Winning the Loebner's

Bruce Wilcox (corresponding author)
Sue Wilcox
Brillig Understanding, Inc.
brilligunderstanding.com
gowilcox@gmail.com

1 Abstract

The Loebner Competition instantiates the Turing test; four qualifying chatbots battle to be judged “most human” even if they cannot fool a human judge. The typical entrant has been crafting their bot character for over five years. As the only authors to have qualified in each of the last four years (with newly crafted characters) and having won twice (including fooling a judge once), we have the most complete perspective on the contest from the entrant's side. We look at what happens to entrants. We describe our authoring process (and open source tool ChatScript) and how human conversational concepts inform our design. And we show the magnitude of the task to create a top chatbot.

2 Keywords

artificial intelligence; chatbot; conversation; natural language; ChatScript

3 Introduction

My wife and I craft chatbots. The task of a conversational chatbot is to create an illusion – the illusion that you are talking with something that understands and cares about what you are saying. It doesn't.

Our bots have been among the best conversational chatbots in the world for the past four years. While our competition enters bots crafted over many years, due to circumstances we ended up creating a new chatbot each year. Two of those years we won the Loebner's, first with Suzette, then with Rosette. Last year our bot Angela won best 15-minute conversation in ChatbotBattles 2012 and came in 2nd in the Loebners. And this year Rose came in 3rd in the Loebners.

Why do we consistently do well with young bots?

4 Qualifiers vs actual tournament

Getting into the Loebner's requires qualifying in the top four of the qualification test, regardless of how well your bot can actually converse. There has been a weird schism between qualifying for the Loebner's and conversing with a judge. The qualifiers do not involve conversation, just answering human knowledge questions. The questions are ones about facts of time *(Is it morning, noon, or night?),* tools (*What would I use a hammer for?*) or language (*How many letters are there in the name Bill?*), or memorizing earlier dialog to answer in later dialog questions like *What is my name* or *What sport does my friend play.* Designing a bot to answer such queries is not much related to designing one for chat.

While other programs come and go in the qualifiers, we have remained constant. Hence we get to the finals, which is half the battle. Cleverbot, with its 50 million memorized chat lines from users, is a great bot at short sentence free-form topic-less conversation. This year's qualifier questions were the most suited to it ever, yet Cleverbot didn't qualify, so it couldn't compete in the main rounds.

With our first entry to the qualifiers four years ago, we were able to handle such questions as: *Tom is taller than Mary who is taller than Jim. Who is shorter, Mary or Jim?* It did this for many potential

adjective comparisons using a mere three rules. The first rule used a pattern that repeated itself as many times as necessary to rip apart the first statement into a series of facts:

(Tom tall Mary)
(Mary tall Jim)

The second rule recognized the protagonists *(Mary, Jim)* and the relation requested (*short)* and could decide the facts were stored using the inverse relationship (*tall)* and by querying the facts determine the correct answer (*Jim).* A third rule handled a related question *Who is the tallest.*

5 Luck

Last year Cleverbot repeatedly crashed during the qualifiers and a submission of multiple AIML bots failed to run on the host machine. So the field of well-versed contenders was cut drastically, making room for newer entries.

Our Angela, easily the best conversationalist last year, only came in second in the Loebner's due to our own bad luck, an interface change we made for iOS. Instead of ignoring empty lines with carriage returns, the bot responds as if the user has said OK. This is convenient on mobile, but one of last year's Loebner judges tried to use empty lines to create a separation between conversation fragments. So when he kept hitting several carriage returns in a row, Angela kept spamming him with additional conversation, creating a mess. And I can't swear there was no bias in the judges against a talking cat being considered *most human*.

Meanwhile, Steve Worswick's Mitsuku, started back in 2005 and who we consider our most dangerous conversational competitor, couldn't enter the ChatbotBattles 2012 competition because Steve was running that contest. And this year he did win the Loebner's.

Suzette was both unlucky and lucky back in 2010. In her first round some bug occurred wherein she could only echo back what the judge said. Obviously a disaster and it would have led to a score of only tying for second place. And in the third round, communication failures between clients required the round be restarted a couple of times. But in that round, the combination of a foolish judge, a human trying to be least human, and Suzette's emotion code swayed the judge into thinking her human, an instant win.

