

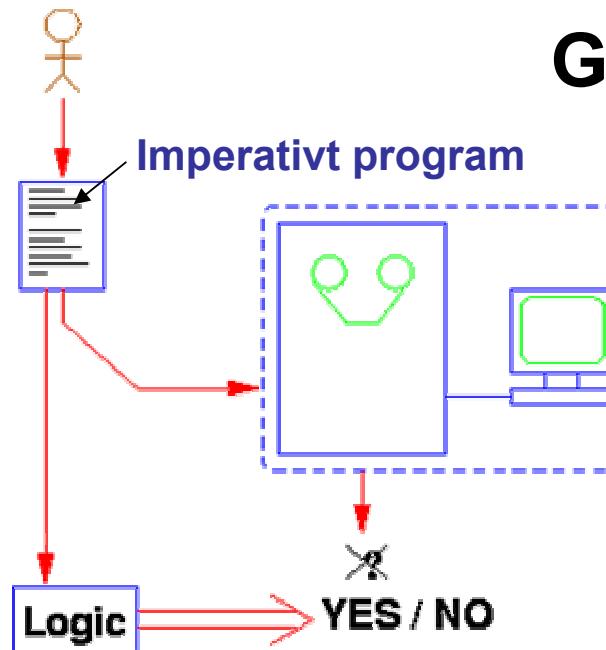
Logisk programmering

- Mitchell kapittel 15
- Grunnleggende ideer
- Programmeringsspråket Prolog
 - Fakta, regler og spørninger
 - Interferens-mekanismen, unifikasjon
 - Lister

Paradigmes/perspectives

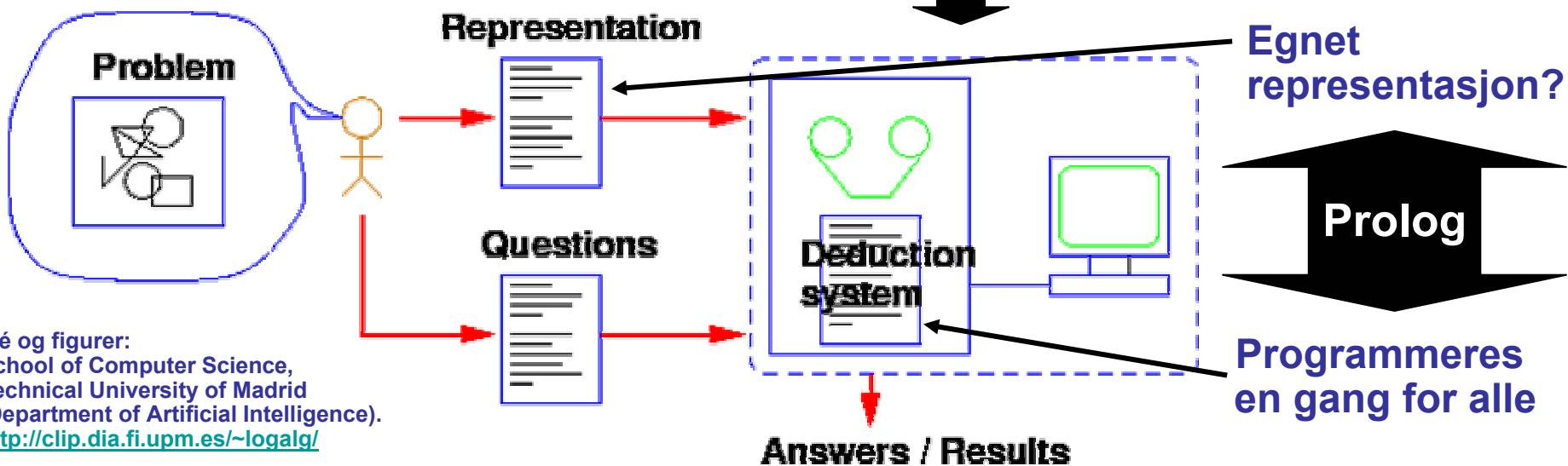
- Procedural/imperative Programming
 - A program execution is regarded as a sequence of operations manipulating a set of registers (programmable calculator)
- Functional Programming
 - A program is regarded as a mathematical function
- Constraint-Oriented/Declarative (Logic) Programming
 - A program is regarded as a set of equations
- Object-Oriented Programming
 - A program execution is regarded as a physical model simulating a real or imaginary part of the world

