User Memorization in Conversation

© Bruce Wilcox, gowilcox@gmail.com brilligunderstanding.com

Revision 8-5-2013 CS 3.52

An initial impression of the bot hearing and understanding the user comes from a gambit asking the user a question, and a rejoinder replying specifically to what the user said. E.g.

```
t: Do you have any pets?
     #! dog
     a: (dog) Dogs make great pets.

     #! cat
     a: (cat) I prefer cats.
```

Or it comes from a responder able to answer the user's question. E.g.

```
#! do you have any pets
?: ( << you ~own pet >>) I have two chickens.
```

The next level of creating an impression is to remember some of the information the user has given, and drop it back into conversation later. This will create a vivid impression on the user.

This is easiest done by memorizing certain kinds of information and having a use for them later. Things like what pet they have, whether they like coffee or not, their age, etc. There are two places one can store longer term data variables and facts. Variables are the cheapest and most efficient thing to use. Here is an example of memorizing pets:

```
t: Do you have any pets?
    #! dog
    a: (%tense!=past dog) Dogs make great pets. $pet = dog

    #! cat
    a: (%tense!=past cat) I prefer cats. $pet = cat

    #! elephant
    a: (_~animals) $pet = _0

s: ( %tense!=past << I ~own _~animal >>) $pet = _0
```

The above will either catch an unprovoked statement by the user that he owns some animal (not owned in the past) or will catch a reference to some animal in response to the bot's question while commenting at the same time. We now have a value for `$pet`. This can be used later in the same topic/conversation as follows:

```
t: do you have any pets?

t: I have two chickens.

t: I love having chickens

t: ($pet) refine()
    a: ($pet=dog) Have you always preferred dogs as pets?
    a: (*) What do you feed your $pet ?
```

The pet gambit will not fire until it knows of a user's pet. Control would skip past it if the user has never named an animal. Someday in the future when it learns of a pet, it might issue this gambit if it ever randomly selected the pet topic again.

An even more impressive bot will bring up the pet again in a future conversation. How to write such script will be explained in a moment. But imagine at the start of some future conversation, the bot, after saying hello, says "I was walking by a pet store the other day and saw the cutest dog in the window, which reminded me of your dog. Has it learned any new tricks?"

There are two parts to the code for this. First, you have to detect you are in a new conversation. Then you have to decide when to bring forth the memory. The simple place to do this is in the introduction topic, when issuing and responding to "hello" and "how are you". You are only in this topic at the start of a conversation, so any knowledge you have memorized about pets clearly has to have been from a prior conversation. So you can try to find a moment when the user has issued a statement (not a question). So the topic might have a bunch of rules like this:

```
s: ($pet) I remember you have a $pet.

s: ($coffee) I know you like coffee.
```

Obviously the sentence output should be more sophisticated. Once this rule has fired, it automatically erases itself and will not fire again. Then again, maybe you want to initiate a subconversation on this by calling a topic to handle it. The topic might look like this:

```
topic: ~yourpet []

t: (!$pet) keep() end(TOPIC)

t: I was passing a pet shop yesterday and it reminded me of your $pet. Do you buy toys for i

t: What do you feed your $pet?
```

This topic has several special characteristics. First, it has no keywords, so it will not be launched accidentally by something the user says. Second, its first gambit

will make the topic exit if no pet has been learned about (and stays available for future use). This is because if the system is trying to find a random topic to gambit at some point in the future, we don't want this topic to do anything if we don't know a pet. The call to the topic from inside introductions can remain the same, or it can be changed to this:

```
s: ($pet) gambit(~yourpet) disable(rule ~)
```

The reason for disabling the rule is that when this rule is first called, it will generate output from ~yourpet, so this rule does not disable itself. That means during the next startup it will try to call ~yourpet again. That's not awful, because only new gambits will be issued. But maybe the topic is not written to be started from anywhere within it. Or maybe you don't want to start two conversations with references to their pet. So if you explicitly disable this rule when it fires, then all this is avoided.

An alternative to the above mechanism of sticking it in the introductions, is to have a topic you can initiate later from somewhere (presumably the control script) which goes and looks at memorized stuff and initiates conversations about them. If you absolutely want it to do this only in new conversations and not the same one where you learned the data, then you need to memorize when you learned the data as well as what. So your memorization might be:

```
$pet = _0 $pethwhen = %input
```

This means we could write script that can tell if this is a new conversation or not.

```
s: ($pet $petwhen<%userfirstline) I remember you have a $pet
```

$petwhen will have been set at volley 20, for example. %userfirstline is always the starting volley id of the current conversation. If the first conversation lasted for 50 volleys, then the start will be at 51 in this new conversation and at 0 in the old conversation. So the relation test fails as long as we are in the old conversation.

The most difficult part of bringing up remembered data is deciding when to do it. Finding a natural segway into the memory at the start of the conversation is best. It is perhaps more awkward if you try to find a break in the conversation to suddenly ramble about what you remember of the user. Which leads to yet another way of dealing with remembered data. Finding an excuse in normal topics to make reference to it. The problem is that the special code for that might get extensive.

Consider that we know they own a dog and the conversation is currently on food. You might have a gambit inside ~food like this:

```
t: ($pet=dog) Do you ever feed your dog table scraps during dinner?
```

This gambit will never execute unless they own a dog, so it's pretty specialized. But you wouldn't ask it for any pet, either:

```
t: ($pet) Do you ever feed your $pet table scraps during dinner?
```

Just imagine asking that about a canary or a goldfish. Of course you could have a set of animals that would eat scraps like this;

```
concept: ~table_pets (dog cat lion goat elephant)

t: ($pet?~table_pets) Do you ever feed your $pet table scraps during dinner?
```

Note that $pet is a variable that, once set, lives forever with the user data unless you explicitly erase it. Therefore it can be used all over the place, and only gambits and responders using it that generate output will get erased, the variable remains for other rules to trigger on.