattern>
<template>Fly.
</template>
</category>

[The fact that not all birds fly should not distract you from understanding this example.]

As was the case with the Isa relation, the Has and Does relations are many-to-one. We again use the <random> tag to group the many assertions together:

<category>
<pattern>WHAT DOES A BIRD HAVE</pattern>
<template>
<random>
Lungs.

```
<li>An eye.</li>
<li>A beak.</li>
<li>A tail.</li>
<li>A wing.</li>
<li>A feather.</li>
<li>Cold blood.</li>
</random>
</template>
</category>
```

DEFAULTS

Some default replies are also necessary to prevent infinite loops. Here "the Art of AIML writing" comes into play. The ultimate default Isa relation returns "Unknown", so the symbol "UNKNOWN" appears in the patters of these defaults.

```
<category>
<pattern>WHAT DOES UNKNOWN HAVE</pattern>
<template>
<random>
<li>Imagine no possessions.</li>
<li>I don't know</li>
<li>The same as everyone else?</li>
</random>
</template>
</category>
<category>
<pattern>WHAT DOES UNKNOWN DO</pattern>
<template>
<random>
<li>Exist.</li>
<li>I don't know.</li>
<li>The same as everyone else?</li>
</random>
</template>
</category>
```

REDUCTIONS

In this step, we apply symbolic reductions to transform a variety of grammatical forms into simpler inputs.

```
<category>
<pattern>WHAT DO * DO</pattern>
<template><srai>WHAT DOES A <star/> DO</srai>
</template></category>

<category>
<pattern>WHAT DOES A * DO</pattern>
<template><srai>WHAT DOES <star/> DO</srai>
```

</template></category>

These reductions, along with others like them,
will transform many grammatical variants like
"WHAT DOES A X DO", "WHAT DOES AN X DO" and "WHAT DO X DO"
into a single canonical form like "WHAT DOES X DO".

DEDUCTIONS

Provided that the AIML interpreter is capable of
executing nested <srai> tags, it is quite possible
to make simple inferences in AIML. The logic presented
here is that if the client asks "WHAT DOES X DO", and
there is no specific answer for X, then the program
will try to answer the question "WHAT DOES Y DO" where
Isa(X, Y).

The corresponding logical deduction would be:
If X is a Y and Y does Z then X does Z.
The deduction may be written in AIML as:

```
<category>
<pattern>WHAT DOES * DO</pattern>
<template>
<srai>WHAT DOES <srai>ISA <star/></srai> DO</srai>
</template>
</category>
```

The similar category for "WHAT DOES X HAVE" is:

```
<category>
<pattern>WHAT DOES * HAVE</pattern>
<template>
<srai>WHAT DOES <srai>ISA <star/></srai> HAVE</srai>
</template>
</category>
```

Using these deductions, we can now ask the robot
"What does a raven have?" and, even though there is
no explicit answer, the program can infer that, since
a raven is a bird, it must also have anything a bird has:
feathers, beak, tail, eyes, lungs, wings, and cold blood.

PLURALS

A slight modification to the above categories allows
us to handle plurals as well as singular forms, in this
class of questions. If we replace the ISA string with ISB
inside the <srai> tags above, we can define an Isb relation
as a gateway or filter on top of Isa. The Isb function simply
transforms its argument to a singular form, and recursively
applies Isa.

```
<category>
<pattern>ISB *</pattern>
<template><srai>ISA <srai>SINGULAR <star/></srai></srai>
</template></category>
```

The knowledge base for the Singular relation is just like the category sets for Isa, Does, and Has.

```
<category>
<pattern>SINGULAR WOLVES</pattern>
<template>wolf
</template></category>
<category>
<pattern>SINGULAR WOMEN</pattern>
<template>woman
</template></category>
<category>
<pattern>SINGULAR WORKS OF ART</pattern>
<template>work of art
</template></category>
<category>
<pattern>SINGULAR WRENCHES</pattern>
<template>wrench
</template></category>
```

The singular function also requires a default case, when the word or words are not recognized in the function domain:

```
<category>
<pattern>SINGULAR *</pattern>
<template><star/>
</template></category>
```

Notice that, if Singular(X, Y) is not defined, i.e. this last category matches, then Isb(X, Y) = Isa(X, Y).

RANDOM INFERENCES

The categories described in sections 1-5 provide answers to questions like WHAT DOES X HAVE by first checking to see if there is an explicit match for WHAT DOES X HAVE. If not, the program proceed to WHAT DOES Y HAVE where Isa(X, Y). If no match is found there, the program proceeds to WHAT DOES Z HAVE, where Isa(Y, Z). Because there is an ultimate WHAT DOES UNKNOWN HAVE category, the procedure will always terminate.

The inferences will not however find ALL instances of the logical reply to WHAT DOES X HAVE. For example, if we know that all animals have a mother, then one answer to "What does a bird have?" should logically be "a mother".

The following two categories show a crude but simple way to overcome this obstacle. If the client asks "What else

do birds have?", the random function will select between two possible alternative questions: "What do birds have?" and, for some X such that Isa(bird, X), "What else do X have?".

If the program chooses the second form, the question itself becomes the input to this same category.

```
<category>
<pattern>WHAT ELSE DO * DO</pattern>
<template><random>
<li><srai>WHAT DO <star/> DO</srai></li>
<li><srai>WHAT ELSE DO <srai>ISB <star/></srai> DO</srai></li>
</random>
</template></category>
<category>
<pattern>WHAT ELSE DO * HAVE</pattern>
<template><random>
<li><srai>WHAT DO <star/> HAVE</srai></li>
<li><srai>WHAT ELSE DO <srai>ISB <star/></srai> HAVE</srai></li>
</random>
</template></category>
```

SAMPLE DIALOG

The following dialog demonstrates the categories described in this technical note.

Client: What does a bird have?
Robot: Feathers.
Client: What does a bird have?
Robot: Lungs.
Client: What does a bird have?
Robot: A beak.
Client: What do birds have?
Robot: A beak.
Client: What does a raven do?
Robot: Fly.
Client: What else does a bird have?
Robot: A tail.
Client: What else does a bird have?
Robot: Nuclear membranes.
Client: What else does a bird have?
Robot: A body.
Client: What else does a bird have?
Robot: Imagine no possessions.
Client: What else does a chicken do?
Robot: Get eaten.
Client: What else does a chicken do?
Robot: Cluck.
Client: What else do chickens do?
Robot: Peck.

