Façade 이후 : 자연어 어플리케이션을 위한 패턴 매칭

(Beyond *Façade* :Pattern Matching for Natural Language Applications)

브루스 윌콕스(Bruce Wilcox)

가마수트라 등록일(2011. 03. 15)

http://www.gamasutra.com/view/feature/6305/beyond_façade_pattern_matching_.php

정정이 가능한 간단한 자연어(NL)를 이해하기 위해서는 방대한 지식과 다양한 추론 스킬이 필요하다. 또한 스펠링, 문법의 실수를 커버할 수 있어야 하며, 간단한 문맥적 상황도 이해할 수 있어야 한다. 엄격하게 자연어 사용을 제한하고 있는 것은 어찌 보면 당연한 일일지도 모른다.

그러나 말이 인터페이스가 되고 있는 시대가 왔다. 작은 디바이스에는 게임 콘트롤러나 마우스보다 더 적절한 입력 도구로 사용되고 있다. 제한적인 어휘나 파서를 활용하고 있는 구식 어드벤처 게임이라도 말로 입력을 할 때 훨씬 재미있다. 구글은 현재 말을 텍스트로 변환시키는 서버를 무료로 제공하고 있으며, 이로 인해 웹 페이지의 텍스트를 말로 사용할 수 있는 움직임이 주목받고 있다.

Scribblenauts 은 최신 퍼즐 게임에 명사와 형용사를 이용하였고, Telltale Games 는 명사와 동사를 섞어 쓰는 제품을 개발 중이다. 즉, 자연어로만 실행되는 게임은 이제 시간 문제라고 할 수 있다. *Bot Colony* 는 자연어 게임을 시도하고 있으며, *Façade* 는 이미 5 년 전에 이를 비슷하게 해 냈다.

이 글에서, AIML, *Façade*, and ChatScript 을 통해 게임 실행을 위한 자연어 처리에 대해 알아볼 것이다. 또한 단어 패턴 매칭에서 의미 패턴 매칭으로 변하고 있는 최신의 기술에 대해서도 알아볼 것이다.

이 글은 아래의 몇 가지 포인트로 구성될 것이다 :

1. 스크립트 문법과 의미가 프로그램을 이용하지 않고 콘텐츠를 제작하기 쉽게 해 준다.
2. 하나의 단어 보다는 여러 단어를 매칭하는 것이 의미에 부합하도록 하는데 더 좋다.

3. 단어의 부재는 단어가 있는 것 만큼 중요하다.
4. (문법적 분석 보다) 와일드카드 매칭이 잘못된 긍정적인 것을 줄이기 위해 공고히 되어야 한다.

AIML (1995)

제한 받지 않은 자연어 입력은 수십 년 동안 존재했다 - Eliza and A.L.I.C.E.(위)와 같은 chatbot 이 그러한 예이다. 그러나 결과물은 유용하지 않다. 자연어 이해를 위한 그럴듯한 추정을 위해서는 많은 데이터를 필요로 한다. 콘텐츠 오서닝을 쉽게 하기 위해서는 이러한 데이터가 주요한 요인이다.

A.L.I.C.E 는 AIML 로 작성되었다. AIML 은 오서닝을 위해 만들어진 시스템이다. 프로그래머로써, 나는 AIML 이 반복되는 자가 수정 코드인 것 같다. 즉, 콘텐츠 제작과 유지가 끔찍하다는 것이다. 게다가 패턴 매칭은 미미하다.

패턴 매칭

AIML 은 간단하다. 여러분이 카테고리(다른 말로 규칙)를 단어의 패턴, 와일드 카드 및 현재 입력된 것이 규칙에 매칭될 때 어떻게 할 것인가를 설정한다. AIML 의 신택스는 XML 기반인데, 아래에 설명한 것과 같이, **pattern** (input) 구문과 **template** (output) 구분으로 구성된다 :

```
<category>
<pattern> I NEED HELP * </pattern>
<template>Can you ask for help in the form of a question?</template>
</category>
```

패턴은 반드시 특수한 경우에 사용되는 단어를 커버할 수 있어야만 한다 ; 구두법은 고려하지 않는다. 와일드카드는 0 이 아닌 단어와 결합되고, 다양한 입력에 매칭할 수 있는 패턴을 생성한다.

오서링의 문제가 즉각적으로 발생한다. *나는 당신을 사랑한다는* 문장을 매칭하는 데 4 개의 규칙이 필요하지만 *나는 당신을 정말 사랑한다는* 문장은 매칭할 수 없다.

<category>
<pattern> I LOVE YOU </pattern>
<template>Whatever</template>
</category

<category>
<pattern> * I LOVE YOU </pattern>
<template>Whatever </template>
</category

<category>
<pattern> I LOVE YOU * </pattern>
<template>Whatever</template>
</category

<category>
<pattern> * I LOVE YOU * </pattern>
<template> Whatever</template>
</category

만약 AIML 와일드카드만 0 이나 그 이상의 단어에 매치된다면, 한 개의 패턴만 생성된다. 여러분은 위의 패턴을 *나*와 *당신* 사이에 와일드 카드를 추가하여 나는 당신을 정말 사랑한다는 매치한 패턴을 만들 수 있다. 그러나 이 경우 *나는 당신을 절대 사랑하지 않을 것이다와* 같은 문장을 수용하게 될 수도 있다. AIML 에서 와일드카드는 부적절하다.

반복적인 치환 (Recursive Substitution)

AIML 의 힘은 반복적인 치환에 있다. *<srai>* 태그를 사용하여 인풋을 제공하고 *<star/>*을 이용한 아웃풋에 contents of *을 넣는다. *What is LINUX?* 라는 질문에 C*an you please tell me what LINUX is right now* 를 맵핑하기 위한 규칙은 아래와 같다 :

<category>
<pattern> * RIGHT NOW <₩pattern>
<template> <srai><star/></srai></template>
</category>

=> *CAN YOU PLEASE TELL ME WHAT LINUX IS* and then

<category>
<pattern> CAN YOU PLEASE * <₩pattern>
<template> <srai> Please <star/></srai></template>
</category>

=> *PLEASE TELL ME WHAT LINUX IS* and then

<category>
<pattern> PLEASE TELL ME WHAT * <₩pattern>
<template> <srai> TELL ME WHAT <star/></srai></template>
</category>

=> *TELL ME WHAT LINUX IS* and then

<category>
<pattern> TELL ME WHAT * IS <₩pattern>
<template> <srai> WHAT IS <star/></srai></template>
</category>

=> *WHAT IS LINUX* and finally

<category>
<pattern> WHAT IS LINUX <₩pattern>
<template> LINUX is an operating system. </template>
</category>

=> *LINUX is an operating system.*

AIML 은 모두 이와 같은 방식으로 이루어진다. A.L.I.C.E.는 1300 개의 규칙으로 이루어져있고, *really* 와 *accordingly* 와 같은 불필요한 부사를 제거하였다.

확장된 패턴

AIML 은 2 가지 방식으로 활용할 수 있다. 아웃풋에 여러분이 설정할 수 있는 주제를 놓고 패턴 쪽에서 테스트를 하는 것이다. 안쪽의 매칭 토픽을 바깥 규칙에 얻을 수 있도록 우선순위를 준다. 그래서 토픽이 "맛있는 아침 식사(tasty breakfast food)"라고 설정되면, 이 규칙은 우선권을 얻게 된다.

```
<topic name="* breakfast *">
<category>
<pattern>I like fish</pattern>
<template> Do you like sushi? </template>
</category>
</topic>
```

이것에 대한 문제는 여러분이 매치시키는 토픽 이름을 저하기 위해서 이 토픽의 바깥에 규칙 설정에 시간을 들여야 한다는 것이다. 얼마나 많은 규칙이 필요해 질지, 어떻게 이것을 만들어 낼 지가 관건이 된다.

다른 패턴 증가 매커니즘은 that 절을 이용하는 것인데, 시스템에 주어진 아웃풋 문장에 패턴 매치를 부가할 수 있게 된다. That 절을 매치하는 규칙은 그렇지 않은 규칙에 비해 우선권을 가지게 된다. 여러분이 예상하고 있는 아웃풋에 기반하여 스크립트를 확장할 수 있도록 that 을 이용할 수 있다. .

```
<category>
<pattern>yes</pattern>
<that> Do * sushi </that>
<template>I hate sushi</template>
</category>
```

이 규칙은 sushi 로 끝나는 "Do" 의문문이 마지막 아웃풋일 경우에만 간단히 yes 에 응답하도록 되어 있다.