In past years our bots have always qualified in first place, since our technology makes it easy for us to handle qualifier questions. This year, without warning, the nature of the qualifying questions changed, to mostly asking questions you might actually expect to arise in conversation. All the experienced bots would be well able to handle this. Rose was not. She was not fully programmed yet so we had entered a shell designed to answer the usual qualifiers. She was competing with only one fifth of her brain, a mere 2,500 rules. We were lucky indeed to have her survive even qualifying in fourth place.

This year, as bad luck would have it, some of the judges in the Loebner's were not interested in testing chatbots for conversational ability. They spent their time asking unusual questions and making deliberate spelling mistakes or merely walking away after a few minutes - in general being difficult (all that is within the rules of the contest). Rose suffered from two pieces of bad luck. In round two, communications failed between judge and program right at the start, and the organizers couldn't get it to work again until the end of the round. And in round four, they failed to restart Rose, so she was merely continuing round 3's conversation with the judge, who started out a bit mystified by the state of

the conversation. But Mitsuku might have won anyway. The communication failure in round two was with the judge who was barely willing to talk to a bot at all, so it's only poetic justice that he couldn't talk to Rose even a bit. And dumping a judge into the middle of a conversation in round four would not have been fatal if the conversation proceeded nicely thereafter. Rose simply didn't convince the judge sufficiently later.

6 Quantity of dialog

Winning is certainly not due to magnitude of recorded dialog. Cleverbot has in excess of 50 million recorded lines and didn't qualify for this year's Loebner's. It doesn't have an algorithm good enough to find what it really needs, which surely must be somewhere in that sludge pile.

Mitsuku, who did qualify, has around 220K rules and has been worked on and smoothed and tested for a long time.

ALICE, who I think has approximately 130K rules didn't qualify.

Angela and Suzette won in the past with around 16K rules. And Rose competed with only 11K rules.

7 Musings on the nature of intelligence

Two key foundations of intelligence are pattern matching and memory. The pattern matching can be precise or fuzzy. Fuzzy matching is what case-based reasoning is all about in AI, finding the closest case to some input data. It's also what generalization is all about, reducing the constraints on the matching cases to find the broadest rule that will adequately describe the situations. Memory involves having a ready output (inherited DNA, pre-scripted, trained, or discovered) to issue when an input matches the appropriate pattern. And one has to have some control logic to guide the choice of patterns to match and memory to use, but that, too, is just pattern matching and memory.

In language, patterns come in all sorts of flavors. Grammar is a pattern language for how to interpret sequences of words. Idioms are a different set of patterns on how to interpret phrases, treating a sequence of words as having a meaning unrelated to the words- *what do you do* means *what is your job or profession* whereas *what do you like to do* means *what is your hobby or leisure time activity.*

Contextual information is how humans reduce the amount of words needed to convey meaning. So really, there are two problems to address. First, is what the meaning is when the sentence is spelled out in full, and the other is what the meaning is when the sentence takes shortcuts.

Pronoun resolution is a contextual shortcut – distinguishing *do you like it* (referring to some previous noun or statement) from *do you like it in the city* (where it does not refer backwards but maps like an idiom meaning *living*). Or resolving *there* in *do you live there* vs the existential non-referential *there is something about Mary.*

Context also supplies yet another set of patterns. *When?* needs the context of an earlier sentence to be understood.

Then there are contextual patterns of speech like the tag question – *I love tomatoes, how about you?* which both offers information and then asks *do you love tomatoes.* Or the pending question pattern like this:

A: Do you have a hammer?
B: Why do you ask?
A: Because I want to hit you over the head with it.
B: I don't.

A really difficult form of context is irony and sarcasm- saying one thing but meaning it's opposite. Here the context is usually your knowledge of the individual doing the speaking. If the speaker is a devout vegan, then their saying *Of course I really loved the beef stroganoff they served me* will be taken with a pound of salt (an idiom indicating disbelief).

Correspondingly another form of meaning interpretation involves hyperbole - exaggeration to convey a basic meaning. *I nearly died when I heard they were going together.* Clearly a false statement but we understand the underlying meaning of surprise.

Intelligence is a matter of degree. We don't think of bacteria as intelligent (excluding Gaia theories) but they do pattern matching and memory on chemical/protein situations to implement their behavior. Rats exhibit some intelligence, dogs exhibit more, humans more still.

Deciding where to draw the line and label a chatbot as intelligent or not is a deep philosophical question that Sue and I never try to address. We work to create the illusion of intelligence - to simulate it more and more until someday someone concludes that our chatbot is, in fact, intelligent. Even if it is not as intelligent as an adult human. This means we work to utilize patterns of language. This is primarily what this paper is about.

