Week 1

- Prolog = Programming with logic
- Declarative, not procedural, berbasis logika matematis
- 1972
  - First prolog interpeter by Colmerauer & Roussel di Aix-Marseille Université, Perancis
- 1977
  - Implementation of DEC10 compiler by Warren
- 1980
  - Define Clouse Grammars implementation by Pereira & Warren
- 1980s/1990s
  - Grows popularity in Europe & Japan
- 2005
  - Program natural language interface in International Space Station by NASA.

- Basic idea of Prolog:
  - Describe situation of interest
  - Ask a question
  - Prolog logically deduces new facts about situation we described
  - Prolog gives its deductions back as answers

- Konsekuensinya adalah think declaratively, not procedurally, perlu mindset baru, dan merupakan high-level language, ga seefisien C, serta berguna ke AI.

- Contoh Knowledge Base:
  -
```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```
fact

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
```

```
?- tattoed(jody).
ERROR: predicate tattoed/1 not defined.
?- party.
yes
```

  -
```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```
fact, fact, rule, rule, rule

head: playsAirGuitar(yolanda)

body: listens2music(yolanda)

- 3 predikat yaitu happy, listens2music, & playsAirGuitar
- 5 clause: 2 facts, 3 rules
- , is conjunction / AND

  - playsAirGuitar(butch):- happy(butch).
    playsAirGuitar(butch):- listens2music(butch).

sama seperti

playsAirGuitar(butch):- happy(butch); listens2music(butch).

- Operators:
  - Implication → :-
  - Conjunction → ,
  - Disjunction → ;

```
woman(mia).
woman(jody).
woman(yolanda).
loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

woman/1

loves/2

```
?- loves(marsellus,X), woman(X).
X=mia
yes
```

?- woman(X).
X = mia;
X = jody;
X = yolanda;
no

?- loves(pumpkin, X), woman(X).
no

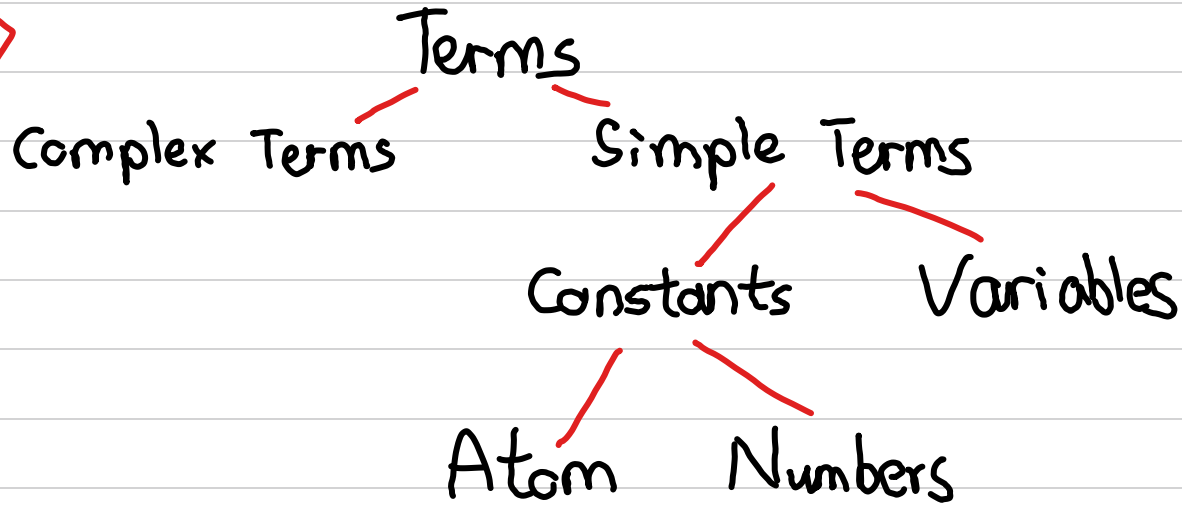
```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

loves/2