패턴 확장에 있어서, 여러분은 애매하게 영리한 코드를 쓸 수 있다. 그러나 애매한 코드는 일반적으로 만드는 이의 각고의 노력을 요하며, 소수의 사람만이 쓸 수 있거나 읽을 수 있다.

복잡한 계산

아웃풋에서, 텍스트를 쓰고, 그 자체를 새로운 인풋으로 사용하는 것을 제외하고, 여러분은 OS 시스템 쉘을 부르거나 자바스크립트를 이용할 수 있다. AIML 은 다양한 변수를 설정할 수 있게 해 주지만 토픽 변인만 제외하고 여러분이 패턴에 이를 참고하여 사용할 수는 없다. AIML 은 대명사를 처리하기 위해 변수를 이용한다. 여러분이 아웃풋을 작성 하고 있을 때, 여러분은 또한 새로운 가치를 구체화하여 영향을 미칠 수 있는 명사를 작성하거나 대명사로 인풋을 리맵하는 패턴을 작성하는 대명사를 작성할 수 있다. 부가적인 프로그램 작성이 늘어나지만 효과는 있다.

실행

AIML 을 실행하는 것은 쉽다. 모든 데이터는 트리 구조로 사전에 컴파일되고 인풋은 특정한 순서로 매치된다. 효과적으로 결정 구조를 따르기 위해서 트리의 노드를 따라서 진행된다. 첫번째 매칭 노드는 템플릿을 실행시키고, 이를 통해 새로운 매치가 시작되거나 또 다른 일을 할 수 있게 된다.

내재적 약점

AIML 의 기본적인 메캐닉은 약점을 가지고 있다. 순환적으로 자가 수정되는 인풋은 문제를 야기한다.

첫째, 여러분은 코드를 한눈에 알아 볼 수 없어서, 깊게 읽고 생각해야만 한다. XML 은 친절한 리더가 아니다. 매우 장황해서, 기계에 적합하지만 인간에게는 결코 쉽지 않다. 위에서 언급한 sushi 에 관한 that 규칙을 생각해 보면 간단히 이해될 것이다. 여러분은 한번에 이것이 무엇인지를 알 수 없다. 여러 번 고찰해 보아야 알 수 있다.

둘째, 여러분은 인풋과 매치될지 아닐지를 알 수 없으며, 규칙을 볼 수도 없다. 여러분은 모든 이전 변형 단계의 모든 것을 알고 있어야만 한다.

이 때문에, 프로그램 작성과 디버깅이 어려워만 지고 유지 보수는 불가능하다.

요약

AIML 은 영리하고 간단해서 초보자가 간단히 만들어 보는데 매우 유용하다. 그러나 15 년 후에, A.L.I.C.E 는 빈약한 120K 규칙을 가지게 되었다. AIML 은 자가 수정 인풋에 의존하고 있고, 그래서 여러분이 모든 가능한 변형에 대해 알고 있지 못하다면, 여러분은 새로운 규칙을 작성할 수 없다. 그리고 간단한 AIML 와일드카드로, 여러분이 원하지 않은 잘못된 긍정적인 결과를 얻기 쉽고 보다 안목있는 패턴을 작성하는데 어려움을 가지게 된다.

여전히, A.L.I.C.E.는 최고의 챗봇이며, 2010 Loebner 컴피티션에서 2 등을 차지하였다.

Façade (2005)

Façade 는 10 전에 개발되었는데, 개발자 Andrew Stern 과 Michael Mateas 는 AIML 보다 더 잘 만들 수 있다고 확신하였다고 한다. 적어도 결혼과 관련된 문제를 겪게 되는 Trip 과 Grace 와 상호작용하여 자연어를 이용하는 스토리에는 자신이 있었다.

Façade 의 자연어 처리는 CLIPS 전문 시스템 언어를 확장하여 제스, 자바를 이용하여 시작하였다. 제스를 이용하면 여러분은 전제조건 매치를 진행할 때 촉진될 수 있는 규칙을 가질 수 있게 되며 철회하거나 공표하는 일을 쉽게 할 수 있다. *Façade* 는 이 템플렛 스크립트 컴파일러를 최우선으로 구성하고 자연어 처리 규칙을 그 다음 작성한다. 그리고 이것을 제스를 통해 컴파일 한다.

이들은 800 개의 템플릿 규칙을 작성하였고, 이것은 6800 개의 제스 규칙으로 옮겨졌는데 자연어 처리를 위해서는 매우 적은 규모에 불과하다. 그러나 이후에, *Façade* 는 더 이상의 진전을 보이지 않았다.

담론 방법 Discourse Acts

Façade 는 한번에 짧은 한 문장을 입력할 수 있도록 제한되었다. *Façade* 가 AIML 을 뛰어 넘는다는 주장은 다음의 문장으로 알 수 있다 : *우리는 반응을 유발하는 텍스트를 구성하지 않는다. 우리는 담론 방법을 그릴 뿐이다. Façade* 는 입력된 말을 완전히 이해하려고 노력하지 않았다. 대신에, 패턴 매치된 것을 *50 개의 담론 방법에* 담았다.

동의, 비동의, 비판과 추파 등을 포함한다. 그래서 그레이스가 진실을 말하지 않았다(*Grace isn't telling the truth)*는 입력은 그레이스를 비판하는(*criticizing Grace)* 담론법에 맞추어졌다. 간단한 한 개의 문장은 동시에 다양한 담론 방법에 맵핑될 수 있었다.

문장을 담론 방법에 맵핑한 후에, 이들은 문맥적 상황에서 어떤 의미를 가지는 지를 담론법에 의해서 결정될 수 있는 코드를 사용하였다. 어떤 것에 집중해야 하는지를 결정하는 것이 중요했다. 그레이스에게 동의하기(*Agreeing with Grace*)는 그레이스를 행복하게 이끌어 주겠지만 트립은 당신이 그레이스 편을 드는 것에 화가 나게 만든다.

담론법은 입력된 문장을 올바르게 해석할 필요가 없다. *Façade* 는 문맥을 알아차리지 못하고 "어?"라고 말하기 보다는 문장을 잘못 해석하는 것을 오히려 선호한다. 70 페선트는 제대로

파악할 수 있다는 것을 활용하였다. 이야기는 진행되고, 이용자는 그 이상은 모르는 모르는 사람이 될 수도 있다.

Façade 는 AILML 이 실행 의존법을 사용하여, 오서 접근적 방법을 사용하지 않아서 AIML 해석기에 의존하여 매칭 순서를 가지고 응답을 하도록 한 것에 대해 반기를 들었다. 나는 이 주장인 잘못되었다고 생각한다. 나는 AIML 의 접근법이 구체적 실행이라고 생각하고 있으며, AIML 이 간단한 응답에만 국한되어 있다고 생각하지 않는다. *Façade* 규칙은 (제스에서 파생된) 돌출에 의해서 순서가 결정될 수 있다. 돌출은 우선적인 규칙 순위를 가지며 오서 콘트롤러는 좋은 방법이 아니다. 심지어 제스 매뉴얼도 이렇게 사용하는 것을 권하지 않는다.

Façade 는 돌출을 4 가지의 이야기 구조를 만들 수 있도록 만드는데 이용하였다. 첫번째, 한 묶음의 로우 레벨 규칙. 그리고 중재적 방법의 정의. 또한 이들은 인접한 부정적인 단어를 재작성하는 규칙을 만들었는데, 그래서 "나쁘지 않아(not bad)"가 "좋아(good)"가 되도록 했다. 결국 이들은 담론 방법을 맵핑 하는 규칙을 만들어냈다. 이 모든 것은 또한 AIML 에서도 이루어졌을 수 있으나, 단언 하건데 불문명한 규칙이라고 할 수 있다.

대명사를 해결하는 방법은 AIML 스타일로 이루어져서 각 아웃풋에 대명사의 의미를 세팅하도록 했다.

