MAC0425 - EP2 - Akinator

Relatório do EP2 de MAC0425 (Inteligência Artificial)

Aluno: Renato Lui Geh
NUSP: 8536030

19 de outubro de 2015

1 Regras e fatos

Neste EP adicionamos regras e fatos das séries de TV *Breaking Bad* e *Game of Thrones*. Temos os seguintes personagens:

- Game of Thrones
  - Tyrion Lannister
  - Tywin Lannister
  - Jamie Lannister
  - Cersei Lannister
  - Arya Stark
  - Jon Snow
  - Sansa Stark
  - Jorah Mormont
  - Bran Stark
  - Hodor
- Breaking Bad
  - Walter White
  - Skyler White
  - Jesse Pinkman
  - Hank Schrader
  - Flynn White
  - Marie Schrader
  - Saul Goodman
  - Mike Ehrmantraut
  - Gus Fring
  - Badger

Os predicados foram escolhidos de tal forma que houvesse maior corte de possíveis personagens durante a consulta. Também evitei colocar predicados desnecessários salvo alguns por efeito de humor (exemplo: (`badger`) `is_stupid`).

2 Predicados ideais

É fácil ver que para deixar a consulta o mais eficiente possível devemos sempre tentar dividir as possíveis respostas em pedaços que tenham a menor diferença possível entre elas. Num mundo ideal, teríamos todos os personagens com predicados que os dividissem igualmente ao meio, gerando uma resposta a qualquer query em $O(logn)$. Infelizmente no mundo real modelarmos de tal forma é impossível, portanto tentamos achar predicados que sejam o mais próximos possíveis a esta "busca binária".

Além disso, se visualizarmos uma consulta como percorrer os nós da árvore de busca, qualquer nó que possua apenas um nó filho pode ser descartado por ser irrelevante. Há casos no EP em que há nós desnecessários que poderíamos descartar.

Um outro ponto importante é a ordem em que os predicados aparecem. Ao ordenarmos os predicados por ordem decrescente de uso, estaremos limitando as possíveis respostas mais rapidamente que se ordenarmos em ordem crescente, já que podemos acabar respondendo perguntas irrelevantes em alguns casos.

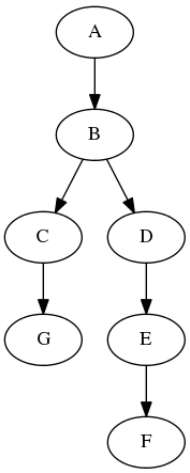

Figura 1: Note que os nós (predicados) A, E, G e F são irrelevantes e podemos tira-los. Além disso o nó B deve ser o primeiro predicado a ser perguntado, já que este impacta mais na inferência.

3 Prolog

Prolog é uma linguagem bem diferente das quais estou acostumado. Por ser uma linguagem onde lidamos com fatos e regras ao invés de ser uma linguagem procedimental e imperativa como C, demorou um tempo para acostumar-se. No entanto, é uma experiência diferente e muito interessante.

4 Desempenho

O programa conseguiu acertar todas as vezes o personagem. No entanto, para certos personagens, se o usuário erra uma das características dele, o programa imprime como certo. Por exemplo, se acertarmos que `flynn_white` tem as características `is_male`, `is_teenager`, `has_daddy_issues` mas errarmos e dissermos que ele `is_bold`, o programa dirá que o personagem escolhido foi `flynn_white` ao invés de `unknown`.

Com relação a perguntas em média, o programa conseguia decidir qual o personagem escolhido com cinco perguntas em média. Além disso, é interessante notar que os personagens que estavam antes na lista de declaração de regras, foram os que obtiveram menor número de perguntas para serem adivinhados. Além disso, aqueles que possuíam características compartilhadas com um pequeno número de personagens tinham maior média de perguntas.

5 Melhorias

Como vimos na seção Predicados Ideais e Desempenho, mudar a ordem dos predicados é um jeito de melhorarmos a performance do programa. Além disso, formular predicados que sejam compartilhados com um maior número de personagens também, já que podemos excluir aqueles que admitem tal característica daqueles que não admitem em maior quantidade, e portanto limitaremos o conjunto de possíveis respostas mais rapidamente. Outro jeito, mais óbvio, de melhorar o programa seria descartar todos os predicados desnecessários, ou seja, que não alterem o resultado da query.

6 Notas

Nas linhas 2 e 18 do código, tentei melhorar o jeito de se adicionar fatos a base de conhecimento. Minha intenção era colocar todos os personagens numa lista e para cada membro, adicionar o fato:

$$guess(X) :- X, !. \tag{1}$$

Infelizmente meu conhecimento de Prolog é muito ruim e isso não funciona. :(