Grunnleggende idé



Logisk tenkning kan brukes for å resonnere rundt eller bevise riktigheten av et imperativt program

Men hvorfor ikke uttrykke problemet direkte i logiske termer og la maskinen utlede løsninger?



Idé og figurer:
School of Computer Science,
Technical University of Madrid
(Department of Artificial Intelligence).
<http://clip.dia.fi.upm.es/~logalg/>

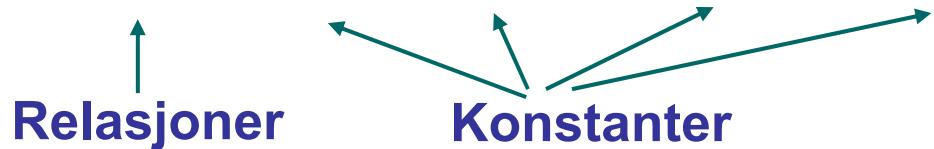
Programmeringsprinsippet – Hva istedet for Hvordan

- Vi programmerer ved å lage en (formell) verden som vi undersøker.
- Programmeringen har to faser:
 - Beskrive den formelle verden.
 - Stille spørsmål om den formelle verden (Prolog svarer)
- Beskrivelsen av den formelle verden består av
 - Fakta: Basale sannheter (``database").
 - Regler: Hvordan splitte et problem i delproblemer.
- Prolog svarer på spørsmålene ved å bruke reglene og faktaene.

Prolog eksempel - Familieforhold

- Fakta
- Vi lar person(a,b,c,d) angi en person med navn a, med b som mor, c som far og d som fødselsår.

```
person(anne, aase, aale, 1960).  
person(knut, aase, aamund, 1965).  
person(lars, aase, aale, 1962).  
person(beate, anne, arne, 1989).
```



- Konstanter: ord som starter med liten bokstav, samt tall
- Relasjoner: ord som starter med liten bokstav

Prolog eksempel – Familieforhold, spørsmål

- Spørsmål
 - | ?- person(anne, aase, aale, 1960) .

yes

- | ?- person(anne, aase, aale, 1962) .

no

- Merk det siste svaret:
Prolog opererer i en lukket verden.
Dersom et faktum ikke finnes, svarer Prolog no, ikke "vet ikke".

Spørsmål med variable

- **Variabel:** Ord som starter med stor bokstav eller med _
- Bruk av variable i spørsmål:
 - Leter gjennom kunnskapsbasen til det er noe som passer ved innsetting for variablene (unifikasjon).
 - Som svar returneres det som blir satt inn.

```
| ?- person(anne, aase, aale, Aar).
```

Aar = 1960 ? ;

no

```
| ?- person(Barn, aase, aale, Aar).
```

Aar = 1960,

Barn = anne ? ;

Aar = 1962,

Barn = lars ? ;

no

Variabel

Let etter flere løsninger

Unifikasjon

- Unifikasjon: å matche (del-)spørsmål med fakta/regler
- For at vi skal få en match må vi ha:
 - samme relasjon ytterst
 - samme antall argumenter
 - for hvert argument:
 - begge er konstater: ok hvis samme konstant
 - en (ubundet) variabel X og en konstant c:
X må bindes til c.
 - to variable X og Y: Y erstattes med X

Sammensatte spørninger

- Sammensatte spørsmål kan bygges opp med komma og semikolon:
 - komma betyr "og"
 - semikolon betyr "eller"

```
| ?- person(lars, aale, Far, Aar);  
    person(lars,Mor,aale,Aar).
```