사실 Facts

제스로 작업하기 위해서, 모든 인풋 단어는 작업 메모리에 입력되어진 사실이 된다.

나는 소고기를 좋아한다(*I like beef*)는

사실이 된다 :

position-1-word 는 나(I)이다.
position-2-word 는 좋아한다(like) 이다.
position-3-word 는 소고기(beef) 이다.

제스는 이러한 사실에 반응하여 이 사실들을 사전 조건으로 가질 수 있도록 규칙을 제정하였다. 규칙은 입력 단어를 부가적인 중재 사실에 맵핑되도록 하였고, 더 많은 규칙이 하나 이상의 최종 담론법 사실에 중재적인 사실로 결합될 수 있도록 하였다. 예를 들어, 여기 세가지 모의의 규칙이 있다 :

hello => iGreet
Grace => iCharacter(Grace)
iGreet AND *iCharacter(?x)* => DAGreet(?x)

Hello 를 보는 것은 *iGreet* 사실을 중재하도록 만든다. *Grace* 를 보는 것은 *iCharacter(Grace)*의 중재적 사실을 생성한다. 그리고 만약 여러분이 *iGreet* 를 보고 즉시 *iCharacter* 를 따르게 된다면, 여러분은 캐릭터의 *greeting* 담론법을 마주하게 될 것이다.

패턴 매칭 **Pattern Matching**

Façade 패턴은 AIML 보다 훨씬 강력하다. 이들은 제한적이지 않은 AIML*을 확장시켜서 0 이나 하나 이상의 단어와 매치할 수 있다. 이것은 많은 규칙을 4 개의 행태로 작성되도록 한 것을 막았다. 그리고 *Façade* 의 새로운 패턴 방식은 실질적인 의미에 보다 근접한 매칭을 할 수 있는 오서링 콘텐츠를 비판하였다.

템플릿 언어는 아래의 단어들 사이에 이루어질 수 있는 관계를 지지한다 :

(X Y) – 차례로 단어의 관계를 순서 짓게 함(**AND**)

(X | Y) – 다른 단어의 존재 (**OR**)

([X]) – X 는 아무 의미가 없을 수 있다 (**OPTIONAL**)

AIML 은 AND 관계에 토대를 둔다. *Façade* 의 **OR** 기능은 혁신적인 향상이었다. Façade 는 중재적인 입장의 사실에 여러 단어를 맵핑할 수 있다. AIML 은 단어의 패턴에 맞추어 매칭한다. *Façade* 는 일련의 단어 세트에 패턴을 매칭시킨다. 이를 통해 의미를 보다 매칭하기 쉽게 해 주었다.

좀더 쉽게 이해하기 위해서, 이들은 통사론의 설탕 sugar 을 활용하였다 :

(tor X Y) which means *((* X *) | (* Y *))*

X 나 Y 가 문장의 어디에서 발생하던지 인풋 문장에서 찾도록 한다는 것을 의미한다.

또 다른 설탕 sugar 은 다음과 같다 :

(tand X Y) which meant ((* X * Y*) | (* Y * X *))

X 와 Y 가 어떤 특정한 차례에 있다는 것을 의미한다.

그리고 세번째는 *(tnot X)* 로, X 를 찾지 않는다는 것을 의미한다. 이것 역시 중요한 부가이다. 단어(not 과 같이 특정한 부정적인 단어)의 부재는 종종 존재하는 것만큼 중요하기 때문이다.

그러므로 *Façade* 는 다음의 세가지 규칙을 활용하여 칭찬(*praise)*의 중재적 사실을 규정하였다 :

```
(defrule positional_Is
(template (tor am are is seem seems sound sounds look looks))
=> (assert (iIs ?startpos ?endpos)))
```

위의 규칙은 만약 여러분이 ~하다는 것과 가까운 동사를 보게 된다면, 작업 메모리에 넣고 새로운 사실의 *iIs* 를 찾을 수 있도록 해 준다는 것을 의미한다.

```
(defrule positional_PersonPositiveDesc
(template (tor buddy comrade confidant friend genius go-getter pal sweetheart))
=> (assert (iPersonPositiveDesc ?startpos ?endpos)))

(defrule Praise_you_are_PersonPositive
(template ( {iPersonPositiveDesc} | (you [{iIs}] [a | my] {iPersonPositiveDesc} *)))
=> (assert (iPraise)))
```

Façade 는 여기서 두가지 방식으로 잘못되었다. 첫째, 이들은 전체 인풋을 매치하였고, 이 때문에 여전히 패턴에 and/or trailing *을 도출하게 되었다. 둘째, 템플릿은 LISP 와 비슷하고

읽거나 쓰기에 적합하지 않다. 세번째 규칙의 *{ iPersonPositiveDesc }*은 일반적인 단어로써라기 보다는 중재적인 사실로 단어를 체크한다는 것을 의미한다. 이것은 규칙을 읽는 것을 애매하게 만드는 엉뚱한 생각이다.

워드 넷 WordNet

Façade 는 스텀 워드넷 단어를 사용할 수 있게 하였고, 워드넷 확장 단어 세트를 허용하였다. 이것을 어떻게 스크립트 할 것인가는 설명하지 않았고, 논리적으로 올바른 상황에서 이것을 사용하는 예시를 제시하지 않았다. 아이디어는 명백했고 유용성이 있었으나, 두 가지 이유로 적절하지 않았다.

첫째, 스터밍은 go, going, goes 를 같은 뜻으로 처리하는데 유용하다. 그러나 스터밍은 비규칙적인 동사의 활용 (*be am was been)*을 처리하지 못할 뿐 아니라 실제로는 쓸모없는 결론을 도출해 낸다. 그래서 여러분은 스텀을 알아야만 한다. 커뮤니티(community)의 스텀을 생각해 보자. *Communiti* 이라는 것을 알겠는가?

둘째, 이들의 워드넷 확장은 일련의 동의어이다. 그러나 워드넷은 완벽한 존재론적인 것이어서, 여러분은 콜리가 현재 존재하고 있는 포유류인 개라는 것을 알 수 있다. *Façade* 는 이것을 전혀 사용하지 않도록 했다.

복잡한 계산 **Complex computation**

Façade 는 제스를 이용한다. 그래서 자바로 스크립트 언어를 쓸 수 있도록 해 주었다. 그러나 제스는 LIPS 스타일의 접두 부호 표현을 이용한다. 많은 프로그래머들이 이것을 싫어한다. 제스는 SQL 등을 이용하여 사실을 명확히 하고 사실의 쿼리를 지지해 준다. 이것은 간단한 관계에는 적합하지만 복잡하게 얽혀있는 것에는 효율적이지 않다. 그러나 이들은 이것을 전혀 고려하지 않았다.

요약 **Summary**

Façade 의 주요 자연어 처리는 중재적 사실을 매칭하고 담론법에 기인하고 있어서, OR 이 일련의 단어 세트와 매칭되고 NOT 은 배재되도록 한 명쾌한 규칙을 가지고 있다. 그러나 *Façade* 의 자연어 처리에 있어서 나의 가장 큰 불만은 여전히 너무 장황하고 오서하기에 너무 어렵다는 것이다.

ChatScript (2010)

나는 CHAT-L 이라고 불리는 Avatar Reality 를 위한 새로운 챗봇 언어를 만들었다. 그리고 이후에 ChatScript 로 이를 재조정했는데, 오픈 소스 챗봇 엔진에 관한 것이었다. *Façade* 의 자연어 처리 능력을 좀더 일반적이고 더 단순하게 만들어 줄 수 있는 역할을 하는 것이었다.

Suzette 는 내가 작성한 챗봇이다. 그녀는 2009 년 Chatterbox Challenge 에서 최고 신인으로 수상하며 데뷔하였고, 2010 년 Loebner Competition (Turing Test)에서 매우 인기있었다.

AIML 은 간단한 단어 패턴 매칭기였다. 담론법에 맞추어진 *Façade* 패턴 매치는 의미의 제한적인 형태에 국한되었다. ChatScript 는 패턴 매치를 일반적인 의미에 맞출 수 있도록 하는데 목적이 있었다.

그러므로 균형점을 찾고 단어의 세트에 주목하며 기본적인 어휘법에 관심을 두고 있다. 또한 검색이 가능한 형태로 데이터를 가용할 수 있도록 하였다.