A bird has a nuclear membrane because it is a prokaryote. All animals have a body, so birds have a body too. Chicken is both

a food and a bird, so it can both cluck and be eaten. Ravens can fly because the robot believes all birds can fly.

LOGICAL DEDUCTIONS IN AIML

We have shown how to make certain simple logical deductions in AIML alone, without the need for an external Prolog or other reasoning program.

The ALICE brain set was augmented with more than 1000 Isa relations, and several hundred categories encoding Has and Does. These categories will be made public in the next release of the ALICE Brain AIML set.

CHAPTER VII. A Personal Finance "Spreadsheet" in AIML

One of the critical requirements for the adoption of AIML technology as a standardized human-computer interface is the creation of a core set of application programming interfaces. For ALICE to become the primary mode of user interaction, she must have interfaces to a collection of common applications such as email, database, address book, calculator, and spreadsheet programs.

This chapter describes a first pass at the "spreadsheet" application for managing personal finances.

SAMPLE DIALOG

The goal of this experiment is to produce a believable natural language interface to a trivial personal finance system. The following sample dialogue illustrates the flavor of conversations the bot should be able to handle:

Client: My checking account balance is 2300.
Robot: I got it. Do you have any more transactions?
Client: My savings account balance is 1000.
Robot: I got it. Anything else I can do for you?
Client: My visa card balance is 2100.
Robot: I got it. Thanks for the information.
Client: My Citibank balance is 9000.
Robot: I got it. Is that it?
Client: I have 200 dollars in my wallet.
Robot: Alright. Thanks for the information.
Client: I spent 10 on a taxi.
Robot: I got it. Anything else I can do for you?
Client: I charged 23 for lunch on my visa card.
Robot: I got it. Thanks for the information.
Client: My landlord is Mr. bill.
Robot: Rent is theft!
Client: No kidding my rent is 1410 a month.

Robot: Would I kid you? 1410 is the market rate.
Client: I paid the rent with check number 101.
Robot: I got it. Thanks for the information.
Client: what is my checking account balance.
Robot: Your checking balance is 890.
Client: I charged 120 on my Visa card for the hotel.
Robot: Ok. Anything else I can do for you?
Client: What is my Visa card balance.
Robot: Your visa balance is -2243.
Client: How much money do I have in my wallet?
Robot: Your wallet balance is 190.
Client: I bought a book for 10.
Robot: Yes. I have recorded your transaction.
Client: now how much is in my wallet?
Robot: Why now? Your wallet balance is 180.

THE TRANSACTIONS

The preceding dialogue reduces to the transactions listed in this section.

The DEBIT and CREDIT transactions have exactly the same format, except that the DEBIT transaction has a payee where the CREDIT has a payer. In either case the program stores the transactions in a comma-delimited format, making for easy import into other financial applications.

Specifically, the transaction record format is a comma delimited string of the form:
DEBIT account, amount, date, number, payee, category, note, id.
or
CREDIT account, amount, date, number, payer, category, note, id.

The account field identifies the account as checking, savings, wallet or the name of another account. The amount is indicated in dollars, as a nonnegative integer. The date includes a time stamp to the resolution of seconds. The number field usually indicates a check number or transaction number, if any. Payer/payee indicates the source/destination accounts for the transaction. The category field helps create a budget report later, showing how our income and expenses break down by categories such as rent, medicine, food, utility bills and so on.

The "security" of the system is limited to recording the client id, in this case localhost.localdomain. If the bot is online, care should be taken so that unauthorized clients cannot initiate personal financial transactions.

The records do not include a unique identifier or row number. Uniqueness of the records is only guaranteed by the timestamp, using the AIML <date/> function. This is fine if the transaction rate is always less than one per second. But a more realistic application may also have to include record identifiers.

The most sophisticated language processing demonstrated in this

experiment was the use of AIML predicates for "landlord" and "rent".
This makes possible the transformation of sentences such as
"I paid the rent with check number 101" into a transaction.

INITIALIZE BANK checking, 2300, Tue Dec 25 08:32:16 PST 2001,
localhost.localdomain.
CREDIT checking, 2300, tue dec 25 08 32 16 pst 2001, none, unknown, unknown,
none, localhost.localdomain.
INITIALIZE BANK savings, 1000, Tue Dec 25 08:32:24 PST 2001,
localhost.localdomain.
CREDIT savings, 1000, tue dec 25 08 32 24 pst 2001, none, unknown, unknown,
none, localhost.localdomain.
INITIALIZE CREDT visa card, 2100, Tue Dec 25 08:32:32 PST 2001,
localhost.localdomain.
DEBIT visa card, 2100, tue dec 25 08 32 32 pst 2001, none, unknown, unknown,
none, localhost.localdomain.
INITIALIZE CREDT Citibank, 9000, Tue Dec 25 08:32:51 PST 2001,
localhost.localdomain.
DEBIT Citibank, 9000, tue dec 25 08 32 51 pst 2001, none, unknown, unknown,
none, localhost.localdomain.
INITIALIZE BANK wallet, 200, Tue Dec 25 08:33:01 PST 2001,
localhost.localdomain.
CREDIT wallet, 200, tue dec 25 08 33 01 pst 2001, none, unknown, unknown,
none, localhost.localdomain.
DEBIT wallet, 10, tue dec 25 08 33 23 pst 2001, none, a taxi, unknown, none,
localhost.localdomain.
DEBIT visa, 23 for lunch, tue dec 25 08 33 34 pst 2001, none, unknown,
unknown, none, localhost.localdomain.
DEBIT checking, 1410, tue dec 25 08 34 07 pst 2001, number 101, Mr. bill,
rent, none, localhost.localdomain.
DEBIT visa, 120, tue dec 25 08 35 00 pst 2001, none, unknown, the hotel,
none, localhost.localdomain.
DEBIT wallet, 10, tue dec 25 08 35 28 pst 2001, none, unknown, a book, none,
localhost.localdomain.

THE AIML CATEGORIES

The dialogue in section 2 resulted from the contents of the ALICE
brain, plus the following set of "spreadsheet" categories.
The basic idea is to reduce every financial statement into
a transaction of the form CREDIT, DEBIT or INITIALIZE. There
are two types of accounts, CREDT accounts, mainly for credit cards,
and BANK accounts, which can include wallets, piggybanks, safes
and other money containers, as well as regular bank debit
accounts like checking and savings.

The purpose of the INITIAIZE statement is to set an initial
balance, open a new account, or override the spreadsheet's
calculated balance for an account. CREDT accounts are opened
with a negative balance, initiated with a DEBIT transaction.
BANK accounts are opened with a positive balance, initiated
using a CREDIT transaction.

