CPSC 312

Functional and Logic Programming

6 October 2015

Assignment 2

1. None of the 5 functions need an auxiliary function that we haven't seen in the class. Lecture slides (particularly Sep 24th) are your best friend for finding your solutions.

2. USE PATTERN MATCHING as much as you can.

3. THINK RECURSIVELY: don't worry about the entire list; only focus on one member at a time. if there was only member what would you do? now if there was more how would you do it *differently*? writing the recursive step is all about figuring out that difference. Train yourself to think about in terms of THE HEAD AND THE REST. what should you do for the head at every step of recursion? what should you do for the rest?

Assignment 2

1. TRACE YOUR PROGRAMS. trace/0 trace/1 trace/2 notrace/0

2. Are your base cases redundant? your base case is not just any possible base case, but the terminating case of your recursion. Some cases are possible base cases for a program but they are not the terminating condition for the recursion you've written. You don't need them.

Summary to this point

- The cut predicate controls backtracking. It tells Prolog not to pass back through this point when looking for alternative solutions.

- The ! acts as a marker, back beyond which Prolog will not go. When it passes this point, all choices that it has made so far are locked in -- they are treated as though they are the only possible choices.

- Note that the cut always appears where a predicate can appear (never, for example, as an argument to a predicate). It is treated like any other predicate, and *it always succeeds*.

Summary to this point

The effect of the cut is as follows:

- Any variables which are bound to values at this point cannot take on other values.
- No other versions of predicates called before the cut will be considered.
- No other subsequent versions of the predicate at the head of the current rule will be considered.

Green cuts vs Red cuts

- That's not to say that red cuts can't be useful.

- Using red cuts we can write an if_then_else construct. How?

```
if_then_else(P,Q,R) :- P, !, Q.
if_then_else(P,Q,R) :- R.
```

Negation as failure

- How can we implement negation in prolog? We've seen the predicate not/1 before, but how does it work?
- In a prolog program, you typically only express what *is* correct, not what isn't. Closed World Assumption ensures, what is not expressed or cannot be proven is false.
- So falseness is not something you can directly express. We haven't, for instance, been writing programs that contain lists of false facts.
- Therefore, negation is an implicit or derived condition modeled as 'failure to prove'.

not/1 or \+

The negation of a formula is true iff it cannot be proven true.

Thus if you cannot prove a(X), then not(a(X)) is considered true!

This definition in Prolog reflects the fact that we don’t know about our world is considered to be false.

not/1 or \+

```
not(P):- call(P), !, fail.
not(P).
```

`?- call(P)` forces an attempt to satisfy the goal P

This is also an if-then-else construct!

If call(P) succeeds, then fail, otherwise succeed.

not/1 or \+

```
not(P):- call(P), !, fail. not(P).
```

`?- call(P)` forces an attempt to satisfy the goal P

This is also an if-then-else construct!

If call(P) succeeds, then fail, otherwise succeed.

If P has variables, if any instances of those variables lead to satisfying P, then not/1 fails.

remember this example..?

How do we prove that fred is the father of everyone?

we can't we can only prove it's opposite:

```
?- not(father(fred,X)).
```

but this is not the true logical opposite.

Using negation

Unmarried student rule:

```
unmarried_student(X) :- not(married(X)),
student(X).
```

looks like the right code...

if I add:

```
student(bill).
married(joe).
```

would querying `?-unmarried_student(X)` produce `bill` as an output?

Using negation

```
unmarried_student(X) :-
not(married(X)), student(X).
```

```
student(bill).
married(joe).

?-unmarried_student(X).
false.
```

Using negation

```
unmarried_student(X) :- not(married(X)),
student(X).

student(bill).

married(joe).

?-unmarried_student(X).

false.
```

because married(X) unified with married(joe), therefore its negation fails.

how to fix it?