챗 스크립트 패턴 ChatScript Patterns

ChatScript 는 간단한 시각적 구문론을 이용한다. XML 이나 LISP 에서 차용해 오지 않았다. 그리고 ChatScript 는 입력되는 모든 단어를 매치시켜야 겠다는 욕구를 버렸다. 이로 인해 4 규칙 신드롬과 패턴의 끝에 와일드카드 사용의 과도함을 피했다. 여기에 간단한 ChatScript 규칙이 있다 :

s: (I love meat) Do you really?

이 규칙은 (*s:*) 문장을 테스트하고 있다. 패턴은 매우 순차적이다. *I, love, meat* 가 입력된 문장 내에서 직접적으로 순차적으로 구성되어 있는가 아닌가를 살펴 본다. 아웃풋은 이러한 차례 검토 뒤에 도출된다.

규칙 형태 Rule Types

s: 와 ?: 은 평서문과 의문문을 각각 연결시켜 주는 것을 의미하는 규칙 형태이다. Façade 와 AIML 은 문장 부호를 고려하지 않고, 평서문과 의문문을 구별하지 않는다. ChatScript 는 인풋이 물음표를 가지고 있는지, 또는 문장의 구조가 의문문 형태인가를 검토한다. 규칙은 평서문, 의문문에 따라 속박될 수 있으나 *u:*는 둘 다를 포괄하는데 사용된다.

여러분은 대화가 계속해서 이어 나가도록 스크립트를 작성할 수 있다(가령, *a:*; *b:*; 와 같이 말이다). 이때 인풋은 즉시 성공적인 ChatScript 규칙 아웃풋을 따라야 한다. 이점에서 AIML 의 처리법 보다 훨씬 더 명쾌하다.

s: (I like spinach) Are you a fan of the Popeye cartoons?

a: (yes) I used to watch him as a child. Did you lust after Olive Oyl?

b: (no) Me neither. She was too skinny.

b: (yes) You probably like skinny models.

a: (no) What cartoons do you watch?

b: (none) You lead a deprived life.

b: (Mickey Mouse) The Disney icon.

s : (나는 시금치를 좋아한다) 너 뽀빠이 만화를 좋아하니?

a: (응) 어릴 때 보곤했었어. 올리브를 좋아했었니?

b: (아니) 나도 아니야. 걔는 너무 말랐어.

b: (응) 넌 아마도 마른 모델 같은 여자를 좋아하나봐.

a: (아니) 너는 어떤 만화를 보는데?

b: (절대 아니야) 너 너무 재미없게 산다.

b: (마이키 마우스) 디즈니 아이콘

컨셉 Concepts

ChatScript 는 컨셉이라고 일컫어 지는 일련의 단어를 지원한다. 이 컨셉은 동의어나 연계된 단어를 나타낼 수 있으며 또는 자연스러운 단어의 순서를 나타내기도 한다 :

컨셉 : ~고기 (베이컨 햄 소고기 살 송아지고기 양고기 닭고기 돼지고기 스테이크 소 돼지)

위의 ~고기는 *고기와 가까운 관계에* 있는 것들을 의미한다. Façade 의 {iMeat}와 동등하다고 볼 수 있지만 읽고 쓰기에 더 쉽다. 여러분은 모든 종류의 고기와 관련 된 것에 응답하는 규칙을 만들 수 있다.

s: (I love ~meat) Do you really? I am a vegan.

s: (나는 ~고기를 좋아한다) 정말? 나는 채식주의자인데.

순서가 정해진 컨셉은 아래에 설명된 바와 같이 포커 게임에서의 핸드오더링 시작을 보여준다.

컨셉 : ~포커핸드 (“로얄 플러시” “스트레이트 플러시” “포 어브 카인드” “풀 하우스”)

패턴 :

?: (which * better * ~pokerhand * or * ~pokerhand) ...

은 풀 하우스와 로열 플러시 중에 어느 것이 더 좋은가와 같은 질문을 알아내고 시스템은 올바른 답을 제공하기 위하여 순서가 정해진 컨셉을 이용할 수 있도록 기능을 하게 된다.

여러분은 컨셉 내에 또 컨셉을 둘 수 있는데, 이렇게 해도 괜찮다 :

컨셉 : ~음식 (~고기 ~라자냐 디저트 ~채소 ~과일)

concept: ~food (~meat ~dessert lasagna ~vegetables ~fruit)

계급적인 유산은 패턴 일반화의 수단으로써, 효율적으로 선택된 규칙 매커니즘으로도 중요하다. 컨셉은 동사, 명사, 형용사, 부사 등의 모든 통사를 만드는데 이용될 수 있으며 일반적인 의미나 자연스러운 의미에 매칭될 수 있도록 해 준다.

규범적 형태 Canonical Form

여러분이 패턴에 규범적 형태를 사용한다면, Façade 의 스터밍 대신에, ChatScript 는 오리지널 형태의 인풋 단어 뿐만 아니라 규범적 형태까지 동시에 매치시킨다.

명사의 복수형은 단수형과 같게 치부하고, 소유격 접미사 '와 's 는 단어의 's 로 변형시킨다. 동사는 부정사로 바꾼다. 형용사와 부사는 기본 형태로 귀속시킨다. *a an the some these those that* 과 같은 한정사는 *a* 로 대표된다. *two thousand and twenty one* 과 같은 숫자는 디지털 형태로 바꾸고 유동적인 점수는 정확하게 가치를 가지고 있을 경우에 정수로 나타나게 한다. *me*, *my*, *myself*, *mine* 과 같은 소유격은 *I* 형태의 주어에 결부시키고, *whom, whomever, whoever* 는 *who* 로 *anyone somebody anybody* 는 *someone* 이 된다.

Façade 는 현재형으로만 작업이 가능한 읽기 어려운 다음의 규칙으로 쓰여졌다 :

(defrule positional_Is

(template (tor am are is seem seems sound sounds look looks))

=> (assert (iIs ?startpos ?endpos)))

아래 챗 스크립트의 간단한 컨셉은 모든 시제와 리스트 작성된 동사의 활용형을 포함한다 :

concept: ~be (be seem sound look)

만약 여러분이 정식 형태의 단어를 이용하지 않거나 인용한다면, 시스템은 이것을 패턴에서 활용해 왔던 것으로 제한할 것이다.

u: (I 'like you) 이것은 *I like you* 와 매치되며, *I liked you* 로 매치되지 않는다.

s: (I was) 이것은 *I was* 와 *Me was* 에 매치되며, *I am* 으로 매치되지 않는다.

워드넷 온톨로지

Façade 는 워드넷의 주요한 가치를 활용하는데 실패했다. 챗스크립트에서, 워드넷 온톨로지는 단어를 네이밍 하거나 여러분이 원하는 의미를 활용함으로써 작동된다.

concept: ~buildings (shelter~1 living_accomodations~1 building~3)

컨셉 *~buildings* 은 워드넷 사전에서 찾을 수 있는 760 개의 일반적이거나 구체적인 빌딩 단어를 나타낸다. 관련된 어떤 단어라도 찾아 낼 수 있다 : shelter 의 정의 1, accommodations 의 정의 1 또는 building 의 정의 3 모두 워드넷의 온톨로지에 있다.

패턴 오페레이터 Pattern Operators

AND 단어 관계는 ( )을 이용하거나 인용 문구를 이용하여 행해진다.

여러분은 컨셉 또는 []을 이용하여 **OR** 선택을 할 수 있다 :

s: (I ~love [bacon ham steak pork (fried egg) "green egg"]) Me, too.

유사한 챗 스크립트는 { }을 이용하는 **OPTIONAL** 을 지원한다 :

u: (I ~go to { a } store) What store?

단어의 부재를 나타내는 **NOT** 은 !을 이용하여 나타내어지며 현재 매치되고 있는 곳 외에는 어디에서도 찾을 수 없다는 것을 의미한다 :

u: (![not never rarely] I * ~ingest * ~meat) You eat meat.

u: (!~negativeWords I * ~like * ~meat) You like meat.