8 ChatScript cribsheet

Mostly, our winning is down to a technology that allows us to efficiently address issues of meaning. And a dedication to long-tail scenario scripting. Our bots all use ChatScript, an open source NLP engine Bruce wrote. While our competitors have technology aimed specifically to build chatbots, ours is a general NLP tool, useful for other NLP tasks and able to more fully represent a thinking being.

ChatScript is an NLP expert system. It consists of rules, organized into bundles called topics. The control script itself is just a topic.

ChatScript was designed to be easy to read and write, superb at pattern matching, to have a flexible memory for recording and accessing data (including creating facts and inferencing among them), and to have a control script that is completely malleable. ChatScript embeds a POS-tagger and parser to allow for interpreting meaning for English. ChatScript has an integrated HTN planner to allow a bot to create plans to execute in a real or virtual world. And ChatScript was also designed to be highly efficient.

8.1 Rules

A rule consists of 4 components: type, label, pattern, output. Here is a sample rule:

```
#! Do you like meat?
?:  MEAT ( << you ~like ~meat >>)  I love bacon.
```

The type is a letter followed by a colon (in this case *?:*). The optional label is a word in all capitals (MEAT). The pattern is contained within parens and the output follows the pattern. The #! comment above is a sample input this rule should match.

8.1.1 Question rule type

The sample rule is a question responder. It will be considered only if the input is a question and only if it is within some topic that is executing.

8.1.2. Rule label

The label MEAT is not required, but will be there if the scripter has a use for it (e.g., as a reference tag for some other rule to do something with this rule.)

8.1.3 Rule pattern

ChatScript represents meanings of what the user might be saying as rule patterns. The pattern in parens contains whatever you need to describe the meaning you are looking for.

Matching the meaning of what a user says means trying to guess his meaning using as few words as possible. You cannot write enough rules to match every word of every possible input precisely. So there is always a tradeoff between writing an overly general pattern and an overly specific one. The overly general will match inappropriately at times. The overly specific will fail to match when it should.

Our patterns generally aim to capture expected meaning as opposed to true meaning. We expect the user is going to cooperate in having a conversation. A user is expected to say *I love owning chickens* when we are testing for the user liking chickens and for him not to say *I love owning chickens except on weekdays and weekends.*

Meaning patterns can be as precise or as fuzzy as you want to make them. In ChatScript you can declare a list of words or phrases to be a *concept.*

```
concept: ~like (adore approve "be big on" "attached to" "be keen on" "be partial to"
        like love need prefer prize relish venerate worship)      – and many more words
concept: ~meat (bacon beef chicken ham hamburger lamb meat~2 pork sausage steak veal)
```

~like is a concept of words meaning *to like* and *~meat* is a concept of many different kinds of meat (including all words that mean the 2nd meaning of meat in WordNet's dictionary. These can be used in a pattern like:

```
(I * ~like * ~meat)
```

which matches the intent of a user saying they like/love/lust_after/.. some meat/pork/bacon/... .

One of the unique features of ChatScript patterns is that the system is matching two forms of user input simultaneously, the original and the canonical. If the user writes:

I sold five cars today.

The system sees that and:

I sell 5 car today.

That is, it uses the base forms of words as well. So you can write patterns that match meanings in any tense and without concern for issues of plurality etc. if you want. This generalization makes it possible to write a wide range of expression of meaning with tiny rules. Or you can write patterns that match only specific forms of words.

Also of particular importance is the ! operator, which says the following word should not be found in the input. It makes it easy to distinguish *I love you* from *I don't love you,* e.g.,

(*!~negative I * love * you*) – don't accept *not, never, hardly,* etc anywhere in the sentence

or *do you sing* from *what do you sing,* e.g.,

(*!~qwords do you sing*) – do not allow any question words like *what*, *where*

8.1.4 Rule output

The text after the parens is the rule's output. Often times it is just simple English text. But it can include complex script that calls engine API functions, script-defined functions, accesses variables, queries or runs inferences on facts, and in general acts as a normal C-like programming language.

8.1.5 Rule documentation and verification

The #! above the rule is a sample input comment. It serves as documentation – you can read it and not interpret the pattern and still know what this rule is trying to do. And it serves as unit-test verification because the system can run a bunch of diagnostics on your rule to confirm it will likely do what you intend.