Using negation

```
unmarried_student(X) :- not(married(X)),
student(X).

student(bill).

married(joe).

?-unmarried_student(X).
false.
```

because married(X) unified with married(joe), therefore its negation fails.

how to fix it?

change the order of the clauses in the body of the rule.

Using Negation

```
unmarried_student(X) :-
student(X), not(married(X)).

student(bill).

married(joe).
```

Using Negation

```
unmarried_student(X) :-
student(X), not(married(X)).
```

```
student(bill).
```

```
married(joe).
```

what if I comment out married(joe) and keep the order as it was? would that work?

Using Negation

```
unmarried_student(X) :- student(X),
not(married(X)).

student(bill).
married(joe).
```

- In SWI-Prolog, you need to use \+ instead of not, so:

```
unmarried_student(X) :- student, \+married(X).
```

Using Negation

```
unmarried_student(X) :-
not(married(X)), student(X).

student(bill).

%married(joe).
```

what if I comment out `married(joe)` and keep the order as it was? would that work?

Using Negation

```
unmarried_student(X) :- student(X),
not(married(X)).

student(bill).

%married(joe).
```

what if I comment out `married(joe)` and keep the order as it was? would that work?

Not in SWI-Prolog:

```
ERROR: unmarried_student/1: Undefined
procedure: married/1
```

Using Negation

```
unmarried_student(X) :- student(X),
not(married(X)).
```

vs

```
unmarried_student(X) :- not(married(X)),
student(X).
```

Overall lesson of this example: Negation does not or may not work as expected with non-ground goals. Use with caution.

Using Negation

- For the purpose of this class: learn very well the meaning of negation as failure and how its behaviour differs from what's expected of a logical negation (because it's not a logical negation).

- Remembering how not/1 (or \+) is implemented will help you reason about its behaviour and output.

- Study what's left of section 11.3 if you're interested in learning to effectively use negation.

Zebra Puzzle

1. There are five houses.
2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.
8. Kools are smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house.
11. The man who smokes Chesterfields lives in the house next to the man with the fox.
12. Kools are smoked in the house next to the house where the horse is kept.
13. The Lucky Strike smoker drinks orange juice.
14. The Japanese smokes Parliaments.
15. The Norwegian lives next to the blue house.

Who drinks water?
Who owns the zebra?

first step: how to structure and represent the information?

- "There are five houses."
- Where are the houses? let's say on a street block.
- So what we have is a block with the characteristic of having five houses. How to represent that in logic?
- 5 members in a list perhaps?
- or 5 arguments in the predicate block?

Generalizing information

- In order to represent information logically, we need to find a way to *generalize* the specifc information we are given. Imaigne you're representing not just this specific set of information, but a world in which this information holds.

- In describing this world, just like urban planning, you first come up with an outline containing the city blocks, then fill the blocks with houses, and give each house different characters.

first step: how to structure and represent the information?

- "There are five houses."
- 5 members in a list perhaps?
- or 5 arguments in the predicate block?
- which one is more general?
- block(House1,House2,House3,House4,House5).
- block(ListofHouses).

Generalizing information

- Now that we have a structure that can "hold" the five houses, let's see how we can generalize the information about the houses.

- Some of the assertions:

2. The Englishman lives in the red house.
3. The Spaniard owns the dog.
4. Coffee is drunk in the green house.
6. The green house is immediately to the right of the ivory house.
7. The Old Gold smoker owns snails.

- What *types of* characteristics are used to describe the houses in each assertion?

Generalizing information

- What types of characteristics are used to describe the houses in each assertion?

2. The Englishman lives in the red house. -> a house has a colour and a resident's nationality.

3. The Spaniard owns the dog. -> each house has (once again) a nationality and a pet.

4. Coffee is drunk in the green house. -> each house has a signature drink (and once again a colour)

6. The green house is immediately to the right of the ivory house. -> each house can be desribed by its relative location to other houses

7. The Old Gold smoker owns snails. -> each house has a signature cigarettes brand (and once again a pet)

Generalizing information

- What types of characteristics are used to describe the houses in each assertion?