Aar = 1962,

Mor = aase ? ;

no

Regler

barn(X, Y) skal bety at X er barn av Y:

```
barn(X, Y) :- person(X, Y, Z, U).  
barn(X, Y) :- person(X, Z, Y, U).
```

:- leses som "hvis"

Vi kan så stille spørsmål som involverer barn-relasjonen:

```
| ?- barn(lars, aale).
```

yes

```
| ?- barn(lars, Forelder).
```

Forelder = aase ? ;

Forelder = aale ? ;

no

Variablene s scop

- Skopet til en variabelforekomst er regelen den forekommer i.
 - Samtlige forekomster av en variabel i en regel avhenger av hverandre.
 - To ulike regler er helt uavhengige.
- Variabelnavnene er vilkårlige, men unngå misforståelser!

Finne svar på spørsmål

```
| ?- barn(lars,aale).
```

For relasjonen `barn` har vi to mulige regler å bruke:

```
person(lars,aale,Z,U).
```

Passer ikke med noe faktum.

```
person(lars,Z,aale,U).
```

Passer med faktum `person(lars,aase,aale,1962)`.

```
| ?- barn(lars,Forelder).
```

To muligheter:

1. `person(lars,Forelder,Z,U)`.

Passer med `person(lars,aase,aale,1962)`, det vil si
Forelder = aase.

2. `person(lars,Z,Forelder,U)`.

Passer med `person(lars,aase,aale,1962)`, det vil si
Forelder = aale.

Vi har altså to løsninger her.

Regler med flere betingelser

- sosken(X,Y) skal angi at X og Y er søsknen:
`sosken(X, Y) :- barn(X, Z), barn(Y, Z), X \== Y.`
- Komma separerer betingelser som alle må være oppfylt.
- Betingelsen $X \neq Y$ angir at X og Y må være tekstlig ulike. Dette kravet gjør at sosken(anne,anne) gir "no".

```
| ?- sosken(anne,X).
```

