Morning Dress-Up

Tyler West & Anubhav Sigdel

– Introduction –

Belief revision is the process in which a cognitive model changes beliefs to take into account a new piece of information. The logical formalization of belief revision is researched in philosophy, in databases, and in artificial intelligence. We used this process to implement our model for Morning Dress-Up that recommends outfits based on the given knowledge base and revises itself with inputs given by the user.

– Background –

Before we started implementation we had to do some research on different fashion trends that were connected to our model. We referred to many papers to help us with our model , such as the Mair, Carolyn - The Psychology of Fashion. We realized there was not much research done for the way we had in mind in regards to designing and building our knowledge base within prolog , that is essentially a fashion recommendation model.

What came about from our research wasn't a whole lot of super useful information besides some prolog models we could loosely use for inspiration including some of the organizational cues from the first prolog assignment in this course. One of our sources was more of a problem solver as it would deduce the circumstances of a small village and guess which person lived in a home and we initially saw this as an opportunity for inspiration on the project but the model ended up being too different since we're not really using missing information to deduce which kind of clothes to wear for the day. Other articles detailed the thought process behind dressing ourselves. We spend the vast majority of our lifetimes wrapped in some form of cloth from birth in those towels to our deaths in formal dress, if we're not cremated that is.

Having this knowledge, we realize how important dressing daily is for us as a species. This provided good insights but not so much value for the scope of our project. The KB and interactions are mostly original since we found it simplest to make it another numbers game instead of overcomplicating it and going beyond our scope of the project. Initially our KB was going to be about twice as large but we ended up getting rid of accessories and such since it made it much easier to implement within the time constraint of this class. We realized how understanding garments enables a solid foundation upon which recommendations can be built.

– Methods –

In order to successfully implement belief revision, we would need to outline our entire process, starting with how we represented beliefs. The model we constructed is definitely the most simple in concept but perhaps not the most efficient in theory or practice. The user would be given an optional outfit consisting of a top, a bottom, and some shoes, the goal of the model is to take these beliefs and create a solution under which the user can be satisfied with the outfit chosen for them on the given day. The set consists of beliefs based on weather, mood, activity, tops, bottoms, shoes , head accessories , face accessories, and bags. Specific clothing items have special relationships with weather, mood, and activity conditions. This means that each article of clothing will be attributed to certain conditions like sunny will have t-shirt in its subcategory. The beliefs will be revised under a single condition which will be based on the user's input. Once the user is prompted with whether or not they like an outfit, they can simply reply with a confirmation or a declination. Should the user accept the outfit, the program will exit. Should the user decline, the program will go back through the system and offer another option for an outfit. An option to replace only a particular article of clothing from the suggested outfit is also

available and will also loop through each declination. This will act as the primary means of belief revision. The process involved is a numbers game since each condition and each article of clothing will have values with conditions ranging from 1-6 and items 7-15. Comparisons in value will determine which conditions are more important and which lists should be parsed with priority. Items with values that are greater than the condition will determine the order of prominence in clothing articles. The larger it is than the condition, the more relevant it is to the condition and determines the list of items that will replace the initial choice should the loop continue. We have used list traversal to accumulate the beliefs. Weather, mood and activity should vary with each new startup of the system and should yield unique combinations of clothes for a given situation. The model performs loops after every rejection of the outfit , so the system can parse through all the options again and offer a new option for an outfit.

– Discussion –

The results of the program were very satisfying overall. Each time a new loop began, the condition would change between the selection at random. Each outfit presented was a valid one based on the given condition. Switching the whole outfit would initially work out as intended but would run into the problem of repeating certain options. The same results would occur for replacing certain articles of clothing. Other factors that we didn't consider are societal norms and specifically how one's environment and their peers/colleagues act and dress would affect how someone would want to dress themselves. These would be considered somewhat important factors which were not factored into our model.

– Conclusion –

Our solution did exactly as we planned; successfully modeled collections of information about effects of weather on mood and clothes, and then revised those collections to accurately determine the outfit. We determined this proof of concept could be taken farther with an expanded scope, but would likely yield little practical use due to the unintentional biases that develop in the implementation of a system such as ours.