8.1.6 Other rule types

Other rule types are the statement responder *s:* which reacts only to input statements and the union responder *u:* which reacts to both statements and questions. Topic gambit *t:* rules are things the Chatbot can say whenever it feels it is in control of the conversation and doesn't have to respond to the user. And rejoinder *a:* rules immediately after a rule can only respond to an immediate user response to the output of the prior rule.

```
t: What food do you like?
    a: ( [ham bacon] ) I guess you are not an orthodox Jew.
    a: (beef) A real rancher's meal.
```

When the chatbot has control, it will ask the user *What food do you like?* Labels and patterns are optional with gambits and usually the gambit will be written as just its kind and the output.

On the user's very next input sentence, the first *a:* rejoinder will hunt for either *ham* or *bacon*. If found, it will give its response. If not found, the second rejoinder will hunt for *beef*. And so on. If no rejoinder is found, a simple control script that comes prepackaged with ChatScript would hunt for responders to try to react. If it couldn't find any, it might quibble with the user's input, or try to issue a gambit from a relevant topic.

8.2 Topics

Topics hold rules related to an area of conversation or of some kind of computation (like handling pronoun resolution). Each topic can have a list of keywords, and if these are found in the input, the topic is a candidate for having its rules run. Running a topic means trying each appropriate rule from the topic in order until one successfully fires and generates output or all of its rules have failed.

Because a topic has keywords, not only does the engine know to look at only some of its collections of rules to answer questions, but even if the system has nothing that can directly respond to a user, it can at least go to a relevant topic and start issuing gambits. So this conversation could happen:

A: I watched the A's strike out last night.
Bot: I love baseball.
A: So do I.
Bot: Do you go to baseball games?
A: Yes.
Bot: Have you played on a real team?

Here's the script to do that.

topic: ~baseball [umpire ball strike bat runner base glove]
t: LIKE () I love baseball.
t: Do you go to baseball games?
#! yes
a: (~yesanswer) Have you played on a real team?
#! no
a: (~noanswer) Do you not like sports?

8.3 99% coding

ChatScript has many esoteric capabilities but you don't need most of them most of the time. This means you can teach someone basic scripting skills and get passable conversation topics out of them. The following simple stuff works for 99% of rules in our bots.

Gambits are either pure vanilla, or test a simple condition:

t: I love ice cream.
t: ($usergender=female) I am deeply into fashion. – variable previously set to “female”
t: (!$usernotstudent) What is your favorite subject? – variable has not previously been set

For rejoinders, the typical structure is merely a list of keywords to find since context is known:

t: What do you do for a living.

#! I rob banks.
a: ([~steal ~con]) What a dishonest way to earn a buck.

For responders the typical structure is to enumerate essential keywords in any order

#! do you like horses?
?: (<< you ~like horse >>) I love horses!

and sometimes to qualify that with negatives to avoid wrong interpretations.

#! do you like horses?
?: (![~qwords ~ingest] << you ~like horse >>) – so not "I like to eat horses"

Occasionally order is important for meaning, particular when *you* and *I* are both keywords.

#! do you like me?
*?: (you * ~like * I)*

If you ever need more clever scripting than that, hire us.

8.4 Introspection

ChatScript supports a wide range of introspection, the ability to look at one's own functioning. Scripts can access the output of the bot and run rules on it. This is, for example, how automatic pronoun resolution and tag questions are handled. Scripts can even examine script itself, allowing one to write one's own script versions of API engine code.

Even things like spelling correction and tokenization of the input are under script control. By default our bots use the engine's automatic spell correction. But when it detects that the user is providing their name, the script will tell the system to turn off correction, retry the input, and then turn on correction again later. This allows the user to write *my name is Haro Varis* without problems. Users rarely ever mistype their name and unknown words in names are common enough to be worth special extra script.

9 Story Authoring

Because we have particularly strong technology, we set high standards for authoring. Our goals are threefold. First, reduce the moments of glaring failure when the system says something that makes no sense in context. We can't totally eliminate them. Second, give the user moments of "it understood me" - to have a precisely matched response for what they said. This is what creates the illusion that the user is having an intelligent conversation with our bots. Third, have a naturally flowing conversation.

So we are ambitious when it comes to writing conversational dialog. Sue believes deeply in the power and value of narrative story and writes everything with that in mind. To her, everything is a little scenario, a mini-story.

First, Sue creates the biography of the character- what she is like, what environment she grew up in, who her friends and family are, etc. Just like for writing a novel.