2. The Englishman lives in the red house. -> a house has a colour and a resident's nationality.
3. The Spaniard owns the dog. -> each house has (once again) a nationality and a pet.
4. Coffee is drunk in the green house. -> each house has a signature drink (and once again a colour)
6. The green house is immediately to the right of the ivory house. -> each house can be desribed by its relative location to other houses
7. The Old Gold smoker owns snails. -> each house has a signature cigarettes brand (and once again a pet)

- If we continue doing this for all assertions, we'll come up with a unique set of characteristics that can be used to describe each house.

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	?	Blue	Chesterfield	?	5

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	?	Blue	Chesterfield	?	5

What's the last pet and the last drink?

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	?	Blue	Chesterfield	?	5

What's the last pet and the last drink?

Remember the question of the puzzle?

"Who drinks water? Who owns the zebra?"

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

We know what goes under each column but we don't know in what order...

that's what we need to find out!

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

We know what goes under each column but we don't know in what order...

that's what we need to find out!

But first, given this table, we now should be able to logically represent a house.

House/6

- Based on the table, a house can be described by its six characteristics: Nationality, Colour, Drink, Cigarettes, Pet, and Location on the block.
- ```
  House(Nationality,
        Colour,
        Drink,
        Cigarettes,
        Pet,
        Location).
  ```
- There would be five of these predicates in the solution, each a member of the listOfHouses in the predicate building.
  ```

House/6

- Based on the table, a house can be described by its six characteristics: Nationality, Colour, Drink, Cigarettes, Pet, and Location on the block.
- ```
  House(Nationality,
        Colour,
        Drink,
        Cigarettes,
        Pet,
        Location).
  ```
- Could we have put these characteristics in a list? what's the difference between this list and the list of houses?
  ```

a list of elements vs arguments of a predicate

- Semantically, a list is considered as a collection of *homogenous* elements. All the houses on the block are of the same nature or type, (in object-oriented paradigm, they are like instances of the same class).

- Whereas the arguments of a predicate are *heterogenous* elements. They have no natural relevance to eachother other than through their relation to the predicate. The characteristics of a house are only connected because of their connection to the entity of a house. Putting them in a list will be semantically confusing.

a list of elements vs arguments of a predicate

- Moreover, lists provide a different access than arguments of a predicate. In lists, the order of elements can be irrelevant.
- For a predicate's argument however, the order is the enabling factor: order makes pattern matching possible.
- In this question, we don't know how the houses should be ordered (that's part of what we want to find out!), so we can't use pattern matching directly. Instead we can state one of the houses should have these two or more characteristics at the same time. This is much easier to done through list processing.

a list of elements vs arguments of a predicate

- All that being said, it doesn't mean this is the only way to solve this problem. After you've seen the solution, you're welcome to try for yourself to change the representation and see how that affects your program.
- These considerations are only to equip you with a kind of reasoning conducive to effective program design in prolog.

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"The Englishman lives in the red house."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	Coffee	*Red*	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Green	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"The Englishman lives in the red house."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	Coffee	*Red*	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Green	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"The Spaniard owns the dog."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	Coffee	*Red*	Old Gold	Horse	1
House 1 to 5	*Spaniard*	OJ	Green	Parliaments	*Dog*	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"The Spaniard owns the dog."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	Coffee?	*Red*	Old Gold	Horse	1
House 1 to 5	*Spaniard*	OJ	Green?	Parliaments	*Dog*	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"Coffee is drunk in the green house."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	Milk	*Red*	Old Gold	Horse	1
House 1 to 5	*Spaniard*	OJ	Yellow	Parliaments	*Dog*	2
House 1 to 5	Ukrainian	*Coffee*	*Green*	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

Each Assertion gets us closer to the correct order. For instance:

"Coffee is drunk in the green house."