```
?- jealous(marsellus,W).
W=vincent
?-
```

b)



- **Atoms**
  - Mulai dari huruf kecil / single quotes
  - Boleh mengandung uppercase/ lowercase/ digits / underscore
  - Special character ; : . :-
- **Numbers**
  - Integers (14, -2)
  - Float (32.5), (+12.5)

compound

- **Variables**
  - Mulai dari huruf besar / underscore
  - Boleh mengandung uppercase/ lowercase/ digits / underscore
- **Complex Terms**
  - Atoms, numbers, variables are building blocks for complex terms.
  - Dibuat dari functor dengan argumentnya
  - Arguments are put in (), seperate dengan ,
  - Functor harus atom
- Number of argument dari complex term = Arity
- Prolog can define 2 predikat with same functor with different arity
- Paradigma pemrograman = Gaya dasar dalam melalcukan pemrograman
- Kategori
  - Imperatif
    - ↳ Procedural, Object-oriented
  - Declaratif
    - ↳ Functional, Logic
- **Paradigma Imperatif**
  - Menekankan bagaimana cara mengeksekusi komputasi sebagai suatu rangkaian aksi
  - Aksi berupa operasi aritmetika (+, -, *, /) atau logika ( ), =, <, OR, AND)
  - Aksi dikendalikan dengan struktur seleksi (if else) atau repetisi (for, while, repeat)

C, Java, Pascal, C++, Fortan

- Program merupakan rangkaian aksi untuk mengubah nilai dari sejumlah variabel.

- **Paradigma Deklaratif**
  - Menekankan apa yang dieksekusi dalam komputasi
  - Terdiri dari fact & rule
  - Komputasi merupakan deduksi / inferensi (Pengambilan keputusan secara logis)
  - Tidak melakukan assignment sebagai operasi dasar

  Prolog, Haskell, SQL

Imperatif vs Deklaratif

Tabel_Nilai_Mhs

NIM	Nama	Nilai
01	Agus	93
03	Budi	56
08	Charles	74

Hitung Nilai Rerata !

HOW

By Imperative :

```
Open Tabel_Nilai_Mhs
While Not EOF(Tabel_Nilai_Mhs)
  Nilai = Nilai + Tabel_Nilai_Mhs.Nilai
  JumlahMhs = JumlahMhs + 1
End While
NilaiRerata = Nilai / JumlahMhs
```

ASSIGNMENT

By Declarative :

```
SELECT AVE(Nilai) FROM Tabel_Nilai_Mhs
```

WHAT

Prolog

Pros	Cons
Komputasi simbolik (Non-numerik) Pemecahan masalah terkait kondisi suatu objek / relasi antar sejumlah objek Proses reasoning / penalaran yang banyak dipakai di AI	Tidak cocok untuk komputasi yang • Need presisi numerik tinggi (Komputasi grafis) • Need portabilitas tinggi (Client server processing)

Week 2

- **Unification** : 2 terms unify if they are the same term or if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal.

  → **Revised Definition** :
  - If T_1 and T_2 are constants, then T_1 and T_2 unify if they are the same atom, or the same number.
  - If T_1 is a variable and T_2 is any type of term, then T_1 and T_2 unify, and T_1 is instantiated to T_2, vice versa
  - If T_1 and T_2 are complex terms then they unify if :
    a) they have same functor & arity
    b) all their corresponding arguments unify
    c) the variable instantiations are compatible

- ?- X=mia, X = vincent.
  ↳ Prolog membaca X sudah diisi dengan mia, jadi tidak dapat unify dengan vincent anymore.

- ?- k(s(g),Y) = k(X, t(k)).
  X = s(g)
  Y = t(k)
  yes

- ?- k(s(g), t(k)) = k(X, t(Y)).
  X = s(g)
  Y = k
  yes

- Infinite terms : ?- father(X) = X.
  X = father(father(....))
  yes

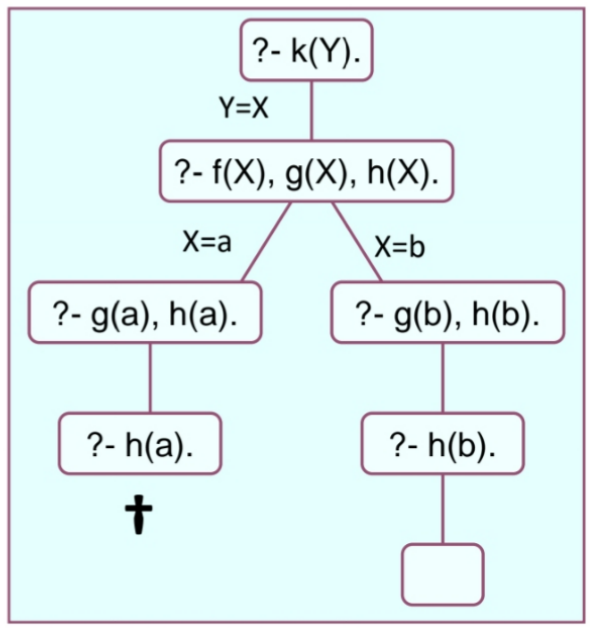
- unify_with_occurs_check() : Untuk unify variabel dengan term lain yang mengecek apakah variabel occurs di term tersebut
  ↳ ?- unify_with_occurs_check(father(x),x).
  no

a) Proof Search = Search knowledge base untuk melihat jika query tersebut satisfied.

a) Example: Search Tree