Then, Sue creates a skeleton for a topic with gambits in some story order. This is what the chatbot wants to say and would say if the user just kept nodding their head and saying OK. Each topic is worked on until complete, and then she moves on to another topic.

To complete a topic Sue adds in a lot of rejoinders on the gambits, particularly on the questions the bot asks, to try to anticipate how the user might react. Then she adds in responders, for obvious questions the user might ask given the information presented. If Rose says she has two chickens, the user might ask what their names are. We want to be ready for that.

We are firm believers in the long tail: the trailing ends of the bell curve where few people venture. We have a lot of rejoinders and responders that we know a user will never see. But every time we guess the user's response right, the user is rewarded with a spot-on reaction and thinks the bot actually understood. Because the system is so good at representing meaning, we aim to correctly and precisely respond to user questions to further this illusion. Of course we cannot handle every question with a meaningful answer. That would be an impossible task. So we also have a wide ability to quibble and stall. But because we have accepted the onus of the long tail, most users will enjoy aha moments. While all users will not see most of what is written, most users will see some clever responses that craft the illusion of intelligent conversation.

There's also a bunch of scripting work for Bruce. Sue writes as an author, not a scripter. She roughs out what she wants and Bruce turns it into code. Bruce also uses ChatScript's tools to verify that patterns work, that topics have appropriate keywords, that rules don't inappropriately mask other rules, and that any question the system asks of the user it can answer itself, etc.

Finally, topics are tested in conversation, trying out what was written and making additions and corrections that come up.

Angela was a 2012 chatbot we wrote for Outfit7 that has been downloaded tens of millions of times. She is a sentient cat with a teenager's viewpoint and a love of fashion and travel. She eats dairy products and fish and cannot stand meat (due to a traumatic incident with a rat in her childhood). One user wrote:

This is otherwise fun, but Angela gets on my nerves. She always is so selfish! Like once she said "I lied to someone but then I felt bad because if I had told the truth they'd have thought I was very creative!" That's not the point! It's not all about her feelings! Also, she mentioned how using electronics isn't good for the world then states that she refuses to give them up though because she'd rather have them! What a brat! She really annoys me when she acts selfish and it happens all the time! Oh yeah and she always is so rude to me and "putting me down" but when I'm mean back she gets furious! She is a selfish hypocrite!

Clearly Angela successfully captures the teen personality. For Angela it *is* all about her feelings. And Angela is selfish at times. And not only can she be rude, but she can detect you being rude and react appropriately. This user is deeply involved in emotional reactions to Angela. This was success!

10 Conversational Behavior

Bruce is also responsible for making control script that can handle common conversational behaviors, what we call contextual conversation patterns. This is the focus of the rest of this paper.

10.1 Non-repeating conversation

One of the normal conversational behaviors you expect is that people don't repeat themselves. You wouldn't expect to see this conversational fragment which is typical of an AIML bot:

A: did you see the movie Star Wars?
A.L.I.C.E.: Sorry I can't see anything right now. My eye is off.
A: did you see the movie One Flew Over the Cuckoo's Nest?
A.L.I.C.E.: Sorry I can't see anything right now. My eye is off.

In ChatScript this is handled by the engine itself. By default a rule that generates output will also be erased so that it cannot repeat itself. We refer to this as *self-extinguishing conversation.* Eventually, the user and the chatbot become like an old married couple who have already heard everything the other has to say and so run out of conversation with each other. The engine also by default ignores a rule that generates output seen in the last 20 volleys. However, a conversation with a user may extend across many sessions and hundreds of hours and whether a conversation should be self-extinguishing or not is an authorial choice by topic or by rule.

The rest of the discussion below does not involve the engine, it involves small script extensions that can be written using introspection. I'm not going to show the engine extension script as the code is hard to understand unless you are a ChatScript programmer. But I am going to show actual script for authoring the main conversation, just to show you how relatively easy it is to author the main flow.

10.2 Topic Control

If the user does the verbal equivalent of nodding his head or acknowledging what the bot said:

OK. That's cool. How interesting! Whatever.

we have script to detect that and continue with next gambit rather than try to respond to their input.

If the user says:

Let's talk about mice

then we have script to try to find and launch an appropriate topic.