그리고 챗스크립트는 << >>을 이용하여 **UNORDERED** 단어를 찾아내는데, Façade 에서 이용한 것 보다 더 간단한 문법이다. 어떤 순서로 단어를 찾는다는 것은 아래와 같은 패턴을 작성하는 것을 더 쉽게 만들어 준다 :

u: (<< you ~like ~meat >>) I do like meat.

to cover *Do you like meat?* and *Is bacon something you desire?* and *Ham is something you like.*

만약 여러분이 문장의 시작과 끝이 어디인지를 알아야 할 필요가 있다면, 여러분은 <나 >을 이용할 수 있다. 그러나 일반적으로 그렇게 중요하지 않다.

s: (< I know) You say you know.

?: (what is love >) Love is a wonderful thing.

와일드 카드 Wildcards

챗스크립트는 와일드카드 콜렉션을 가지고 있다. 제한되지 않은 wildcard *은 0 이나 더 많은 단어를 의미한다. wildcard *~2 의 범위는 0-2 단어를 의미하고, *~3 은 0-3 의 단어가 될 것이다. 와일드카드의 범위는 관련되지 않은 것을 매칭시키기 위해 콘트롤을 잃어버리지 않도록 해 주기 때문에 유용하다. 구체적인 wildcards *1, *2, *3 …들은 많은 단어를 필요로 한다. 그리고 심지어는 거꾸로 가는 *-2 와 같은 와일드카드도 있는데, 이것은 현재 위치에서 2 개 이전의 단어를 찾게 해 준다.

아래의 범주가 있는 와일드카드 패턴을 검토해 보자 :

s: (I *~2 ~love *~2 ~meat)

이것은 *I love ham* 과 *I almost really love completely uncooked steak* 를 가능하게 해 주지만 *I love you and hate bacon* 은 수용하지 않을 것이다.

범주가 있는 와일드카드와 컨셉을 매치시키는 능력은 경제성과 정확성에서 기인한다. 이 간단한 패턴과 관계 있는 수 천가지의 모욕적인 말을 찾을 수 있다 :

(!~negativewords you *~2 ~negativeAffect) Why are you insulting me?

~negativeAffect 세트의 4000 개의 단어를 활용하여, 위의 예는 *you dork* 나 *you have an ugly face* 에 응답하도록 하겠지만, *you aren't stupid* 에는 반응하지 않는다 (~negative 단어는 *not, never, rarely, hardly* 를 포함한다). 게다가 *you can always tell the poor from the merely stupid* 에도 역시 반응하지 않을 것이다.

AIML 은 자동으로 와일드카드 텍스트를 엮게 되기 때문에 여러분은 결과물을 계상하는 동안 이를 수습하거나 개선할 수 있다. Façade 는 어떠한 텍스트 결과물을 발생시키지 않기 때문에 어떠한 것도 캡쳐하지 못한다. 챗스크립트는 여러분이 접두사 an 을 이용하여 매치된 단어를 캡쳐할 수 있게 해 준다. 여러분이 *I really adore raw chicken in the morning with soggy beans* 라고 말한다면, 이에 따르는 패턴은 구체적인 고기와 채소의 이름을 인풋 정보에서 찾아 이것을 아웃풋으로 내보내도록 캡쳐한다.

s: (I * ~like * _~meat * and * _~vegetable) I hate _0 and _1.

변수 Variables

챗스크립트의 이용자 변수는 $로 시작하여 텍스트를 포함한다. 이들은 아웃풋을 위해서 뿐만 아니라 패턴 내에서도 활용될 수 있다.

s: (I be * ~genderWords) refine()

a: (~maleGenderWords) $gender = male

a: (~femaleGenderWords) $gender = female

?: ($gender << what be I gender >>) You are $gender.

s: ($gender=male I ~like ~male) I prefer women.

첫번째 규칙은 이용자의 성별(소녀, 왕, 경찰)에 대한 것을 찾는 것으로 시작된다. 그리고 즉시 연결되는 상황을 찾을 때까지 테스트를 계속하여 정제시키고, 이용자의 성별과 관계없이 저장된다. 두번째 규칙은 이용자의 성별이 정의되고 그들이 누구라는 것을 확인한 다음 응답을 제시한다. 세번째 규칙은 이용자의 성별이 남성일 경우에만 매치시키고 나머지가 매치된다.

챗스크립트는 또한 %rand (무작위 가치), %length(문장 길이), %month(현재 달)과 %tense 를 포함하는 시스템 변인을 가지고 있다. 여러분은 동사원형 컨셉을 이용할 수 있으나 여전히 아래와 같은 과거 시제에 대한 be 동사를 요청할 수 있다 :

u: (%tense=PAST I ~like you) What happened?

Fact Triples

챗스크립트는 *fact triples* 를 지원하고 컨셉과 다른 것들을 내부적으로 이것을 이용하여 나타낸다. 여러분은 여러분자신의 테이블과 데이터를 작성할 수 있고 발전시킬 수 있다. 그러나 여러분은 일반적인 형태 세트나 유저에게 익숙한 그래프, 각각의 구성원에 대한 정보 저장 등을 원할 것이다. 테이블 정의는 테이블의 열의 이름을 짓고, 각 테이블 라인을 처리하는 코드를 만들어 테이클을 채우게 되는 것을 의미한다. 예:

table: ~malebooks(^author ^title ^copyright)

createfact(^author member ~author)

createfact(^title member ~book)

createfact(^title exemplar ^author)

if (^copyright != "*") { createfact(^copyright birth ^title) }

DATA:

"Orson Scott Card" "Ender's Game" 1985

"Daniel Defoe" "Robinson Crusoe" *

적절한 규칙으로, 여러분은 이용자가 제목이나 저자를 입력했을 때를 알 수 있으며, 어떤 것이 어떤 장르에 연결되어 있는지, 책이 언제 출판되었는지를 알 수 있다. 예:

u: (who * [write author pen] * _~book objectquery(_0 exemplar)) @0object

규칙은 저자를 알 경우에만 책의 이름에 반응한다. 오브젝트 쿼리(사전 정의된 기능)는 _0 의 콘텐츠가 어디에 있는지, 주어와 동사가 격식에 맞는지를 찾는다. 만약 실패한다면, 패턴도 실패한다. 쿼리는 항상 찾아낸 사실을 특정한 장소(이 쿼리를 위한 디폴트는 @0 세트로 되어 있다)에 저장된다. 아웃풋은 @0 의 찾아낸 사실 중 하나를 오브젝트 필드, 즉 Daniel Defoe 를 산출한다. ("Robinson Crusoe"는 Daniel Defoe"를 의미한다)

기능 Functions

여러분은 패턴 기능과 아웃풋 기능을 정의할 수 있다. *what-is* 형태의 질문을 찾을 수 있는 WHAT_IS 일반 형태를 만들 수 있는데, 아래와 같이 활용할 수 있다 :

u: (WHAT_IS (LINUX))

이것은 반복되는 인풋의 수정을 수반하지 않는다. 이름만으로 기능이 어떤 것인지를 대충 알 수는 있겠지만, 직접적인 패턴으로 모든 형태의 질문을 커버할 수 있는지 아닌지를 알아보기 위해서 기능을 각기 검토해야 한다.

요약 Summary

챗스크립트 패턴은 간결하고 강력한데다 이해하기 쉽고 읽고 쓰기 쉽다. 여러분이 불명확한 방식으로 밀고 나가지만 않는다면, 여러분은 즉시 입력되는 정보가 여러분의 패턴에 매치되고 있는지 아닌지를 알 수 있다. 그래서 패턴은 디버그 될 수 있고, 즉각성을 유지할 수 있다. 여러분은 심지어 특수한 코멘트를 규칙 앞에 부가할 수 있는데, 이를 통해 인풋 문장의 프로토타입을 제공할 수 있다. 이것은 어떤 규칙이 매치되어야만 하는지, 시스템이 회귀 테스트의 부분으로 기능을 하는지 입증하도록 하는지를 기록한다.