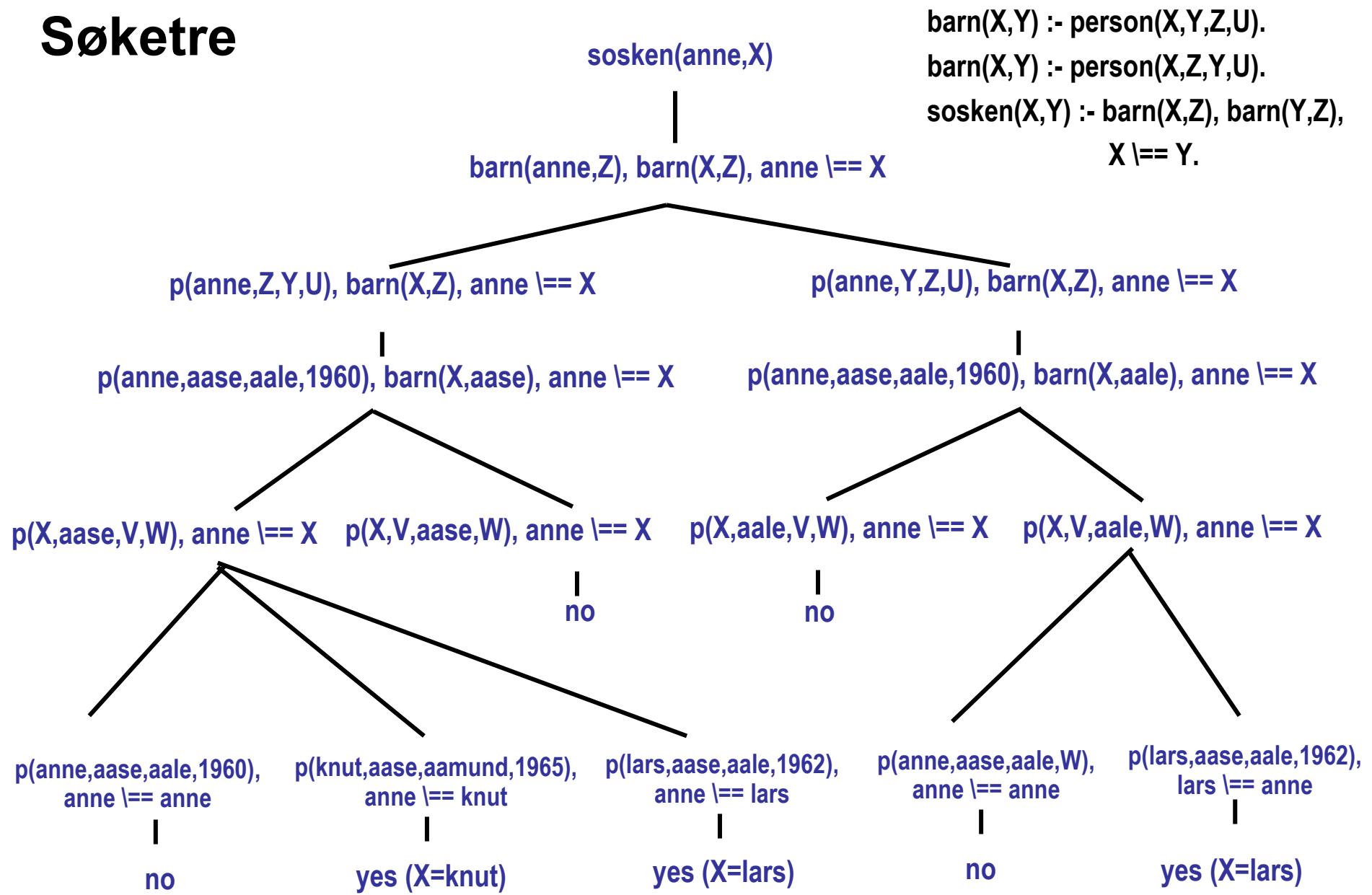
```
X = knut ? ;
```

```
X = lars ? ;
```

```
X = lars ? ;
```

```
no
```

Søketre



Flere regler (ekte søsken)

- `esosken(X,Y)` angir at X og Y er "ekte" søsken.

```
esosken(X,Y) :- barn(X,Forelder1),  
                  barn(Y,Forelder1),  
                  X \== Y,  
                  barn(X,Forelder2),  
                  barn(Y,Forelder2),  
                  Forelder1 \== Forelder2.
```

Flere regler (halvsøsker)

- halvsosken(X,Y) angir at X og Y er halvsøsker.

```
halvsosken(X, Y) :- barn(X, Forelder),  
                  barn(Y, Forelder),  
                  X \== Y,  
                  barn(X, Forelder1),  
                  barn(Y, Forelder2),  
                  Forelder \== Forelder1,  
                  Forelder \== Forelder2,  
                  Forelder1 \== Forelder2.
```

Spørninger – søsken og halvsøsker

```
| ?- esosken(anne,X) .  
  
X = lars ? ;  
  
X = lars ? ;  
  
no  
| ?- esosken(X, anne) .  
  
X = lars ? ;  
  
X = lars ? ;  
  
no  
| ?- halvsosken(anne,X) .  
  
X = knut ? ;  
  
no  
| ?- halvsosken(X, anne) .  
  
X = knut ? ;  
  
no  
| ?-
```

Rekursive regler

- etterkommer(X,Y) skal angi at X er etterkommer til Y:
`etterkommer (X, Y) :- barn (X, Y) .
etterkommer (X, Y) :- barn (X, Z) , etterkommer (Z, Y) .`
- Rekkefølge:
 - Ikke-rekursiv regel først!
 - Rekursivt delmål til slutt!

```
| ?- etterkommer (anne, X) .  
X = aase ? ;  
X = aale ? ;  
no  
| ?- etterkommer (X, aase) .  
X = anne ? ;  
X = knut ? ;  
X = lars ? ;  
X = beate ? ;
```