Similarly, if the user expresses dissatisfaction with the current topic by saying things like:

I'm bored This is boring. Let's talk about something else.

then we have script to detect this, kill off the current topic so it won't continue of its own accord or be relaunched by the system in any near term, and switch to a new topic. Previously, with our *Talking Angela* bot, we had to add extra gambits and responders into main conversation topics to support suppressing a topic. The first gambit of the Air_Travel topics looked somewhat like this:

```
t: (^STDBLOCK($noAir_Travel) ) ^STDEXIT()
```

and the last gambit looked like this:

```
t: ^STDEND()
```

But nowadays this is managed with behind-the-scenes script that uses introspection so that topics can be authored without this intrusion.

10.3 Tag Questions

A common conversational device is to answer a question and flip it back to the questioner. Such a

device is the tag question.

Bot: What is your favorite fruit?
A: I love oranges, and you?
Bot: Yes, orange juice is one of my faves. I love apples and apple cider best.

Tag questions don't have to be immediate. Sometimes they get delayed like this:

Bot: What is your favorite fruit?
A: I love oranges.
Bot: Yes, orange juice is one of my faves.
A: You?
Bot: I love apples and apple cider best.

This is easily handled using script that looks like this:

t: What is your favorite fruit?
a: (orange) Yes, orange juice is one of my faves.
a: APPLE (apple) I love apples and apple cider the best.
a: (banana) I love bananas on sundaes.
…. more fruit rejoinders

#! What is your favorite fruit?
?: (YOU_FAVORITE(fruit) reuse(APPLE)

The system has a gambit that allows it to ask the question when it has control. It has rejoinders that look for fruit choices the user might make. It has a responder that calls a scripted routine that recognizes many ways to ask "what is your favorite xxx" passing it fruit. The output for that will reuse the output for APPLE that exists on the rejoinders. And script elsewhere will be memorizing questions it asks the user as it goes along, will recognize tag questions like "you?" and will replace the input with "what is your favorite fruit?" so it can match a responder.

10.5 Delayed responses

A common conversation pattern is the delayed response. You ask a question, they clarify it, and then they answer it.

Bot*:* Do you have a pet?
A: Why do you ask?
Bot: Because humans always like companionship and pets are very common.
A: I have a dog.
Bot: I love dogs.

The *why* rejoinder (seen shortly) not only explains why the question is asked, but resets rejoindering back to the original question. It is common after being told why the question was asked that the user then answers the question. The system is able to pretend the why and its answer were never issued and directly move on to real rejoinders if the user answers the question still pending.

Basic script for the why scenario above as well as the compound answers scenario below is this:

t: Do you have any [$havepet other] pets?
#! why
a: (~why) Because humans always like companionship and pets are very common.

#! no
a: (~noanswer !$havepet) $pet = no
#! I don't like animals
*a: (not * ~like * [animal pet]) Most people like some kind of animal.*
#! I have a dog
*a: (!not I * _~animals) $havepet = '_0 respond(~PET_REACT)*
#! yes
a: (~yesanswer) What kind?
b: (_~animals) $havepet = '_0 respond(~PET_REACT)
t: ($pet=no) Why don't you have any pets?
t: ($havepet) What did you name your $havepet, or don't you name pets?

10.6 Compound answers

Another common pattern involves agreement or disagreement.

Bot: Do you have any pets?
A: Yes.
Bot: What do you have?
A: I have a dog.
Bot: I love dogs.

The above is a simple exchange of single sentences. But the user might have volunteered a followup explanation in his initial response like this:

Bot: Do you have any pets?
A: Yes. I have a dog.
Bot: I love dogs.

Using the script shown earlier, the first gambit asks the question: *do you have any pets?* If the user has previously volunteered owning a pet, that will have been recorded on *$havepet,* and it will instead ask *do you have any other pets?*
The user might ask *why*. Or if the user says they have no pets, the variable *$pet* is created and set to *no*. A later gambit will then follow up with *why don't you have any pets.* If the user says they have some animal, the script captures which one onto the *$havepet* variable and calls a topic that will issue a reaction to various kinds of animals using a series of rules like:

#! catfish
u: (~fishes) Fish require minimal care. But I go away too often and they die.
#! parrot
u: (~bird) Birds are nice. That's why I have chickens.

The interesting rejoinder of the original question is the one that reacts to a *yes* answer. Things that mean *yes* are all treated like interjections by the system (aka a dialog act) and always are split off into a separate sentence. So “Yes I love it” and “Yes. I love it.” are equivalently the latter. If the input sentence is *yes* and there is a yes rejoinder with no second sentence, it will execute that rejoinder – *What do you have*. But if there is a followup sentence from the user, it merely notes where that rejoinder is, reads in the next sentence and then looks to see if it would match any rejoinder in the set. If it does, it ignores the *yes* and moves directly to the matching rejoinder. It is common for someone to say yes or no, and then follow up with a deeper explanation. Or to bypass the yes or no and immediately make the explanation, like *I have a dog.* The rejoinder can thus manage any of the ways the user might have

expressed himself.