복잡한 계산 Complex Computation

챗스크립트는 직접적으로 아웃풋 텍스트를 C 스타일 스크립트 언어와 섞는다. 그래서 아웃풋 뿐만 아니라 패턴에서도 이용될 수 있다. 챗스크립트는 사실을 규정하고 이에 맞는 쿼리를 짜도록 지원해 주며, 늘일 수 있는 그래프 매커니즘을 가지고 있다. 그래서 이것은 직접적으로 온톨로지 계층을 돌아다니도록 해 주고, 유저 친화적 노드와 링크 맵을 그릴 수 있게 해 준다. (예를 들어, 일방통행로의 도시 그리드에서의 여행을 묘사하는 것)

챗스크립트는 새로운 입력을 합성하는 AIML 의 특성을 발휘할 수 있고 인풋 스트림을 통해 이를 처리할 수 있는데, Suzette 는 분석할 수 있는 대명사를 가지고 문장을 다시 쓰기 위해 이것을 이용한다.

실행 Implementation

데이터 재현 Data Representation

실행의 초기 원칙은 사전으로, 워드 넷에서 기인한다. 직접적인 인덱싱을 제쳐놓고, 단어는 샅샅이 해부되어 접근된다. 각각의 단어는 단어의 형태 (시제, 비교 수준, 복수 등)와 말의 부분처럼 고정적인 정보를 담고 있고, 관련있는 단어와 연계된다 (예를 들어, 불규칙 동사의 활용형은 순환으로 이어지게 된다).

이 정보는 정공법으로 정하기에 필수적이다. 성별과 같은 단어나 인간, 장소 또는 시간에 관한 것 같은 단어 역시 여기에 저장된다. 워드넷 합성어도 저장되어 유사어가 될 수 있는 단어 리스트가 포함된다.

단어 목록은 추론에 이용되는 다양한 "여기에 속하는 것"으로 마크될 수 있으며 현 문장에서 단어가 어디에 있는가에 대한 목록까지를 포함한다. 패턴 속에서 단어를 마주치게 되었을 때, 여러분은 빨리 훑어 볼 수 있고, 즉각적으로 단어가 위치해 있는 곳을 통해 인풋에서 모든 정보를 찾을 수 있게 된다는 것을 의미한다. 컨셉 이름, 스트링과 특수 용어 뿐만 아니라 일반적인 단어에 적용된다. 또한 변수는 사전에 저장된다.

다른 근본적인 시스템은 *팩트*이다. 팩트는 *주어, 동사, 목적어의* 세 분야이다. 팩트 분야의 가치는 온톨로지 포인터나 팩트 포인트로써 생성된다. 온톨로지 포인터는 사전 단어 색인에 대한 것을 포함하고 있고, (워드넷의 단어 뜻이나 구술 언어의 한 부분으로의 단어 뜻 등의) 의미 인덱스를 구성한다. 0 의 의미 인덱스는 단어의 모든 뜻을 말하는 것이고, *help* 또는 *help~0* 으로 쓰여질 수 있다. 필드가 팩트 포인터라면, 그 값은 팩트 배열에서의 인덱스이다.

각각의 사전 목록과 팩트는 제 위치에서 활용하는 팩트 리시트를 가지고 있다. 그래서 "bill"이라는 단어는 "bill"이 주어로 사용되는 모든 팩트에 대한 리스트와 "bill"이 동사로 사용될 때의 리스트 및 목적어로 사용될 때의 리스트도 가지게 된다.

워드넷 온톨로지는 is 동사를 활용하는 사실로 구성되어 있다. 그래서 워드넷에서 읽는 것은 (*Persian~3 is cat~1)*과 *(cat~1 is animal~1)*과 *(animal~1 is entity~1)*이 유사한 팩트로 구성될 것이다.

컨셉은 동사 *member* 를 이용한다, 그래서 *concept: ~birdrelated (bird~1 "bird house")*은 (*bird~1 member ~birdrelated)*과 *(bird_house member ~birdrelated)*의 사실을 의미한다.

데이터 처리 Data Processing

표시 Tokenization

토크나이저는 가장 먼저 문장의 연속성으로 이용자의 인풋을 나눈다. 그리고 자동 스펠링 체크를 수행하고, 숙어를 치환하며, 적당한 이름과 숫자를 맞추어 본다. 구두법을 맞추어 보기 위해 분해를 하지만, 문장이 느낌표나 물음표를 가지고 있는지 아닌지를 파악한다. 스크립트 콘트롤은 문장의 구성에 대해서 검사할 것이고, 이를 통해 물음표가 없는 질문은 적적히 표시되어 질 것이다.

단어와 컨셉 마킹하기 Marking Words and Concepts

결과는 각각의 단어에의 대안을 발생하게 하는 정공법을 통해 운영된다. *Façade* 가 최초이자 중재하고 있다고 생각하고 있는 것을 표시하기 위해 이 두가지의 인풋 정보가 사용된다. ChatScript 에서는 이것이 팩트가 아닐지라도, 이들은 문장에 있는 위치에 따라서 사전 목록에 있는 주석이기는 하다.

(일반 및 표준) 각 단어를 위해, 시스템은 주어로써 이들의 팩트 계급을 구성한다. 동사 *member* 와 *is* 를 찾아서 이것을 목적어와 바꿀 수 있게 해 주어, 처리를 지속시킨다. 단어와 대체 단어에 적용시키면, 원래 단어가 있었던 위치를 사전에 마크 시키는 방식이 된다. 사전에 있는 근접한 단어의 시퀀스에도 똑같이 적용한다.

패턴 매칭 Pattern Matching

패턴 매칭은 지금 시작할 수 있다. 매칭 비용은 패턴 내 상징의 수에 비례한다 (쿼리 함수를 사용하는 패턴은 제외함). 인풋이 *I really love pigeons* 라고 상정하면, 이것은 아래와 같이 매치되도록 규정된다 :

s: (I *~2 ~like * _~birdrelated) I like '_0 , too.

문장에만 적합하도록 되어 있는 이 규칙은 *I* 가 문장 내 어디에서 발견되는 가를 묻고 있다. *I* 는 문자 그대로 I 일수도 있고, *me myself mine my* 과 같은 "격"의 변형물 일수도 있다. 우리는 단지 사전에서 *I* 만 찾아 보고 제 위치에 마크되어 있었는지를 알아본다. 마크가 되어 있다는 것을 확인하고, 우리는 가장 최근의 매치된 위치를 추적하게 된다. 만약 거기에 없다면, 이 패턴은 잘못된 것이다.

다음은 제한된 범위의 와일드카드 이다. 이것은 2 개의 차순위 단어를 포함한다. 그래서 우리는 기록하고 다음으로 넘어간다. *~like* 를 살펴 보면, 우리는 사전에서 이것을 찾고 *I* 보다는 나중에 (from *love*)가 제대로 마크되어 있는지를 알아본다. 우리는 새로운 매치 포지션을 찾아본다.

우리는 다음 * wildcard 를 기록하고 an _을 찾기 위해 넘어 간다. 이것은 다음의 매치가 있다는 것을 기억하도록 하는 표시이며, 이를 위해 깃발을 꽂아 둔다. 그리고 나서 우리는 사전에서 *~birdrelated* 를 찾는다. 이것은 *pigeons* 으로부터 마크되며, 문장의 뒷 부분에 적절히 배치된다. _ flag 때문에, 찾아진 위치를 기억하고, 특수 매치 변수를 실제 문장에 대입하여 *pigeons* 를 *pigeon* 에 대입되도록 한다.

패턴과 매칭을 끝냈기 때문에, 우리는 아웃풋을 실행하고 I *like pigeons, too* 를 쓴다. '_0 은 원래 형태에서 0 번째로 기억되도록 회수한다는 것을 의미한다.

아웃풋이 발생되었기 때문에, 시스템은 다음에 무엇을 할 것인지를 결정하기 위해서 콘트롤 스크립트로 돌아간다. 아웃풋이 발생되지 않았다면, 다음 규칙으로 이동할 것이다.

패턴 매칭은 제한된 역추적을 할 수 있다. 시스템이 궁극적으로는 실패한 최초의 매치를 찾는다면, 시스템은 첫번째 패턴 단어의 매치를 대체시킬 것이고, 실패 한다면, 다시 시도할 것이다. 예를 들어, 테스트 된 규칙이 (*I *~ ~like * _ ~birdrelated)*이고 인풋이 *I hate dogs but I really love pigeons* 이었다면, *I*는 *I hate* 에 매치되고, *~like* 는 충분히 쉽게 찾아 지지 않았을 때, 시스템은 *I* 를 고정시키지 않을 것이고 *I really* 의 *I* 와 다시 매치를 시작하게 될 것이다. 전체를 역추적하는 것보다 이렇게 하는 것이 더 간단하고 더 효율적이다.