Lister i Prolog

- [] – den tomme listen
- [a,b,c] – en liste med tre elementer
- [a | [b,c]] – samme som [a,b,c]
- Generelt: [X | Y] er en liste med X som første element og Y som restliste.
- Altså:

[first, second, third] = [A | B]

unifiserer til

A = first og B=[second, third]

Unifikasjon av lister

- [a,b,c] unifiserer med [Head | Tail] , Resultat: Head=a og Tail=[b,c]
- [a] unifiserer med [H | T] , Resultat: H=a og T=[]
- [a,b,c] unifiserer med [a | T] , Resultat T=[b,c]
- [a,b,c] unifiserer ikke med [b | T]
- [] unifiserer ikke med [H | T]
- [] unifiserer med []. To tomme lister matcher alltid!

Eksempler på unifikasjon av lister

- Anta at vi har følgende faktum: $p([H|T], H, T)$.
- La oss gjøre noen enkle spørninger:

```
?- p([a,b,c], X, Y).
```

X=a

Y=[b,c]

yes

```
?- p([a], X, Y).
```

X=a

Y=[]

yes

```
?- p([], X, Y).
```

no

Øving: Unifikasjon av lister

- I noen av de følgende eksemplene kan unifikasjonen gjennomføres, i andre ikke. Hvilke?

Skriv ned substitusjonene for de eksemplene der unifikasjonen kan gjennomføres .

- [a, d, z, c] og [H | T]
- [apple, pear, grape] og [A, pear | Rest]
- [a | Rest] og [a, b, c]
- [a, []] og [A, B | Rest]
- [One] og [two | []]
- [one] og [Two]
- [a, b, X] og [a, b, c, d]

Søke i lister

- Fremgangsmåte:
 1. Sjekk om listehodet er det søkte elementet
 2. Hvis ikke, kast listehodet og gjenta prosessen på resten av listen.
 3. Søket stopper når vi har funnet elementet, eller når vi forsøker å søke i den tomme listen.
- Prolog-program:

```
finn(Element, [Element | Rest]) . /* det søkte elementet først i listen? */  
finn(Element, [DisregardHead | Tail]) :-  
    finn(Element, Tail) .
```

Bygge lister

Fremgangsmåte:

- Ta en eksisterende liste Liste1 og et element Element.
Lag en liste Liste2 ved unifikasjon med $\text{Element} | \text{Liste1}$
- Prolog-program:

```
Liste2 = [Element | Liste1].
```

Skjøte sammen lister

- Vi skal lage en relasjon som skjøter sammen to lister L1 og L2 til L3:
?- append([forste, andre, tredje], [fjerde, femte], Resultat).
skal gi Resultat = [forste, andre, tredje, fjerde, femte]
- Fremgangsmåte:
 1. Sjekk om den første listen er tom.
 2. Del den første listen (rekursivt) opp i enkeltelementer
– til slutt står vi med det siste elementet i listen.
 3. På vei opp av rekursjonen, føy enkeltelementene til i forkant av den andre listen – start med det siste enkeltelementet.
- Prolog-program

```
append( [ ], List, List ) .  
append( [Head|Tail], List2, [Head|Result] )  
      :- append(Tail, List2, Result) .
```

Funksjoner ?

- Prolog har ikke funksjoner!
- Funksjonen $f: A \rightarrow B$ kan erstattes med relasjonen $\text{erf}(a,b)$ slik at $\text{erf}(a,b)$ svarer til $f(a) = b$.
- Eksempel:
`append(Liste1, Liste2, Resultat).`
Her kan vi tenke på Liste1 og Liste2 som "inn-parametre", og Resultat som "ut-parameter".