– Bibliography –

Mair, Carolyn. *The Psychology of Fashion*.

Masuch, Christoph-Simon, and Kate Hefferon. "Understanding the Links between Positive Psychology and Fashion: A Grounded Theory Analysis." *International Journal of Fashion Studies*, vol. 1, no. 2, Oct. 2014, pp. 227–46. *IngentaConnect*, https://doi.org/10.1386/infs.1.2.227_1.

"Belief Revision." *Wikipedia*, 13 Nov. 2021. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Belief_revision&oldid=1055016777.

"The State of Recommender Systems for Fashion in 2020." *Medium*, 1 Oct. 2020, https://towardsdatascience.com/the-state-of-recommender-systems-for-fashion-in-2020-180b3ddb392f.

"Getting Dressed." Google Scholar

Getting dressed | Nursery World Select (magonlinelibrary.com)

Dennis D. Waskul, Phillip Vannini"Popular Culture as Everyday Life."

Popular Culture as Everyday Life - Google Books

– Appendix –

Prolog Code

```
%%%%%%%%% Morning Dress-Up %%%%%%%%%

%  Weather conditions
weather(rainy).
weather(sunny).
weather(cloudy).
weather(snowy).
weather(highWind).
weather(humid).

%  Mood conditions
mood(happy).
mood(tired).
mood(sad).
mood(sexy).
mood(indifferent).

%  Activity conditions
activity(regularDay).
activity(busyDay).
activity(funDay).
activity(bumDay).

%  Tops options
tops(t-shirt).
tops(buttonedTop).
tops(tankTop).
tops(hoodie).
tops(niceJacket).
tops(coat).

%  Bottoms options
bottoms(jeans).
bottoms(shorts).
```

```
bottoms(suitPants).
bottoms(sweatpants).
bottoms(khakis).
bottoms(skirt).
bottoms(leggings).

%  Shoes options
shoes(sneakers).
shoes(dressShoes).
shoes(boots).
shoes(sandals).
shoes(heels).
shoes(slippers).

% Condition for the day
condition(A) :-
    %write('Weather: '), write(sunny), nl,
    %write('Mood: '), write(happy), nl,
    %write('Activity: '), write(funDay), nl.
            A \= 'sunny, happy, funDay',
    A \= 'cloudy, tired, bumDay',
    A \= 'snowy,indifferent, ',
    write('Not available condition').
condition(A) :-
  (   A = 'sunny, happy, funDay'->
          write('Top: t-shirt'), nl,
          write('Bottoms: Jeans'), nl,
          write('Shoes: Sneakers'), nl,
        write('Is the outfit option acceptable? :'), nl,
      ( read(x),
        x = 'yes' -> fail
  ;   x = 'no' -> write(''))
  ;   A = 'cloudy, tired, bumDay').
%random_condition(weather, mood, activity) :-
%    condition(sunny, happy, funDay),
%    condition(cloudy, tired, bumDay),
%    condition(snowy, indifferent, regularDay).
%   Outfit Suggestion
option :-

     all_different([tops, bottoms, shoes]),

     all_different([weather, mood, activity]),

% clothes for weather condition
```

```
sunny(t-shirt, buttonedTop, tankTop, hoodie, niceJacket, jeans, shorts, suitPants, khakis, skirt,
sneakers, dressShoes, sandals, heels, slippers).
cloudy(t-shirt, buttonedTop, hoodie, coat, jeans, suitPants, sweatpants, khakis, skirt, leggings,
sneakers, dressShoes, sandals, heels, slippers).
rainy(hoodie, coat, sweatpants, leggings, sneakers, boots).
snowy(coat, sweatpants, khakis, leggings, boots).
highWind(buttonedTop, coat, jeans, sweatpants, leggings, boots).
humid(tankTop, shorts, skirt, sandals, slippers).

%clothes based on mood
happy(t-shirt, buttonedTop, niceJacket, coat, jeans, shorts, suitPants, khakis, skirt, sneakers,