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	*English*	***Milk***	*Red*	*Old Gold*	*Snails*	***3***
House 1 to 5	***Japanese***	Water	Ivory	***Parliaments***	Zebra	2
House 1 to 5	***Norwegian***	*Coffee*	*Green*	Chesterfields	Horse	***1***
House 1 to 5	*Ukrainian*	*Tea*	***Yellow***	***Kools***	Fox	4
House 1 to 5	*Spaniard*	*OJ*	Blue	*Lucky Strike*	*Dog*	5

Each Assertion gets us closer to the correct order.

After applying all assertions except for 6,11,12,15

Applying the constraints

- "The Englishman lives in the red house".
- How do we represent this contraint now that we have a predicate for houses and one for the block?

Applying the constraints

- "The Englishman lives in the red house".
- How do we represent this contraint now that we have a predicate for houses and one for the block?
- There needs to be a house/6 in the ListOfHouses, such that the nationality of the house is english and the colour is red.

Applying the constraints

- "The Englishman lives in the red house".
- How do we represent this contraint now that we have a predicate for houses and one for the block?

- There needs to be a house/6 in the ListOfHouses, such that the nationality of the house is english and the colour is red.
- `house(english,red,_Pet,_Drink,_Cigs,_Loc).` this effectively means, if nationality is english, then colour is red and vice versa, regardless of the other characteristics.

Applying the constraints

- "The Englishman lives in the red house".
- How do we represent this contraint now that we have a predicate for houses and one for the block?

- There needs to be a house/6 in the listOfHouses, such that the nationality of the house is english and the colour is red.
- `member(house(english,red,_Pet,_Drink,_Cigs,_Loc),ListOfHouses).`

Applying the constraints

- "The Spaniard owns the dog".
- `house(?,?,?,?,?,?).`

Applying the constraints

- "The Spaniard owns the dog".

- `member(house(spaniard,_Colour,dog,_Drink,_Cigs,_Loc),ListOfHouses).`

Applying the constraints

- "The Spaniard owns the dog".

- `member(house(spaniard,_Colour,dog,_Drink,_Cigs,_Loc),ListOfHouses).`

- "Coffee is drunk in the green house".

- `member(house(_Nationality,green,_Pet,coffee,_Cigs,_Loc),ListOfHouses).`

Applying the constraints

- "The Spaniard owns the dog".
- `member(house(spaniard,_Colour,dog,_Drink,_Cigs,_Loc),ListOfHouses).`
- "Coffee is drunk in the green house".
- `member(house(_Nationality,green,_Pet,coffee,_Cigs,_Loc),ListOfHouses).`
- Write the rest on your own. (all except for 6,11,12,15)

Applying the constraints

- What about these ones?
- "6. The green house is immediately to the right of the ivory house."
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- "12. Kools are smoked in the house next to the house where the horse is kept."
- "15. The Norwegian lives next to the blue house."

Applying the constraints

- What about these ones?
- "6. The green house is immediately to the right of the ivory house."
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- "12. Kools are smoked in the house next to the house where the horse is kept."
- "15. The Norwegian lives next to the blue house."
- The different here is that we are no longer describing one house, rather describing two houses in relation to one another.

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- the green house: house(_,green,_,_,_,_).
- the ivory house: house(_,ivory,_,_,_,_).

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- the green house: house(_,green,_,_,_,_).
- the ivory house: house(_,ivory,_,_,_,_).
- How do you state that the location of the green house is to the right of the ivory house?

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- the green house: house(_,green,_,_,_,Lg).
- the ivory house: house(_,ivory,_,_,_,Li).
- How do you state that the location of the green house is to the right of the ivory house?

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- the green house: house(_,green,_,_,_,Lg).
- the ivory house: house(_,ivory,_,_,_,Li).
- How do you state that the location of the green house is to the right of the ivory house?
- Lg is Li+1

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- the green house: house(_,green,_,_,_,Lg).
- the ivory house: house(_,ivory,_,_,_,Li).
- How do you state that the location of the green house is to the right of the ivory house?
- Lg is Li+1
- putting it all together...