Fjerne elementer i lister

- Eksempel: Fra listen [1,12,3,14,5,8], lag en ny liste der elementer mindre eller lik 6 er fjernet:

```
sift([1,12,3,14,5,8], Result).  
skal gi Result = [12,14,8]
```

- Fremgangsmåte
 - Sjekk om listen er tom. I så fall er resultatlisten også tom.
 - Sjekk om hodet tilfredsstiller kriteriet. Hvis ja, legg det til resultatlisten.
 - Hvis ikke, kast listehodet og gjenta prosessen på resten av listen.
- Prolog-program:

```
sift([], []).  
sift([X|T], [X|Result]) :- X > 6, /* er X større enn 6 ? */  
                      sift(T, Result). /* hvis ja, prøv å finn flere */  
  
sift([DisregardHead|Tail], Result) :- sift(Tail, Result). /* Kast hodet, søk i resten */
```

Anonyme variable

- Den anonyme variabelen `_` passer med hva som helst og brukes når vi ikke er interessert i verdien, bare at den finnes.
Hver forekomst er en "ny" variabel!

- Eksempel:

I regelen

```
sift([DisregardHead|Tail],Result) :-  
    sift(Tail,Result).
```

er vi egentlig ikke interessert i DisregardHead.

Derfor kan vi istedenfor skrive

```
sift([_|Tail],Result) :- sift(Tail,Result).
```

Dermed unngår vi også advarsler om "singletons" fra Prolog.

Logisk puslespill

Given the following information, who drinks water and who owns the zebra?

1. There are 5 houses in a line, each with husband, wife, pet, drink, color
2. The english husband lives in the red house
3. The spanish husband owns the dog
4. Coffee is drunk in the green house
5. The ukrainian husband drinks tea
6. The green house is immediately to the left of the ivory house
7. Eve owns snails
8. Sue lives in the yellow house
9. Milk is drunk in the middle house
10. The norwegian husband lives in the first house on the left
11. Cindy lives next to the house with the fox
12. Sue lives in the house next to the house with the horse
13. Laura drinks orange juice
14. The japanese is married to Paula
15. The norwegian husband lives next to the blue house

puzzledata(H, W, Z) :-

```

    same(H, [house(norwegian, _, _, _, _), _, house(_, _, _, milk, _), _, _]),      % 1,10,9
    member(house(englishman, _, _, _, red), H),                                         % 2
    member(house(spaniard, _, dog, _, _), H),                                         % 3
    member(house(_, _, _, coffee, green), H),                                         % 4
    member(house(ukrainian, _, _, tea, _), H),                                         % 5
    iright(house(_, _, _, _, ivory), house(_, _, _, _, green), H),                   % 6
    member(house(_, eve, snails, _, _), H),                                         % 7
    member(house(_, sue, _, _, yellow), H),                                         % 8
    nextto(house(_, cindy, _, _, _), house(_, _, fox, _, _), H),                      % 11
    nextto(house(_, sue, _, _, _), house(_, _, horse, _, _), H),                     % 12
    member(house(_, laura, _, orange-juice, _), H),                                % 13
    member(house(japanese, paula, _, _, _), H),                                 % 14
    nextto(house(norwegian, _, _, _, _), house(_, _, _, _, blue), H),                % 15

    member(house(W, _, _, water, _), H),                                              % Q1
    member(house(Z, _, zebra, _, _), H).                                              % Q2

```

member(X, [X|Y]).

member(X, [_|Y]) :- member(X, Y).

iright(LEFT, RIGHT, [LEFT, RIGHT|_]).

iright(LEFT, RIGHT, [_|REST]) :- iright(LEFT, RIGHT, REST).

same(X, X).

nextto(X,Y, [X,Y|_]).

nextto(X,Y, [Y,X|_]).

nextto(X,Y, [_|List]) :- nextto(X,Y, List).

<http://www.cs.oberlin.edu/classes/rit/plc/lab7/lab73.html>