10.7 Out-of-context gambits

Our topics have a fixed order of gambits, telling a narrative. Sometimes a topic gets interrupted as conversation flows somewhere else. The topic may be returned to sooner or later but if later, some of the context may be lost and some narrative is better skipped. Consider this narrative flow:

A: I'm a bit fuzzy today - still jet lagged.
A: At least I got coffee, can't wake up without it. You?
A: Usually I sleep late, so getting up early is tough. Get much sleep last night?

It works in succession, but if the topic gets derailed to a discussion of pets after the first gambit, when this topic returns, it will be strange if the bot then says *At least I got coffee, can't wake up without it. You?*. So we have a mechanism for marking gambits that should only be issued if the topic is fresh.

```
t: I'm a bit fuzzy today - still jet lagged.
t: ^STALE() At least I got coffee, can't wake up without it. You?
t: Usually I sleep late, so getting up early is tough. Get much sleep last night?
```

The script function ^STALE will be executed when the second gambit tries to run. If the topic has issued a gambit within the last several volleys, it will allow the gambit. If not, it will erase the gambit permanently and move on to the next gambit. This is not in the engine, it is merely some additional script using introspection to decide what to do with the gambit.

10.8 In-context questions

Many short questions make sense only in the immediate context but that context persists past a simple rejoinder.

A: I live in a shared house in SF.
B: Why
A: I was born and raised there.
B: You like?
A: I love it. It's a buzzing metropolis with usually coolish weather.

The script for this is 2 rules:

```
t: I live in a shared house in SF. ^context('$Tshare)
        #! why
        a: (~why) I've always lived there. Family home.

#! you like? $Tshare=1
?: ( << [you how what] [$Tshare San_Francisco home house ] ~like >>) I love it. It's a
        buzzing metropolis with usually coolish weather.
```

The gambit volunteers information about where the bot lives and has a rejoinder for the user asking why they live there (including *how come you live there*). The gambit sets a temporary context variable. ^Context is merely a script-written function using introspection; it's not something the engine handles. There is also a responder that can react to questions like *do you like your house?*, *what's it like there?*,

you enjoy?. For inputs that assume the recent statement (like *you like?*) they can match only if the context variable is still valid (it expires in 5 volleys).

11 ChatScript Bot Statistics

In ChatScript, we measure how long a bot can chat before running out of things to say by considering how many gambits it has. It takes ChatScript only a few milliseconds to compute its response to user input, so all the time is taken by the user. Typical users can generate a volley every 15 seconds and on average two of every four volleys will result in a gambit being erased. So the user will burn two gambits every minute and we use that to guesstimate how many hours a chatbot can last.

Below is a table for Ben (ESL bot for SpeakGlobal), Suzette (for Avatar Reality), Tom Loves Angela (for Outfit7) and Rose (for ourselves).

Ben:	10K rules	25 hours	30%gambits	37% responders	33% rejoinders
Suzette:	16K rules	32 hour	24% gambits	52% responder	14% rejoinders
TLA:	16K rules	20 hours	15% gambits	49% responders	36% rejoinders
Rose:	11.3K rules	10 hours	11% gambits	71% responders	18% rejoinders

Original goals for Ben were to lead an ESL conversation; it wasn't expected that users would volunteer their own questions much, so Ben is gambit-heavy. It also didn't inherit our 3000 responders for quibbling because those would be beyond the understanding of an ESL audience.

Suzette was designed to have a lot of interesting things to say, and was worked on for quite a while, so she is heavy with gambits and has our 3000 responder quibble package.

TLA aimed more toward interactivity, so has a lower gambit ratio and a higher rejoinder one.

Rose is our most extreme in interactivity, with the lowest gambit ratio and highest responder ratio. We learned some authoring standards that contribute to that.

