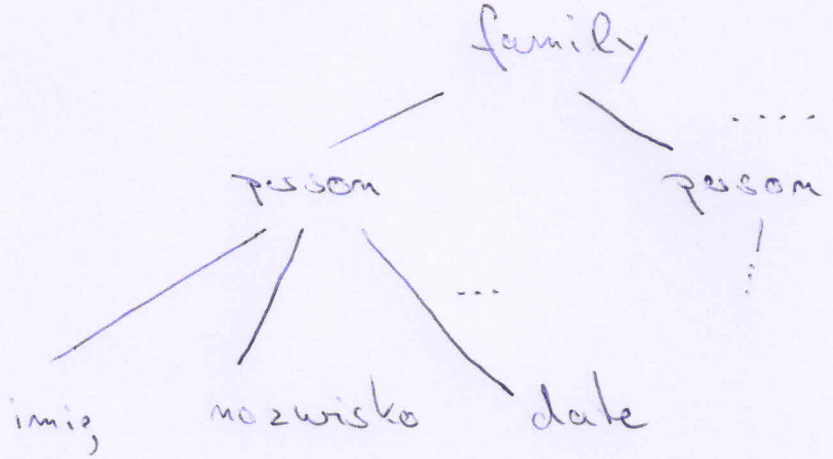
§4 Programowanie z termami

4.1 Informacje w bazach danych




```
family(
    person(tom, smith, date(7, may, 1950), unemployed),
    person(anne, smith, date(8, may, 1955),
                                works(IBM, 200.000)),
    [person(jack, smith, date(15, april, 1980), unemployed),
     person(pete, smith, date(28, nov, 1983), unemployed)]
```

obiekty (informacje) na podstawie struktury:

```
family(_, X, [_,_,_]).
family(person(Name, Surname, _, _), _, [_]).
```

```
wife(X) :- family(_, X, _).
child(X) :- family(_, _, Children),
            member(X, Children).

date_of_birth(person(_, _, Date, _), Date).

salary(person(_, _, _, works(_, S)), S).
salary(person(_, _, _, unemployed), 0).

total([], 0).
total([Person|R], Sum) :-
        salary(Person, S),
        total(R, S1),
        Sum is S + S1.

?- family(H, W, C),
   total([H, W|C], Income).
```

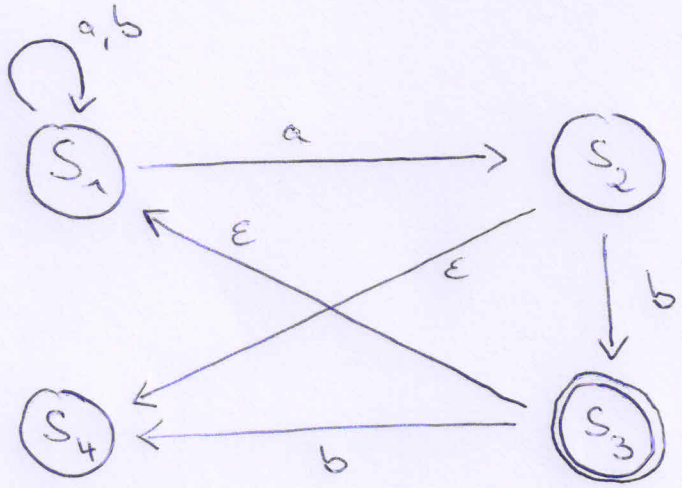
Data Abstraction

```
husband(family(Husband, _, _), Husband).
children(family(_, _, Children), Children).
firstchild(Family, FirstChild) :-
        children(Family, [FirstChild|_]).
```

↳ "konstruktory"

firstname(P1, tom), surname(P1, smith),
firstname(P2, peter), surname(P2, smith),
husband(Family, P1),
secondchild(Family, P2).

4.2 Automaty



predykaty: final(S),
trans(S_1, X, S_2),
silent(S_1, S_2)

final(s3).
trans(s_1, a, s_1).
trans(s_1, a, s_2).
⋮
silent(s_3, s_1).
silent(s_2, s_4).

```
accepts ( State, [] ) :- final ( State ).
accepts ( State, [X|R] ) :-
        trans ( State, X, State1 ),
        accepts ( State1, R ).
accepts ( State, String ) :-
        silent ( State, State1 ),
        accepts ( State1, String ).
```

```
?- accepts ( s1, [a,a,a,b] ).
   yes
```

```
?- accepts ( S, [a,b] ).
   S = s1 ;
   S = s3
```

```
?- accepts ( s1, [X1, X2, X3] ).
```

4.3 Problem 8 hetmanów

```
?- solution ( Positions ).
   Positions = [1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1]
```

↳ $[1/Y_1, 2/Y_2, \ldots 8/Y_8]$

bardziej ogólne niż 8×8 !

(i) zero hetmanów
pusta lista jest rozwiązaniem

(ii) [X/Y | Others]
- nie ma ataku w Others, czyli Others jest rozwiązaniem
- X, Y ∈ {1, ... 8}
- hetman w X/Y nie atakuje hetmana w Others

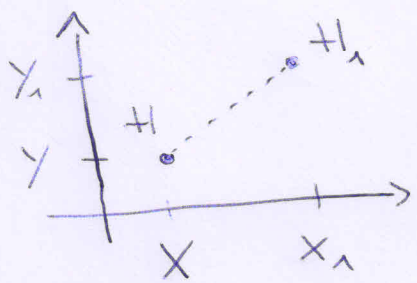
```
solution([]).
solution([X/Y|Others]):-
    solution(Others),
    member(Y, [1,2,3,4,5,6,7,8]),
    noattack(X/Y, Others).

noattack(H1, HList)
```

(i) zero hetmanów (HList = [])
True

(ii) [H1 | HList1]
- H1 nie atakuje H1
- H1 nie atakuje hetmana w HList1

(iii) H nie atakuje H1, jeżeli

- Y i Y_1 są różne
- odległość w kierunku X i odległość w kierunku Y są różne.




```
noattack(_, []).
noattack(X/Y, [X1/Y1|Others]) :-
        Y =\= Y1,
        Y1 - Y =\= X1 - X,
        Y1 - Y =\= X - X1,
        noattack(X/Y, Others).

template([1/Y1, 2/Y2, ....., 8/Y8]).

?- template(Pos), solution(Pos).
   Pos = [ ..... ];
   Pos = [ ..... ];
     :
```