All of the DEBIT and CREDIT transactions are recorded in the default gossip file. This may not be the best place, but it made for a simpler pedagogical example. The program balance.sh calculates the account balances as need (see next section).

The following AIML categories represent a first-pass attempt to create the simple "spreadsheet". Not a true double-entry bookkeeping system, the categories nonetheless demonstrate the feasibility of building a personal finance application in AIML.

```
<aiml>
<category><pattern>ACKNOWLEDGE TRANSACTION</pattern>
<template>
<random>
<li>Ok.</li>
<li>Yes.</li>
<li>Alright.</li>
<li>I got it.</li>
</random>
<random>
<li>Is there anything else I can help you with?</li>
<li>Do you have any more transactions?</li>
<li>I have recorded your transaction.</li>
<li>Anything else I can do for you?</li>
<li>Thanks for the information.</li>
<li>Is that it?</li>
</random>
</template></category>
<category>
<pattern>CREDIT ACCT * AMT * DATE * NUMBER * PAYER * CATEGORY * NOTE
*</pattern>
<template>
<srai>ACKNOWLEDGE TRANSACTION</srai>
<think>
<gossip>CREDIT <star/>, <star index="2"/>, <star index="3"/>, <star
index="4"/>, <star index="5"/>, <star index="6"/>, <star index="7"/>, <id/>.
</gossip>
</think>
</template></category>
<category>
<pattern>DEBIT ACCT * AMT * DATE * NUMBER * PAYEE * CATEGORY * NOTE
*</pattern><template>
<srai>ACKNOWLEDGE TRANSACTION</srai>
<think>
<gossip>DEBIT <star/>, <star index="2"/>, <star index="3"/>, <star
index="4"/>, <star index="5"/>, <star index="6"/>, <star index="7"/>, <id/>.
</gossip>
</think>
</template></category>
<category><pattern>INITIALIZE BANK ACCT * AMT *</pattern><template>
<srai>ACKNOWLEDGE TRANSACTION</srai>
<think>
<gossip>INITIALIZE BANK <star/>, <star index="2"/>, <date/>, <id/>.</gossip>
<srai>CREDIT ACCT <star/> AMT <star index="2"/> DATE <date/></srai>
</think>
</template></category>
```

```
<category><pattern>INITIALIZE CREDT ACCT * AMT *</pattern><template>
<srai>ACKNOWLEDGE TRANSACTION</srai>
<think>
<gossip>INITIALIZE CREDT <star/>, <star index="2"/>, <date/>, <id/>.</gossip>
<srai>DEBIT ACCT <star/> AMT <star index="2"/> DATE <date/></srai>
</think>
</template></category>
<category><pattern>BALANCE ACCT *</pattern><template><system>sh
/home/alicebot/balance.sh <star/></system></template></category>
<category><pattern>HOW MUCH IS IN MY WALLET</pattern><template><srai>BALANCE
ACCT WALLET</srai></template></category>
<category><pattern>HOW MUCH MONEY _ MY
WALLET</pattern><template><srai>BALANCE ACCT
WALLET</srai></template></category>
<category><pattern>HOW MUCH CASH _ I HAVE</pattern><template><srai>BALANCE
ACCT WALLET</srai></template></category>
<category><pattern>WHO IS MY LANDLORD</pattern><template><get
name="landlord"/></template></category>
<category><pattern>WHAT IS MY RENT</pattern><template><get name="rent"/> per
month.</template></category>
<category><pattern>MY RENT IS *</pattern><template><set
name="rent"><star/></set> is the market rate.</template></category>
<category><pattern>MY RENT IS * DOLLARS PER MONTH</pattern><template><srai>MY
RENT IS <star/></srai></template></category>
<category><pattern>MY RENT IS * PER MONTH</pattern><template><srai>MY RENT IS
<star/></srai></template></category>
<category><pattern>MY LANDLORD IS *</pattern><template> <think><set
name="landlord"><person/></set></think> Rent is theft!</template></category>
<category><pattern>MY CHECKING ACCOUNT BALANCE IS
*</pattern><template><srai>INITIALIZE BANK ACCT checking AMT
<star/></srai></template></category>
<category><pattern>MY CITIBANK BALANCE IS
*</pattern><template><srai>INITIALIZE CREDT ACCT Citibank AMT <star/></srai>
</template></category>
<category><pattern>MY SAVINGS ACCOUNT BALANCE IS
*</pattern><template><srai>INITIALIZE BANK ACCT savings AMT
<star/></srai></template></category>
<category><pattern>MY VISA CARD BALANCE IS
*</pattern><template><srai>INITIALIZE CREDT ACCT visa card AMT
<star/></srai></template></category>
<category><pattern>I HAVE * IN MY WALLET</pattern><template><srai>INITIALIZE
BANK ACCT wallet AMT <star/></srai></template></category>
<category><pattern>I HAVE * DOLLARS IN MY
WALLET</pattern><template><srai>INITIALIZE BANK ACCT wallet AMT
<star/></srai></template></category>
<category><pattern>WHAT IS MY * ACCOUNT
BALANCE</pattern><template><srai>BALANCE ACCT
<star/></srai></template></category>
<category><pattern>I HAVE * DOLLARS TO MY
NAME</pattern><template><srai>INITIALIZE BANK ACCT wallet AMT
<star/></srai></template></category>
<category><pattern>I HAVE * DOLLARS</pattern><template><srai>INITIALIZE BANK
ACCT wallet AMT <star/></srai></template></category>
<category><pattern>WHAT IS MY * CARD BALANCE</pattern><template><srai>BALANCE
ACCT <star/></srai></template></category>
<category><pattern>DEPOSIT * IN CHECKING
```

```
ACCOUNT</pattern><template><srai>CREDIT ACCT checking AMT
<star/></srai></template></category>
<category><pattern>I PAID MY RENT WITH CHECK *</pattern><template><srai>DEBIT
ACCT checking AMT <get name="rent"/> DATE <date/> NUMBER <star/> PAYEE <get
name="landlord"/> CATEGORY RENT</srai></template></category>
<category><pattern>I PAID THE RENT WITH CHECK
*</pattern><template><srai>DEBIT ACCT checking AMT <get name="rent"/> DATE
<date/> NUMBER <star/> PAYEE <get name="landlord"/> CATEGORY
RENT</srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
</aiml>
```

The remainder of the AIML categories have been placed in an appendix due to considerations of length.