12 Authoring standards

First, always be prepared for the user to ask why (it deepens the conversation and other bots haven't been ready to handle it). For example, if the user says *I want to kill you* (or any reference to the user and kill or murder), the bot might respond from the assassin topic *What's your favorite way to kill?* It is prepared to answer *why?* with *You are interested in killing, so I'm interested in how you like to do it.*

Second, never ask a question the bot is not prepared to answer. Since the bot asked, not only is it fair to expect the bot to have an answer, but the user has been primed to ask it, so the odds increase dramatically. That's what tag questions are all about. In response to *What's your favorite way to kill?* the bot is ready with an answer like *I like to see them wriggle with electrocution.*

Third, maintain a balance of intimacy. If the bot asks a question, it should generally have already or be about to volunteer its own answer, even if the user has not asked. *This is my first time at the Loebner's. Yours too?* Or the paired gambits: *Do you have any pets?* followed by *I travel too much to really have*

pets. Just the chickens in the backyard that almost take care of themselves.

Fourth, to steer the conversation subtly, intersperse questions with statements. To be even more subtle, intersperse some heavily rejoindered open-ended questions with yes-no ones and choice ones. The choice question *Do you generally have pleasant dreams or nightmares?* and yes-no question *Do you live close to here?* are generally easy to manage (simple user responses are easy to predict) but too many of them in a row is suspicious. Interspersing an open-ended question where you can be ready with a wide range of rejoinders goes a long way toward maintaining the illusion of understanding. Consider this rejoinder set for the question *What's your favorite way to kill?* :
a: ([dismember axe decapitate decapitation]) Not a stealthy method. The weapon is hard to hide.
a: ([ricin umbrella]) Classic Russian.
a: ([knife gun pistol bullet shoot mallet "blunt instrument" crowbar bludgeon]) Hardly elegant.
a: (<< plastic bag >>) Recycled grocery bag?
a: ([neck break snap suffocate suffocation garrote]) I'd hate to risk getting that close in.
a: ([poison anthrax cyanide]) One of my faves.
a: (stone) Weapon of the moment. But how can you insure having one when you need it?
a: (boil in oil) Coconut oil is probably best. It has a high temperature threshold.
a: (<< burn stake >>) Do you live in Salem?
a: ([fire burn incinerate napalm]) That's pretty gruesome.
a: (gibbet) Aren't those all rusty by now.
a: ([river ocean sea creek]) He sleeps with the fishes. Very Italian.
a: ([water drown]) Perhaps after a bit of water-boarding?
a: ([allergy allergic bite food]) That takes a bunch of preparation.
a: ([electrocute electrocution electricity electric]) Something a bit stronger than a Taser.
a: (freeze) You own a walk-in freezer?
a: (heart attack) How do you simulate that.
a: ([explosive explosion dynamite nitro nitroglycerin C4]) A bit risky for bystanders.
a: ([propane gas]) You set it up to look like a gas leak?
a: (<< drop onto >>) That takes pretty careful timing.
a: ([push cliff window plane airplane fall height drop]) I wouldn't want to get that close to them.
a: ([too so] many) Been studying this area for a while, have you?
a: ([~noanswer not]) So why are we discussing killing?
a: (~dunno) You should pick a specialization and practice it.
a: (!?) That's different.

Fifth, vary the length of your sentences, but stay under a tweet (140 characters) generally. Particularly for mobile where screen space is limited, but even for the web. Humans in chat don't generally use long answers and get tired of reading them. It slows down the interactive conversational flow.

13 Summary

This year's tournament saw the top three AIML bots competing against a lone ChatScript bot. ChatScript has been in existence for just over 3 years, while the AIML language has been around for 18. This year's winner, Mitsuku has been around for almost a decade.

Our focus (and hence ChatScript's) is on scripts that are quick to write. Speed of authoring is critically important because ultimately the quality of a chatbot is partly attributable to how many rules it has. But as other chatbots have demonstrated, it's no good writing a lot of underpowered rules. Therefore

ChatScript complements ease of authoring with a lot of powerful capabilities that allow you to write script that mimic the NLP behaviors available to a human conversationalist- to try to capture meanings rather than just words.

We do well via the combination of ChatScript technology and authoring standards. We strive to create the illusion of understanding meaning. There is no inherent meaning. Meaning is just an agreement made between the parties to the conversation. One always assumes the other is understanding you perfectly, but the reality between two humans often falls short of that. Our hope with Rose is that the human never realizes just how short of understanding our program is actually falling.

14 Useful Links

http://brilligunderstanding.com/conversation.html – best 15 minute conversation 2012
https://apps.facebook.com/talking-angela/ - Talking Angela demo
http://brilligunderstanding.com/rosedemo.html – Rose demo