Applying the constraints

- Let's take this as an example:
- "6. The green house is immediately to the right of the ivory house."
- Forms one conjunctive clause:

```
member(house(_,green,_,_,_,Lg), ListOfHouses),
member(house(_,ivory,_,_,_,Li),
ListOfHouses),Lg is Li+1.
```

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- the man who smokes Chesterfields:
  `house(_,_,_,_,chesterfields,_)`
- the man with the fox:
- `house(_,_,fox,_,_,_).`

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- the man who smokes Chesterfields:
  `house(_,_,_,_,chesterfields,_)`
- the man with the fox:
- `house(_,_,fox,_,_,_).`
- What about next to?

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- the man who smokes Chesterfields:
  `house(_,_,_,_,chesterfields,_)`
- the man with the fox:
- `house(_,_,fox,_,_,_).`
- What about next to?
- something is next to another thing, when it's either on its right side or its left side.

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- the man who smokes Chesterfields:
  `house(_,_,_,_,chesterfields,_)`
- the man with the fox:
- `house(_,_,fox,_,_,_).`
- next to:
- `next_to(P,Q) :- P is Q-1.`
- `next_to(P,Q) :- P is Q+1.`

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- putting it all together:
- `member(house(_,_,_,_,chesterfields,Pc), ListOfHouses),member(house(_,_,fox,_,_, Pf),ListOfHouses), next_to(Pc,Pf).`

Applying the constraints

- What about this?
- "11. The man who smokes Chesterfields lives in the house next to the man with the fox."
- putting it all together:
- `member(house(_,_,_,_,chesterfields,Pc), ListOfHouses),member(house(_,_,fox,_,_, Pf),ListOfHouses), next_to(Pc,Pf).`
- write the rest on your own.

What is the end result?

- After applying the constraint, the end result will be one really large conjuction (or body).

What is the end result?

- From applying the constraint, the end result is one really large conjuction (or body).

- What's the head?

What is the end result?

- From applying the constraint, the end result is one really large conjuction (or body).
- What's the head?
- `block(ListOfHouses) :- ...`

What is the end result?

- From applying the constraint, the end result is one really large conjuction (or body).
- What's the head?
- `block(ListOfHouses) :- ...`
- meaning: if all these conditions are true then ListOfHouses describes the houses on this block and their locations.

What is the end result?

- The full program will be available for you to download on Piazza this afternoon.

What is the end result?

- The full program will be available for you to download on Piazza this afternoon.
- Are there other ways to solve this problem?

What is the end result?

- The full program will be available for you to download on Piazza this afternoon.
- Are there other ways to solve this problem? yes!
- As we said, we could have used arguments of a predicate, instead of a list, or vice versa.

What is the end result?

- The full program will be available for you to download on Piazza this afternoon.
- Are there other ways to solve this problem? yes!
  - As we said, we could have used arguments of a predicate, instead of a list, or vice versa.
  - Another way we could've implemented it was to bind the location of a house to its place on the list (or in the list of arguments).

What is the end result?

- The full program will be available for you to download on Piazza this afternoon.
- Are there other ways to solve this problem? yes!
- As we said, we could have used arguments of a predicate, instead of a list, or vice versa.
- Another way we could've implemented it was to bind the location of a house to its place on the list (or in the list of arguments).
- Instead of having an explicit Location argument in house/6, we'll write house with 5 arguments and in describing the constraints about location, we'll use the place of the house element in the list of houses.

Nondeterminstic programming

- Chapter 14 of your textbook (in particular sections 14.1 and 14.2) talks about nondeterministic programming.

Nondeterminstic programming

- Chapter 14 of your textbook (in particular sections 14.1 and 14.2) talk about nondeterministic programming.

- The program we just wrote is an example of nondeterministic programming, because:

Nondeterminstic programming

- Chapter 14 of your textbook (in particular sections 14.1 and 14.2) talk about nondeterministic programming.