THE SHELL SCRIPT

There are an infinite number programs one could write to calculate account balances based on the DEBIT and CREDIT transaction recorded in the gossip file.

Ideally the AIML script should be tightly integrated with a real spreadsheet application. For now we use a simple shell script to stand in for the real application. The script uses awk to calculate sums. Its use of temporary files is unfortunate.

```
# the account balance shell script
# This example is for instructional purposes only.
# Usage: balance <account>
rm -f *.temp
echo 0 > credit.temp
echo 0 > debit.temp
echo "Your "$@" balance is"
grep DEBIT /home/alicebot/data/gossip.data | grep $@ | awk -F , '{print $2;
n-=$2; print n}' | tail -1 > debit.temp
grep CREDIT /home/alicebot/data/gossip.data | grep $@ | awk -F , '{print $2;
n+=$2; print n}' | tail -1 > credit.temp
cat credit.temp debit.temp | awk '{print $1; n+=$1; print n}' | tail -1
```

AN AIML SPREADSHEET

We have demonstrated the feasibility of a simple personal finance application with AIML. Despite its limitations, the program is adequate for a small set of personal finance needs.

To make this application more robust we should:

a). Create a realistic double-entry bookkeeping system.

b). Add a unique identifier to each transaction.

c). Integrate the program more closely with a real financial application.

d). Develop an expanded set of AIML patterns to capture a wider variety of transaction statements, including dollars and cents, international currencies, interest payments, and taxes.

e). Enhance the security feature to prohibit unauthorized clients from making personal financial transactions.

f). Substitute voice for text input. This is the kind of application you want to carry around with you, so you can utter "I spent 10 on a taxi", at the same time you pay the driver, and "I took out 100 from the ATM", while you are still at the cash machine.

More generally, we need to work on AIML APIs for a variety of everyday applications. These include, but are not limited to, address books, spreadsheets, databases, email, web browsers, word processors, and games. In the dream world of the future, the talking Star Trek/HAL-style computer will handle all of the applications we use now on WIMP (Windows, Icons, Menus, Pointing Device) computers.

Appendix A. AIML categories for personal finance

Note: For the personal finance application, these categories should be included
along with those listed in section 4. Most of the categories have been moved to an appendix so that this document is not too long.

```
<aiml>
<category><pattern>DEPOSIT * IN CHECKING</pattern><template><srai>CREDIT ACCT
checking AMT <star/></srai></template></category>
<category><pattern>DEPOSIT * IN MY CHECKING
ACCOUNT</pattern><template><srai>CREDIT ACCT checking AMT
<star/></srai></template></category>
<category><pattern>DEPOSIT * TO CHECKING
ACCOUNT</pattern><template><srai>CREDIT ACCT checking AMT
<star/></srai></template></category>
<category><pattern>DEPOSIT * TO CHECKING</pattern><template><srai>CREDIT ACCT
checking AMT <star/></srai></template></category>
<category><pattern>DEPOSIT * IN MY SAVINGS
ACCOUNT</pattern><template><srai>CREDIT ACCT savings AMT
<star/></srai></template></category>
<category><pattern>DEPOSIT * IN SAVINGS
ACCOUNT</pattern><template><srai>CREDIT ACCT savings AMT
<star/></srai></template></category>
<category><pattern>DEPOSIT * IN SAVINGS</pattern><template><srai>CREDIT ACCT
savings AMT <star/></srai></template></category>
<category><pattern>DEPOSIT * TO SAVINGS
ACCOUNT</pattern><template><srai>CREDIT ACCT savings AMT
<star/></srai></template></category>
<category><pattern>DEPOSIT * TO SAVINGS</pattern><template><srai>CREDIT ACCT
savings AMT <star/></srai></template></category>
```

<category><pattern>CREDIT ACCT * AMT *</pattern><template><srai>CREDIT ACCT
<star/> AMT <star index="2"/> DATE <date/></srai></template></category>
<category><pattern>CREDIT ACCT * AMT * DATE *</pattern><template><srai>CREDIT
ACCT <star/> AMT <star index="2"/> DATE <star index="3"/> NUMBER
NONE</srai></template></category>
<category><pattern>CREDIT ACCT * AMT * DATE * NUMBER * PAYER * CATEGORY
*</pattern><template><srai>CREDIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYER <star index="5"/> CATEGORY
<star index="6"/> NOTE NONE</srai></template></category>
<category><pattern>CREDIT ACCT * AMT * DATE * NUMBER * PAYER
*</pattern><template><srai>CREDIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYER <star index="5"/> CATEGORY
UNKNOWN</srai></template></category>
<category><pattern>CREDIT ACCT * AMT * DATE * NUMBER
*</pattern><template><srai>CREDIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYER
UNKNOWN</srai></template></category>
<category><pattern>I DEPOSITED * IN *</pattern><template><srai>CREDIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>I DEPOSITED * INTO *</pattern><template><srai>CREDIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>I DEPOSITED * TO MY *</pattern><template><srai>CREDIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>I DEPOSITED * TO *</pattern><template><srai>CREDIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>_ LOANED ME * DOLLARS</pattern><template><srai>CREDIT ACCT
wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYER <person/>
CATEGORY loan NOTE repay <person/> </srai></template></category>
<category><pattern>_ LOANED ME *</pattern><template><srai>CREDIT ACCT wallet
AMT <star index="2"/> DATE <date/> NUMBER none PAYER <person/> CATEGORY loan
NOTE repay <person/> </srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM AT * FOR

*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM AT *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM AT *</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD AT *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>

NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM AT *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM FOR * AT
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD FOR

*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM FOR *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE unknown CATEGORY
<star index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM FOR *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE unknown CATEGORY
<star index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE unknown CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM FOR *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE unknown CATEGORY
<star index="2"/></srai></template></category>
<category><pattern>I PAID * WITH CHECK *</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/> NUMBER <star index="2"/> PAYEE
UNKNOWN</srai></template></category>
<category><pattern>I PAID * DOLLARS TO * WITH CHECK NUMBER
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER <star index="3"/> PAYEE <star index="2"/></srai></template></category>
<category><pattern>I WROTE A CHECK FOR * TO *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER UNKNOWN PAYEE <star
index="2"/></srai> </template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM
CARD</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY
ATM</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM

CARD</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I PUT * ON MY ATM</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM
CARD</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I PAID * * DOLLARS WITH CHECK NUMBER
*</pattern><template><srai>DEBIT ACCT checking AMT <star index="2"/> DATE
<date/> NUMBER <star index="3"/> PAYEE <star/></srai></template></category>
<category><pattern>I WROTE CHECK NUMBER * FOR * TO
*</pattern><template><srai>DEBIT ACCT checking AMT <star index="2"/> DATE
<date/> NUMBER <star/> PAYEE <star index="3"/></srai> </template></category>
<category><pattern>I WROTE A CHECK TO * FOR *</pattern><template><srai>DEBIT
ACCT checking AMT <star index="2"/> DATE <date/> NUMBER UNKNOWN PAYEE
<star/></srai> </template></category>
<category><pattern>I WROTE CHECK NUMBER * TO * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star index="3"/> DATE
<date/> NUMBER <star/> PAYEE <star index="2"/></srai> </template></category>
<category><pattern>WITHDRAW * FROM CHECKING
ACCOUNT</pattern><template><srai>DEBIT ACCT checking AMT
<star/></srai></template></category>
<category><pattern>WITHDRAW * FROM CHECKING</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai></template></category>
<category><pattern>WITHDRAW * FROM MY CHECKING
ACCOUNT</pattern><template><srai>DEBIT ACCT checking AMT
<star/></srai></template></category>
<category><pattern>I GOT * FROM THE ATM</pattern><template><srai>DEBIT ACCT
checking AMT <star/></srai> </template></category>
<category><pattern>I WITHDREW * FROM THE ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai> </template></category>
<category><pattern>I TOOK OUT * FROM THE ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai> </template></category>
<category><pattern>I TOOK OUT * WITH THE ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai> </template></category>
<category><pattern>I GOT OUT * WITH THE ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai> </template></category>
<category><pattern>WITHDRAW * FROM MY SAVINGS
ACCOUNT</pattern><template><srai>DEBIT ACCT savings AMT
<star/></srai></template></category>
<category><pattern>WITHDRAW * FROM SAVINGS
ACCOUNT</pattern><template><srai>DEBIT ACCT savings AMT
<star/></srai></template></category>
<category><pattern>WITHDRAW * FROM SAVINGS</pattern><template><srai>DEBIT
ACCT savings AMT <star/></srai></template></category>
<category><pattern>I WITHDREW * FROM SAVINGS</pattern><template><srai>DEBIT

ACCT savings AMT <star/></srai> </template></category>
<category><pattern>DEBIT ACCT * AMT *</pattern><template><srai>DEBIT ACCT
<star/> AMT <star index="2"/> DATE <date/></srai></template></category>
<category><pattern>DEBIT ACCT * AMT * DATE *</pattern><template><srai>DEBIT
ACCT <star/> AMT <star index="2"/> DATE <star index="3"/> NUMBER
NONE</srai></template></category>
<category><pattern>DEBIT ACCT * AMT * DATE * NUMBER * PAYEE * CATEGORY
*</pattern><template><srai>DEBIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYEE <star index="5"/> CATEGORY
<star index="6"/> NOTE NONE</srai></template></category>
<category><pattern>DEBIT ACCT * AMT * DATE * NUMBER * PAYEE
*</pattern><template><srai>DEBIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYEE <star index="5"/> CATEGORY
UNKNOWN</srai></template></category>
<category><pattern>DEBIT ACCT * AMT * DATE * NUMBER
*</pattern><template><srai>DEBIT ACCT <star/> AMT <star index="2"/> DATE
<star index="3"/> NUMBER <star index="4"/> PAYEE
UNKNOWN</srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD AT * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD AT *</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE

<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD AT *</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD FOR * AT
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE unknown CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE unknown CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD FOR *</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/> NUMBER none PAYEE unknown
CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE unknown CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE unknown CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE unknown CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY *
CARD</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/></srai></template></category>

<category><pattern>I PUT * DOLLARS ON MY *
CARD</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD</pattern><template><srai>DEBIT ACCT
<star index="2"/> AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY *
CARD</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD</pattern><template><srai>DEBIT ACCT
<star index="2"/> AMT <star/> DATE <date/></srai></template></category>
<category><pattern>I WITHDREW * FROM MY *</pattern><template><srai>DEBIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>I WITHDREW * FROM *</pattern><template><srai>DEBIT ACCT
<star index="2"/> AMT <star/> </srai></template></category>
<category><pattern>I PAID * DOLLARS FOR * AT *</pattern><template><srai>DEBIT
ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/>
CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I SPENT * AT * FOR *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY
<star index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLA RS AT * ON *</pattern><template><srai>DEBIT
ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/>
CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PAID * DOLLARS FOR *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I PAID * FOR *</pattern><template><srai>DEBIT ACCT wallet
AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * AT *</pattern><template><srai>DEBIT ACCT wallet
AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS AT *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON * AT *</pattern><template><srai>DEBIT
ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON *</pattern><template><srai>DEBIT ACCT wallet
AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I PAID * FOR * AT *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="3"/> CATEGORY
<star index="2"/></srai></template></category>
<category><pattern>I SPENT * ON * AT *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="3"/> CATEGORY
<star index="2"/></srai></template></category>
<category><pattern>I BOUGHT * FOR * AT *</pattern><template><srai>DEBIT ACCT
wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYEE <star index="2"/>
CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I BOUGHT * FOR * DOLLARS AT
*</pattern><template><srai>DEBIT ACCT wallet AMT <star index="2"/> DATE
<date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star

index="3"/></srai></template></category>
<category><pattern>I BOUGHT * FOR * DOLLARS</pattern><template><srai>DEBIT ACCT wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYEE unknown CATEGORY <star/></srai></template></category>
<category><pattern>I BOUGHT * FOR *</pattern><template><srai>DEBIT ACCT wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYEE unknown CATEGORY <star/></srai></template></category>
<category><pattern>MY RENT IS * A MONTH</pattern><template><srai>MY RENT IS <star/></srai></template></category>
<category><pattern>MY RENT IS * DOLLARS A MONTH</pattern><template><srai>MY RENT IS <star/></srai></template></category>
<category><pattern>MY RENT IS * DOLLARS</pattern><template><srai>MY RENT IS <star/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM FOR * ON *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM FOR * ON *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM ON * FOR *</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star