모두 함께 모으기 : 토픽 Putting it all together: Topics

시스템은 모든 규칙을 실행하지는 않는다. 오써는 토픽에 맞추어 규칙의 콜렉션을 조직한다. 토픽은 또한 일련의 관련된 키워드를 가지고 있다. 이것들은 인풋과 가장 가까이 관련된 토픽을 찾는데 이용된다.

topic: ~RELIGION (~fake_religious_sects ~godnames ~religious_buildings ~religious_leaders ~religious_sects ~worship_verbs sacrament sacred sacrilege saint salvation sanctity sanctuary scripture sect sectarian secular secularism secularist seeker seraph seraphic seraphim soul spiritual spirituality "supreme being" tenet theocracy theology tithe pray venerate worship)

토픽은 부가적인 규칙 형태를 소개한다. 이용자 인풋 (*s: ?: u:*)을 위한 응답기 뿐만 아니라, (*t:*)를 제공할 수 있는 수를 낼 수 있는 토픽도 가지고 있다. 우세를 확보하기 위한 수는 이어지는 이야기를 만들어 낸다. 이용자가 수동적으로 "OK" 나 "right"와 같이 말함으로써 대화가 지속되도록 조종할 수 있다. 그러나 이용자가 질문을 받으면, 시스템은 직접적으로 응답을 하거나 어떠한 꾀를 내는 방법을 쓰거나 하는 응답기를 활용할 수 있다.

토픽은 초반에 우세를 장악하기 위한 말 모드(meaning *t:* lines fire)로 실행되거나 (meaning *s: ?:* and *u:* fire)의 응답기 모드로 실행된다. 규칙은 여러분이 실행되기를 원하는 순서대로 자리를 잡는다. 우세를 지니기 위해서는 순서가 이야기를 말한다. 응답 모드에서는 규칙이 대개 구체적인 것에 덜 구체적인 것으로 순서가 되어 있고, 하부 주제로 확장될 수 있도록 된다.

t: An atheist is a man who has no invisible means of support. 무신론자는 어떠한 지원의 수단도 가지고 있지 않은 사람이다.

a: (<< what mean >>) It means God (invisible and supporter of life) doesn't exist for an atheist. 이것은 신(보이지 않지만 삶을 지원해 주는 자)이 무신론자에게는 존재하지 않는다는 것을 의미한다.

t: Do you have any religious beliefs? 종교적인 믿음이 있습니까?

a: (~no) How about ethical systems that dictate your behavior instead? 아니면 여러분의 행동에 영향을 주는 종교 시스템이 있습니까?

a: (~yes) What religion are you? 종교가 무엇입니까?

b: (_~religious_sects) Were you born _0 or did you convert? 모태 신앙입니까 개종하셨습니까?

t: RELIGION () Religion is for people who are afraid to stand on their own. 종교는 스스로 서기를 두려워하는 사람들을 위한 것이다.

?: (<< [where what why how who] [do be] God >>)

[There is no God.] [신은 없다]

[A God who is everywhere simultaneously and not visible is nowhere also.] [신은 어디에나 함께 계시며, 보이지 않을 뿐이다]

?: (be you *~2 follower *~2 [~fake_religious_sect ~religious_sects ~religious_leaders]) reuse(RELIGION)

u: (<< [I you] ~religion >>) You want to talk about religion? 종교에 대해서 말하고 싶습니까?

아웃풋 함수 재사용은 더블 오써링 정보를 피하기 위해 중요하다. 이것은 아웃풋 섹션으로 "건너 가서" 또 다른 규칙이 된다.

기본적으로, 시스템은 그 자체를 반복하는 것을 피하고 있으며, 매치된 활용 규칙을 마크하는 것을 피한다. 모든 규칙은 부착된 라벨이 있으며 모든 갬빗은 필수 패턴을 가질 수 있다. 그래서 여러분이 규칙을 아웃풋 데이터와 공유하도록 만들거나 그렇게 하기를 원한다면 시스템은 갬빗을 조건부로 활용한다.

토픽 시스템은 챗스크립트가 효율적으로 관련된 규칙을 찾을 수 있도록 찾는 걸 돕는다. 이용자 문장 핵심 단어는 가장 가까운 매칭 토픽을 찾는 것을 도와준다. 매칭 응답기를 찾을 수 있다면, 이것은 토픽으로 전환되어 응답한다. 그렇지 않다면, 다른 매칭 토픽으로 시도된다.

결국, 적절한 응답기를 찾을 수 없다면, 가장 관련있는 토픽으로 되돌아 가서 갬빗을 없애 버릴 수 있다. 그래서 만약 여러분이 *Were Romans mean to Christians?*라고 말한다면, 이것은 *An atheist is a man with no invisible means of support* 을 돌아갈 수 있다. 적어도 이것은 적당한 토픽으로 대화를 시작할 것이라는 것을 의미한다.

토픽은 논리적으로 규칙을 함께 엮어 두는 것을 용이하게 한다. 토픽 핵심 단어는 저작자가 독립 대화 스크립트를 작성할 수 있다는 것을 의미한다. 그리고 시스템은 자동으로 이것을 찾아서 이에 합당한 응답을 줄 것이다. 콘트롤은 AIML 처럼 최초의 시퀀스 단어에 기반을 두고 있지 않는다. 대신에 문장과 혈족 관계에 기반을 둔다. 새로운 토픽이 어떤 다른 토픽의 규칙과 겹치는 규칙을 가지고 있느냐는 것은 문제가 아니다. 여러분은 *~burial_customs* 과 *~death* 라는 규칙을 모두 가질 수 있다. *I don't believe in death* 라는 인풋 문장은 여러분이 어떤 토픽으로든 갈 수 있게 해 줄 것이고, 이것으로 오케이다.

Façade 의 특징 대신에, 토픽은 ChatScript 를 시각적 규칙 순서로 만든다. 여러분은 분리된 토픽을 정의하고 각각을 부르는 것만으로 규칙의 묶음을 분리할 수 있다.

콘트롤 스크립트 Control Script

ChatScript 가 할 수 있는 것과 *Façade* 가 할 필요가 없는 것은 반영이다 – 시스템은 이 자체로 이들의 능력을 이용할 수 있다. 결정을 따르고 스스로의 방식으로 패턴 매치를 할 수 있다.

Suzette 는 이용자가 직접적으로 질문애 ㅐ 답변을 할 수 있는지 아닌지를 결정하기 위해서 정보를 세팅하는 반영을 이용한다. 만약 그녀가 "얼마나 많은 사람..."이라고 말한다면, 그리고 나서 이용자 인풋이 숫자나 *few* 와 같이 숫자를 나타내는 단어를 포함하고 있다면, 그녀는 이용자가 질문에 답변을 한 것이라고 알아차릴 수 있고 계속 토픽을 이어나갈 수 있다. 그러나 이용자가 벽에 대한 것을 말한다면, 그녀는 새로운 토픽으로 전환할 수 있도록 핵심단어를 재선정할 것이다.

모든 아웃풋이 발생한 후에 적당한 이어짐이 가능할 수 있도록 이것을 실행할 수도 있다. 그렇게 되면 매우 지루할 것이다. 대신에 콘트롤 스크립트를 관리하는 것이 좋다. 이것은 아주 일부의 스크립트만을 필요로 한다.

ChatScript 콘트롤 스크립트는 그 자체로 그냥 토픽이다. 그리고 토픽은 다른 토픽을 불러낼 수 있다. 그래서 여러분은 여러분이 원하는 콘트롤 구조가 어떤 것인지를 정의할 수 있고, 심지어는 아무 때나 콘트롤 구조를 바꿀 수 있다. 챗봇은 콘트롤 데이터를 조종함으로써, becoming drunk, being drunk 등을 시뮬레이트 할 수 있다.

콘트롤 스크립트는 또한 다른 규칙의 조합으로 다양한 처리를 진행할 수 있는데, 다양한 아우풋을 발생시키고 이용자와 관련된 사실과 상관없는 이야기를 저장하기도 한다.