- The program we just wrote is nondeterministic, because:

1. we didn't explicitly write a procedure that constructs the answer in a deterministic way. we expressed the constraints and let prolog find the answer.

Nondeterminstic programming

- Chapter 14 of your textbook (in particular sections 14.1 and 14.2) talk about nondeterministic programming.

- The program we just wrote is nondeterministic, because:

1. we didn't explicitly write a procedure that constructs the answer in a deterministic way. we expressed the constraints and let prolog find the answer.
2. The way prolog builds the solution is by starting off with a general solution, applying the constraints to it one at a time, until a constraint fails on an existing solution; then it backtracks and tries other ways of applying previous constraints. (trace the execution of the zebra porgram to see how this works). This means it tries out a great number of possible solutions until the right one is found.

Nondeterminstic programming

- This is a general template for writing nondeterministic programs:

- `find(X) :- generate(X), test(X).`

- generate all the possible solutions, then test each one to see if it is the right answer.

Nondeterminstic programming

- This is a general template for writing nondeterministic programs:

- `find(X) :- generate(X), test(X).`

  - generate all the possible solutions, then test each one to see if it is the right answer.

  - Could we have implemented the zebra puzzle this way?

Nondeterminstic programming

- This is a general template for writing nondeterministic programs:

- `find(X) :- generate(X), test(X).`

  - generate all the possible solutions, then test each one to see if it is the right answer.

  - Could we have implemented the zebra puzzle this way? yes!

Generalizing information

	Nationality	Drink	Colour	Cigarettes	Pet	Location
House 1 to 5	English	Coffee	Green	Old Gold	Dog	1
House 1 to 5	Spaniard	OJ	Red	Parliaments	Horse	2
House 1 to 5	Ukrainian	Milk	Yellow	Kools	Snails	3
House 1 to 5	Norwegian	Tea	Ivory	Lucky Strike	Fox	4
House 1 to 5	Japanese	Water	Blue	Chesterfield	Zebra	5

We know what goes under each column but we don't know in what order...

we can generate all the permutations of a column against other columns, and test the constraints on each solution.

Nondeterminstic programming

- implement this approach on your own:

- ```
  find(X) :-
  generate_all_possibilites_for_hou
  ses(X), test_the_constraints(X).
  ```
  ```

Nondeterminstic programming

- implement this approach on your own:

- ```
  find(X) :-
  generate_all_possibilites_for_hous
  es(X), test_the_constraints(X).
  ```

- The problem is this approach is computationally extremely expensive.
  ```

Nondeterminstic programming

- What your book suggests for improving the efficiency of nondeterministic programs is "pushing the tester into the generator."

Nondeterminstic programming

- What your book suggests for improving the efficiency of nondeterministic programs is "pushing the tester into the generator."

- And that is exactly what we've done in our implementation of zebra: instead of generating a complete but arbitrary potential solution and then testing it (how many arbitrary solutions for the 5 houses are there?), we start off with an incomplete solution (i.e. full of unknown variables), then gradually apply each constraint.

Nondeterminstic programming

- What your book suggests for improving the efficiency of nondeterministic programs is "pushing the tester into the generator."
- And that is exactly what we've done in our implementation of zebra: instead of generating a complete but arbitrary potential solution and then testing it (how many arbitrary solutions for the 5 houses are there?), we start off with an incomplete solution (i.e. full of unknown variables), then gradually apply each constraint.
- Prolog is still generating then testing, but the scope of what it tests is now much more narrowed.
- On your own: using the time predicate run the first implementation and the nondeterministic generate-and-test implementation and see the difference in efficiency.

Nondeterminstic programming

- I'll upload another example of this type of problem solving (with sudoku this time) to piazza for you to study.

Questions

Next Class

- On Thursday we will talk about Database Manipulation, some built-in extra-logical predicates and some second-order predicates.

- If there is time we will also talk about Definite Clause Grammars and Natural Language Processing in Prolog. (Ch. 19)