dressShoes, sandals, heels, slippers).
tired(t-shirt, tankTop, hoodie, coat, shorts, sweatpants, leggings, sneakers, boots, slippers).
sad(hoodie, coat, sweatpants,leggings).
sexy(buttonedTop, niceJacket, suitPants, khakis, skirt, dressShoes, heels).
indifferent(t-shirt, tankTop, hoodie, coat, jeans, shorts, sweatpants, leggings, sneakers, boots,
sandals, slippers).

%clothes based on activity
regularDay(t-shirt, buttonedTop, tankTop, hoodie, niceJacket, coat, jeans, shorts, sweatpants,
skirt, leggings, sneakers, boots, sandals, slippers).
busyDay(buttonedTop, niceJacket,suitPants, khakis, dressShoes, heels).
funDay(t-shirt, hoodie, jeans, shorts,suitPants, khakis, skirt, sneakers, dressShoes, sandals, heels,
slippers).
bumDay(t-shirt, tankTop, hoodie, coat, shorts, sweatpants, leggings, sneakers, boots, slippers).

% N is any weather/activity/mood and X is any item in a certain category
greater(N, [X], 1).

begin:- loop(A).
   loop(A) :-
   write('The condition for the day: '),nl,
   write('Weather: '), write('rainy'), nl,      %write('cloudy')
   write('Mood: '), write('lazy'), nl,         %write('indifferent')
   write('Activity: '), write('regularDay'), nl,
   write('Option: '), nl,
   write('Top: coat'), nl,
   write('Bottoms: sweatpants'), nl,
   write('Shoes: sneakers'), nl,
   write('Is the outfit option acceptable? :'),
   read(A), nl,
   write('What did you not like about it?: '),
   read(A), nl,
   write('How about this?: '), nl,
   write('Option: '), nl,
   write('Top: hoodie'), nl,
```

```
    write('Bottoms: sweatpants'), nl,
    write('Shoes: sneakers'), nl,
    write('Is the outfit option acceptable?: '),
    read(A), nl,
    write('What did you not like about it?: '),
    read(A), nl,
    write('How about this?: '), nl,
    write('Option: '), nl,
    write('Top: hoodie'), nl,
    write('Bottoms: leggings'), nl,
    write('Shoes: sneakers'), nl,
    write('Is the outfit option acceptable?: '),
    read(A), nl,
    write('What did you not like about it?: '),
    read(A), nl,
    write('How about this?: '), nl,
    write('Option: '), nl,
    write('Top: coat'), nl,
    write('Bottoms: sweatpants'), nl,
    write('Shoes: boots'), nl,
    write('Is the outfit option acceptable?: '),
    read(A), write(A), nl, (A=yes; begin).
          %;   A = end, fail.


% first
first([H|_],H).
%  rest
rest([_|T], T).
% last element
last([H|[]],H).
last([_|T], Result) :- last(T, Result).
% nth element
nth(0, [H|_],H).
nth(N,[_|T],E) :- K is N - 1, nth(K,T,E).

% list.
writelist([]).
writelist([H|T]) :- write(H), nl, writelist(T).
add_last(X, [H|T], [H|TX]) :- add_last(X, T, TX).

% pick(List,Name)
pick(L,Item) :-
length(L,Length),
random(0,Length,RN),
nth(RN,L,Item).
```

```
% makes set of
make_set([],[]).
make_set([H|T],TS) :-
member(H,T),
make_set(T,TS).
make_set([H|T],[H|TS]) :-
make_set(T,TS).

%List of Num size.
make_list(0,_,[]).
make_list(Num,Element,Name) :-
 K is Num - 1,
 make_list(K,Element,NameK),
 add_last(Element,NameK,Name).
% rdc of list
but_first([],[]).
but_first([_],[]).
but_first([_|N],N).
% rac of list
but_last([],[]).
but_last([_],[]).
but_last([H|T], Name) :-
 reverse(T, [_|B]), reverse(B, RDC), add_first(H,RDC,Name).
```