index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM ON * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM ON *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM ON *</pattern><template><srai>DEBIT ACCT
checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD ON *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM ON *</pattern><template><srai>DEBIT
ACCT checking AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY ATM FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star

```
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I CHARGED * ON MY ATM FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I PUT * ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY ATM FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I SPENT * ON MY ATM FOR * ON
*</pattern><template><srai>DEBIT ACCT checking AMT <star/> DATE <date/>
NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="2"/></srai></template></category>
<category><pattern>I TOOK OUT * AT THE ATM</pattern><template><srai>DEBIT
ACCT checking AMT <star/></srai> </template></category>
<category><pattern>I GOT OUT * AT THE ATM</pattern><template><srai>DEBIT ACCT
checking AMT <star/></srai> </template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
```

index="4"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD ON * FOR
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star
index="4"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD ON *</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD ON *</pattern><template><srai>DEBIT
ACCT <star index="2"/> AMT <star/> DATE <date/> NUMBER none PAYEE <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * DOLLARS ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I CHARGED * ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * DOLLARS ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PUT * ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I SPENT * ON MY * CARD FOR * ON
*</pattern><template><srai>DEBIT ACCT <star index="2"/> AMT <star/> DATE
<date/> NUMBER none PAYEE <star index="4"/> CATEGORY <star
index="3"/></srai></template></category>
<category><pattern>I PAID * DOLLARS FOR * ON *</pattern><template><srai>DEBIT
ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/>
CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I SPENT * ON * FOR *</pattern><template><srai>DEBIT ACCT
wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY
<star index="3"/></srai></template></category>
<category><pattern>I SPENT * DOLLARS ON * ON *</pattern><template><srai>DEBIT
ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="2"/>
CATEGORY <star index="3"/></srai></template></category>

<category><pattern>I PAID * FOR * ON *</pattern><template><srai>DEBIT ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star index="2"/></srai></template></category>
<category><pattern>I SPENT * ON * ON *</pattern><template><srai>DEBIT ACCT wallet AMT <star/> DATE <date/> NUMBER none PAYEE <star index="3"/> CATEGORY <star index="2"/></srai></template></category>
<category><pattern>I BOUGHT * FOR * ON *</pattern><template><srai>DEBIT ACCT wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
<category><pattern>I BOUGHT * FOR * DOLLARS ON *</pattern><template><srai>DEBIT ACCT wallet AMT <star index="2"/> DATE <date/> NUMBER none PAYEE <star index="2"/> CATEGORY <star index="3"/></srai></template></category>
</aiml>

CHAPTER VIII. Building a Subscriber Based Bot Business with Pandorabots

If I was making the film "The Graduate", with Dustin Hoffmann, today, instead of advising him to go into plastics, I would say "bots." I would say forget about graduate school and academia, that is a dead end. Go to work for yourself if you can, and join the A. L. I. C. E. and AIML free software community. You will have the benefit of a huge free software development effort and community, free, and be able to devote most of your efforts to marketing, promotions, advertising and sales. Seriously, if I was a young, healthy person like yourself, I would go for it too.

Here is a list of the top 10 "killer apps" for AIML I put together:

Ten Killer Apps for A. L. I. C. E. and AIML

1. ESL Bot
ESL sites are the #1 source of traffic to Alicebot.org. ESL students need a lot of practice chatting in English, and the bot provides a very safe environment to practice.

2. FAQbot
Any FAQ can be converted to AIML easily. Any organization or individual answering a set of frequently asked questions can reduce their workload with a bot.

3. Toy
Combined with speech recognition, creates talking teddy bear or other toy. Falling prices of embedded micro controllers and DSPs make A.I. toys practical.

4. Celebrity Bot
Fans of Britney Spears can go to her fan site and chat with Britney Bot. ELVIS and John Lennon have already been done, and there is likely to be

a "land rush" on deceased celebrity names: Timothy Leary, Andy Warhol, Ronald Reagan, JFK, Frank Zappa, Richard Nixon, Malcom X, Martin Luther King, Jr., John Travolta, O.J. Simpson, Albert Einstein, Marilyn Monroe, Picasso, Janis Joplin, and finitely many more.

5. Personal Assistant
Voice-based personal secretary handles finances, address book, database and email. All of the most common computer applications will have natural language voice based interfaces.

6. IM Plugin
Instant Messaging users can set their chat connection to "automatic" or "manual."

7. Targeting Tool
Programs to assist the botmaster analyze the log files and add bot content.

8. Bot Hosting
Many people cannot afford 24/7 Web connections but would like to have their bots always online.

9. Social Security Bot
Many disabled and retiring people have basic questions about benefits, applications, eligibility, and other aspects of the Social Security process.

10. Go Ask ALICE: Harm Reduction Bot
Help reduce the harmful effects of hard drug use by providing truthful information to drug users.

PayPal allows businesses and consumers with e-mail addresses to send and receive payments via the Internet, accepting credit card or bank account payments for purchases. If you have an email account, you can create an account on PayPal with the same name. For example, if your email name is artjohnson@xyz.ww, then your PayPal business name would also be artjohnson@xyz.ww. You can then invite people to send you money from their PayPal accounts, or send money to their accounts.

Go to www.PayPal.com and click on the link that says, "Sign up for your free PayPal account." You have to fill out a one page form of information. You only need your email address to create an account. Later, you can add your checking account and/or credit cards. The PayPal system will email you a verification letter. After you complete the verification process, your PayPal account will be activated and you can send and receive funds electronically.

Pandorabots (http://www.pandorabots.com/) is a software robot (also known as a bot) hosting service. From any browser, you may create and publish your own robots to anyone via the web. We believe that our technology yields the fastest bots available on the Internet. The bots are based on AIML and spring entirely from the work of Dr. Richard Wallace and the A. L. I. C. E. and AIML free software community based at http://www.alicebot.org/. There is no charge for using this site. We gladly accept any inquiries for custom versions of this site for

particular applications. Send mail to info@pandorabots.com) with questions. No one has yet discovered the true potential of virtual personalities and bots, and our mission is to design tools which will help Bots achieve greater relevance in both the commercial and personal sectors. Capabilities such as voice recognition, text to speech synthesis, invoking programs on remote or local machines, content development tools, e-commerce and cartoon animations are under development to help you bring your bot to virtual life. You should expect numerous changes to this site, so please save your bot creations on your own machine using the "file save" facility of your browser. (Your bot's knowledge is typically stored in a file called update.aiml and is available via the "edit" link, which is itself available from the "Botmaster control" link.) Rudimentary documentation now exists - see the "Help" link and will undergo changes as the site changes. The Bots hosted here conform in most respects to the bot specifications given at http://www.alicebot.org/.