고정된 ChatScript 콘트롤 전략은 다음과 같다 :

a. 어떤 이용자의 인풋을 처리하기 이전에 저작자 구체화 토픽을 실행하라

b. 입력 시 개별 이용자 문장에 저작가 구체화 토픽을 실행하라.

c. 모든 이용자의 인풋을 처리한 후에, 저작자 구체화 토픽을 실행하라.

사전-스크립트는 변수를 재 조정할 수 있게 해 주고, 새로운 인풋을 준비할 수 있게 해 준다.

각각의 문장의 주요한 스크립트를 분석하는 동안, Suzette 는 토픽을 구동시킨다. 관용어 문장을 다시 쓰고, 대명사를 처리하며, 문장을 담론 법에 맞게 구성하여 패턴 매칭을 위한 데이터로 처리한다. 또한 이용자에 대한 팩트를 학습하고, 가장 가까운 매칭 응답을 찾고, 이야기를 발생시키거나 대화를 지연시키는 등의 일을 하나다.

사후-스크립트는 시스템이 이용자가 말한 모든 것을 분석할 수 있게 해 주며, 챗봇이 응답한 모든 것을 분석할 수 있게 해 준다. Suzette 는 사후-스크립트를 많이 이용한다. 토픽을 바꾸어야 할 지를 결정하고, 결정 했다면, 변천하고 있는 문장을 삽입한다. 이용자에게 질문을 하는 것으로 끝이 났다면 이용자가 적절하게 대답을 했는지 아닌지를 알 수 있도록 도와주는 데이터를 준비하도록 한다. 그리고 이용자의 인풋을 다시 간략하게 정리하여 실질적인 응답에 감정적인 반응을 보여줄 것인가를 결정하도록 한다. 그래서 여러분이 *What is 2 + 2, dummy?*라고 물으면, 그녀는 *Who are you calling a dummy? It's 4* 라고 답할 수 있다.

ChatScript 에서 여러분은 여러분이 어떻게 대명사를 처리하고 싶은지를 선택할 수 있다. AIML 스타일의 해결책을 복제할 수 있고, 이렇게 하면 저작자 대명사의 값은 아웃풋을 저작할 당시의 값으로 설정된다. 그렇지 않으면, Suzette 와 같이, 여러분은 자동 대명사 처리 방법을 사용할 수 있는데, 사후-스크립트로 아웃풋을 분석하고 대명사 값을 계산한다.

결과 Results

Loebner Competition 은 챗봇을 위한 해마다 Turing Test 를 한다. 인간은 (인간인 척하는) 인간 공모자와 (사람이라고 하면서 평가자를 속이려고 하는) 챗봇 사이의 대화를 평가한다. 2010 년에 봇 출전자는 주 경쟁을 위한 자질 평가를 위해 지식 테스트를 받아야만 했다. 여기에 최종 4 팀의 리스트가 있다 :

#1 Suzette - Bruce Wilcox - 11 점. 첫 출전
#2 A.L.I.C.E. - Richard Wallace 7.5 점. (2000, 2001, 2004 Loebner winner)
#3 Cleverbot - Rollo Carpenter 7 점. (2005, 2006 Loebner winner)
#4 Ultra Hal - Robert Medekjsza 6.5 점. (2007 Loebner winner)

어떻게 2 년 전에 시작한 신규 진입자인 Suzette 가 쉽게 여러 번 우승한 팀을 이겼을까?

테스트 (정확한 질문이 아니라)의 특징이 미리 공표되었다.

1. 시간과 관련된 질문, *예를 들어, 몇 시입니까? 지금이 아침입니까, 점심입니까, 저녁입니까?*

2. 일반적인 질문, *예를 들어, 망치를 이용하여 무엇을 할 수 있습니까? 택시를 왜 탑니까?*

3. 관계에 관련된 질문, *예를 들어, 포도와 자몽 중에 어느 것이 더 큽니까?*

존이 매리 보다 나이가 많고, 매리는 사라 보다 나이가 많다. 이들 중 누가 가장 나이가 많습니까?

4. "기억"을 보여줄 수 있는 질문, 예를 들어,

선행 : *테니스를 치는 걸 좋아하는 해리라는 이름의 친구가 있다.*

후행 : *내가 방금 말했던 친구의 이름이 무엇입니까?*

해리가 무엇을 하기를 좋아했는지 알고 있습니까?

일반적인 질문은 명사와 관련된 수천 가지를 의미한다. 그래서 *what does one use an xxx for?* 가 정말 의미하고 있는 것에 대한 수십가지의 패턴이 있을 수 있다.

그래서 가장 중요한 것? 그것은 바로 데이터이다. A.L.I.C.E.는 데이터를 철회하고 저장할 좋은 능력을 가진 규칙이 없었다. Cleverbot 은 이 테스트에 적합하지 않은 큰 데이터 베이스를

가지고 있다. 바로 그 안에 ChatScript 장점이 있다. 데이터를 넣기도 철회하기도 쉽다. 나는 각각의 테스트 토픽을 만들고, 오브젝트에 테이블을 포함시켜 함수를 구성했다 :

table: :tool (^what ^class ^used_to_verb ^used_to_object ^used_to_adverb ^use)
-- table processing code not shown ---
[table coffee_table folding_table end_table] furniture rest objects * "eat meals on"
[ladder step_ladder] amplifer reach * higher "reach high areas"

테이블은 아래와 같은 것을 다룬다 :

실제 테스트 : *칼로 무엇을 할까요*

답: *칼은 음식을 자르는데 이용됩니다.*

이용자가 언급한 바를 기억하고 있는가에 관련된 Suzette 의 토픽은 :

실제 테스트*: 내 친구 밥은 테니스 치는 것을 좋아한다. 밥이 좋아하는 운동은 무엇입니까?*

답: *그 게임은 테니스입니다.*

그리고 Suzette 는 55 개의 규칙으로 많은 토픽을 가지고 있어서, 직접 간단한 수식을 처리하여 계산하여 수학적 문제를 이용한 이야기를 만들 수 있었다.

실제 테스트: *12 다음의 숫자는 무엇입니까?*

답: *13*

그러나 지식 테스트는 대화가 아니다. 합격자의 성공은 실제 콘테스트에서의 판정자로부터 얻은 승리임에 분명하다. 2009 년에 심사위원은 10 분 간의 시간 만 소비하여 결정하였다. 2010 년에는 25 분을 소비하였다. 그럼에도 불구하고, Suzette 는 4 명 중의 한 명의 심사위원을 속여서 그녀가 인간이라고 생각하게 만들었고, 대회에서 승리했다.

결론 Conclusion

수 많은 데이터는 자연어를 다루는데 매우 중요하다. 그러나 데이터가 전부가 아니다. 획득되는 챗이 4500 만개에 달했음에도 불구하고, Cleverbot 는 Loebner Competition 에서 3 등을 차지하였다. 적절한 검색을 위한 좋은 매커니즘이 부족했기 때문이다. 구식의 AIML 기반의

A.L.I.C.E.는 120K 규칙으로 2 등을 차지했다. 우승자, ChatScript 기반의 Suzette 는 A.L.I.C.E. 보다 조금 더 많은 데이터를 가지고 있었지만, 훨씬 더 간결하고 정확한 검색을 할 수 있었다.

ChatScript 는 간단한 비주얼 컴퓨터 언어 문법을 지원하고 강력한 패턴 매칭 특징을 가지고 있다. 하나의 단어 대신에 컨셉을 매칭시키고 인풋에 있어서는 안되는 단어를 명시할 수 있도록 하는 것은 의미의 패턴을 조정할 수 있도록 해 주었다. 범위가 있는 와일드카드를 사용하는 것은 잘못된 구분을 줄여주었다. 핵심단어를 중심으로 세부적으로 나누어진 규칙은 검색이 용이한 규칙의 콜렉션을 만들었고, 콘텐츠 저작을 쉽게 해 주었다. 그리고 이것은 좋은 시작에 불과하다.

참고 자료 References:

alicebot.blogspot.com/

www.interactivestory.net/ (Façade)

www.chatbots.org/chatbot/suzette/ (Feb/2010 버전, Loebner Oct/2010 version 은 아님)

www.sourceforge.net/projects/chatscript (Loebner chatbot engine)

www.loebner.net/Prizef/loebner-prize.html

www.cleverbot.com/

www.personalityforge.com