The Subscriber Service allows you to receive money from customers interacting with your robot at www.pandorabots.com. Here's now it works: Using the hosting service www.pandorabots.com you create and publish a robot, and you don't reveal the published location to anyone. You recruit and persuade customers to sign up to interact with your robot. You decide who talks to your robot, the amount to charge and how to collect the money. Click the Subscriber services button and follow the directions to allow access to each customer. Pandorabots will be adding more services in the near future. At the time of this writing there is no charge for these additional services. We expect future services to be compatible with this service yet we can make no guarantees. Also, as this hosting site is free of any charges please be aware that we make no guarantees or warranties of any type.

How can I Sign Up Customers? Recruiting and Persuading Customers.... Use your imagination to recruit new customers. To get started consider these ideas: (1) e-mail to promote your site, (2) advertise your robot from a (free) demonstration robot supplying free-sample interactions, (3) advertise on other botmaster's robot pages, (4) paying other botmasters to advertise your robots, (5) or use other botmasters robots to advertise your pages. Advertisements can be placed on pages of particularly popular robots. Embed advertising messages into your robots and sell the service to other companies and botmasters. Use your imagination!

When a customer makes a purchase
with PayPal, either by clicking a button on a web page or following a link in an e-mail, the PayPal system will alert you, the seller, by sending you an e-mail. You can also go to the PayPal site and log into your account to check your daily activity. Any new orders appear in the list of daily transactions in your account. Once you link your checking account to PayPal, you will have the option to withdraw funds and transfer them directly to your checking account. This process may take a few days, depending on your bank. PayPal has several other options for withdrawing funds, check their website.

On Pandorabots, go to the Botmaster Control page. In the column marked Subscribers, click on the Subscribers link. On the Subscribers page, you will see a form with two text fields to Add a New

Subscriber. Enter the email address and the number of months this customer will subscribe. After you click the "Add subscriber" button, Pandorabots will update the table to display a new row corresponding to the new customer's bot link. Under the column marked "Access URL", click on "Subscriber's bot" and follow the link. The URL that appears in the browser is the customer's private subscription link. This is the link that you will email to the customer, and tell them not to share with anyone else.

After you have collected the payment and created the subscriber's bot, it is time to send them their private link. Cut and paste the URL from the browser (see previous) step. Send the customer the link in an email message. It is a good idea to have a form letter ready (like this one) explaining to the customer that they should bookmark the link and keep it private. If they lose it, they can contact you and you can resupply it.

What about Refunds and Deactivation? In any business some customers are bound to come back dissatisfied with their purchases. You should be prepared to handle requests for refunds, or have a strict no-refunds policy. (In the case of the ALICE A.I. Foundation, subscriptions to the A. L. I. C. E. Silver Edition are considered a donation to a nonprofit charity and refunds are not given, no exceptions). You can use PayPal to refund the customer his or her money, but they should remember that PayPal collects a fee for every transaction. As part of the cost of doing business, a full refund cannot be expected. The buyer should beware that even a money back guarantee may not include all the fees required to process both the purchase and the refund through an e-payment system like PayPal. On Pandorabots, the Subscriber page has a feature to deactivate subscribers for any reason. In the first column marked "State" locate the link called "Click to deactivate" and follow the directions to remove a subscriber from your bot subscription list.

The easiest way to receive money from customers is through the service www.PayPal.com. Other ways include checks, cash, and credit cards. You can also use wire transfer services such as Western Union or postal money orders.
Receiving money Credit Cards is difficult unless you have a merchant credit card account - and so we recommend you seek payment using PayPal. Of course, you can also provide free access to your robot...

CHAPTER IX. The Future of AIML

I began working on a new bot named Interviewer that is designed to automate some of the botmastering process via the natural language interface:

Human: start
Bot: what is your name
Human: rich
Bot: Nice to meet you, rich! Are you a man or a woman?
Human: male
Bot: Your gender is recorded as male. What is your birth date?

Human: 1960
Bot: Being born on 1960 is very lucky. Who is your mother?
Human: minty
Bot: Thank you for the information. Who is your father?
Human: stan
Bot: Great. So far we have Name: rich. Gender: male. Birth date: 1960. Mother: minty. Father: stan. Do you have any corrections now?

Eventually the bot tries to get more of your personal biographical information (number of siblings, favorite color, favorite movie, location etc.)

Then we add a personality test. The Interviewer tries to classify the botmaster's personality type in order to select a "basic bot personality profile," in other words an AIML set customized for a specific personality type. The Interviewer will also ask the botmaster to enter some pickup lines to customize the AIML personality. (In case it is not obvious, the goal here is to "digitize" the botmaster's personality).

The results of this interview can be "dumped" into a new bot personality. And, boom, in about 2 hours, your personalized custom chat bot is complete and ready to chat for you!

Someone asked, what other language acquisition system is there besides learning. Well, botmaster training is one example. I have argued that a group of funded botmasters could reach the "goal of A.I." (however that is defined) before the learning machine group even defined its architecture. Even faster than a child can learn language.

Here is another scenario. Suppose we had a "botmaster accelerator" tool, based on a kind of personality test. The botmaster answers questions about his/her personality, background and preferences for a couple of hours. Then, the program classifies the botmaster into one of, say, 64 basic personality types. Using the basic "personality profile" for that type, the entire AIML set is automatically generated. This may not be an exact digitization of the botmaster, but it may be close enough for a first pass or say, 60% accurate (the same as a personality test). By applying the individual's biographical background and preferences, the bot is not simply a "clone" of one of the 64 personality types, but a unique individual personality.

The stark implication here is that (theoretically) we may be able to reduce the entire process of language learning and personality development to a couple of hours with a computer!

Consider the "Botmaster Accelerator" tool. We can speed up the personality digitization process by administering a personality test, and then selecting one of nine to twelve or sixteen (say) prototype bot personalities, rather than editing the entire ALICE brain, or starting from scratch. Then, using bot properties, we can instantiate the bot with biographical data and preferences.

Now, it doesn't really matter which personality classification system you pick. In some sense what you said is true, they are all bunk. Or maybe,

they seem to work 60% of the time, for 60% of the people, perhaps.

Now suppose the first botmaster menu is:

Do you prefer:
- Luscher Color Quiz
- Astrology
- Enneagram
- Meyers-Briggs
- Pick-a-celebrity
- Homeopathy
- None of the above

The Personality Test for the Astrology choice is simple: "What is your birthday?" Now, it doesn't matter if the Virgo bot personality really matches the botmaster's personality or not, as long as the botmaster believes it does. The customer is always right. If the Four botmaster believed the Enneagram system was more reliable, then she might get a better result with the Enneagram Four bot. If the botmaster believed that all personality classifications are bunk, then they could do it the old fashioned way we do it now with "None of the Above".

CONCLUDING REMARKS

I have a very interesting chart of technology adoption during the last century. It was originally produced by the Wall Street Journal, but I have a small copy reproduced in the book *Rules for Revolutionaries* by Guy Kawasaki (Harper, 1999). The chart shows, for a variety of technologies from the telephone to satellite TV, the growth of each in the marketplace as a function of year. The Y axis is expressed in percentage of (U.S.) households, so none of the curves exceeds 100%. Generally all of the curves follow an S-shape, that looks like exponential growth at first, but then eventually levels off as the technology saturates the marketplace.

Television is an instructive example. Introduced commercially in 1947, growth was slow until the "knee" in 1949-51. Following that, TV spread at an approximately linear rate until about 1958, when it began to slowly level off. By 1980, nearly 100% of households had TV sets.

Philo T. Farnsworth, like myself a native of Maine and resident of San Francisco, invented the cathode ray picture tube here in 1927. He had to wait over twenty years before his invention saw the commercial light of day!

Another interesting trend is the increasing "steepness" of the curves. The telephone, introduced in 1876, did not saturate the market until about 100 years later. The curve for AM radio starts in 1923 and levels off in 1963, a span of about 40 years. The VCR was launched in 1980, but levels off in 2000, only about twenty years. Of course, there are a lot of other unfinished curves crowded near the end of the graph--cell phone, PC, internet, camcorder etc.--whose future courses remain uncertain.

Another analysis might categorize the curves into those that look more like an S-curve and those that look more like a straight slope. The curve for the telephone is actually a bit ragged, taking a big dip downward during the Great Depression from 40% to 30% of the market (the same cannot be said of any other technology on the chart), but overall the phone curve is more like a long straight line than an S-curve.

Cable TV is also flatter than many others. From its inception in 1976, you can almost draw a straight line to its present position at 68% market share. The PC also runs in a straight line from the coordinates [1980, 0] to [2000, 45%]. In some sense, this is why the Cable TV and PC businesses are so predictable and boring. Winning over that last remaining half of households is just a matter of time and not making any big mistakes. It also tells you why people like Bill Gates always want to expand into new business areas like games and entertainment. Even when the PC market saturates, Bill can only be twice as rich as he is now. To get more, he needs another line of business.

What does all of this have to do with A. L. I. C. E. and AIML? People have often imagined that A. L. I. C. E. is like Netscape in 1995, poised to take off on the next great S-curve of growth. But I have argued it is more like Apple in 1975, or the whole PC industry at that time. We are mainly amateurs and hobbyists, with few real business applications we can point to. Yet we all have the same kind of "fever" that gripped an earlier generation of PC enthusiasts. We KNOW this is the Next Big Thing. We KNOW we are the vanguard of the revolution.

The first Apple appeared in 1975. But the technology adoption curve for PCs doesn't even begin until 1980. Those early PC pioneers were like Philo Farnsworth and his TV picture tube. They knew that they had a great thing, but getting people to adopt it was like feeding them liver. Jobs, Wozniak, Gates and friends only had to wait five years, compared to Farnsworth's twenty. I've been at A. L. I. C. E. and AIML for almost seven years now.

I don't know if we are on the knee, on the ramp, or if the adoption curve has even started yet. I have learned in seven years not to expect miracles, and to have patience. Unfortunately we need the perspective of ten or twenty years to see what is really happening here.

I conclude with a quote from Kawasaki's book:

"Don't worry about people stealing your ideas. If your ideas are any good, you'll have to ram them down people's throats." -- Howard Aiken

REFERENCES

Prolog References:
1. Prolog Tutorial
http://cs.wwc.edu/~cs_dept/KU/PR/Prolog.html

2. A Short Tutorial on Prolog
http://cbl.leeds.ac.uk/~tasmin/prologtutorial/
3. Prolog Programming: A First Course
http://cbl.leeds.ac.uk/~paul/prologbook/
4. GNU Prolog site
http://pauillac.inria.fr/~diaz/gnu-prolog.

References on Zipf's Law:
http://www.cut-the-knot.com/do_you_know/zipfLaw.html

http://linkage.rockefeller.edu/wli/zipf/

http://www.useit.com/alertbox/zipf.html

1. Documentation
http://alicebot.org/documentation
2. Don't Read Me: Program dB
http://alicebot.org/articles/wallace/dont-dB.html
3. AIML Specification (Working Draft)
http://alicebot.org/TR/2001/WD-aiml

ACKNOWLEDGEMENTS

This research was funded by donations to, and carried out
by volunteers of, the ALICE A.I. Foundation, Inc., a
nonprofit charitable research corporation.

The author is grateful to Peter Plantec for his careful reading and comments
on an early draft of this manuscript.

COMMENTS ABOUT THE AUTHOR

Some of the best comments about the
Slashdot Interview with Dr. Wallace on http://slashdot.org/interviews/
At the time it appeared, it was the longest single article published in
the history of slashdot, and had to be broken into three parts.

"If you can't find a useful quote in this interview to use as a sig,
you're weak... really weak."

"this is the *BEST* interview I've ever read on /. bar none."

"there is some serious doubt on this very forum whether this is ALICE
or the good Doctor."

"He does a good job off coming off as a troll without his very own
impersonator."

"This was one of the best /. interviews I've ever read. This guy is a genius."

"this man is brilliant and I only wish he had written more."

"Great interview. Probably the best I've ever read on Slashdot (and I'll definitely come back eventually to read everything I glazed over). Does anyone else think it's strange that the leading AI researcher in the world is a self described 'mental patient?'"

"This is honestly one of the best interviews, or literary pieces I have ever read. He is one of the most though provoking people I've read, and I'd honestly like to meet the man."

"I think the man is rather smart - either he's got us all thinking he is ALICE, or he's actually got us thinking ALICE is him. Either way, he's won."

"I have much more respect for Wallace after reading this reply. He's a deeply insightful individual and doesn't appear to be taken in by much of the bullshit of the AI field."

"Goddamn, what a thoughtful set of paragraphs. This is the first slashdot article I've decided to print. I don't care about the length."

"who knew an AI specialist could be such a skilled writer. amazing interview."

"The insights on AI, particularly, the digression into the functions of AIML for A.L.I.C.E were wonderful in this interview."

Index