```
f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).
```

```
?- k(Y).
Y=b;
no
?-
```

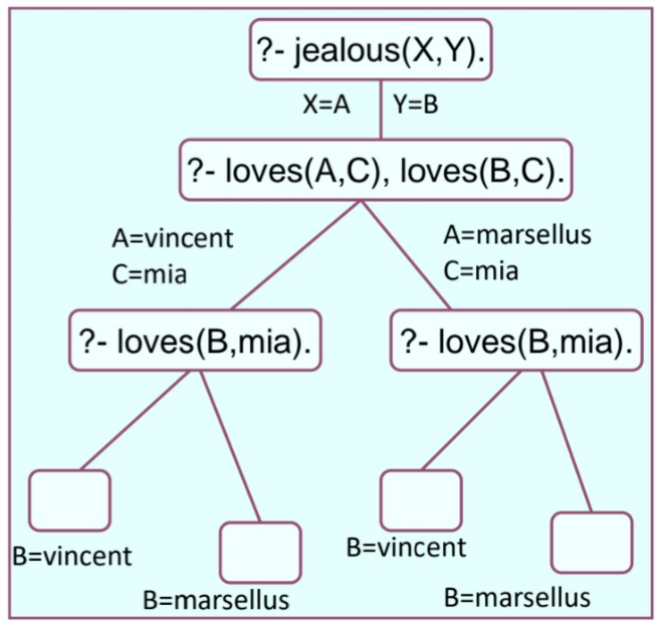


a)

```
loves(vincent,mia).
loves(marsellus,mia).

jealous(A,B):-
   loves(A,C),
   loves(B,C).
```

```
….
X=marsellus
Y=vincent;
X=marsellus
Y=marsellus
no
```



Week 3

a) A predicate is recursively defined if one or more rules in its definition refers to itself.

a) p :- p.
?- p.
ERROR. Out of memory

a)
```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).

descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

```
?- descend(anna,donna).
no
?-
```

a) Suppose use this way to write numerals:
1) 0 is a numeral
2) If X is a numeral, then so is succ(X)

a)
```
numeral(0).
numeral(succ(X)):- numeral(X).
```

```
?- numeral(X).
X=0;
X=succ(0);
X=succ(succ(0));
X=succ(succ(succ(0)));
X=succ(succ(succ(succ(0))))
```

a)
```
add(0,X,X).                    %%% base clause

add(succ(X),Y,succ(Z)):-      %%% recursive clause
    add(X,Y,Z).
```

```
?- add(succ(succ(0)),succ(succ(succ(0))), Result).
Result=succ(succ(succ(succ(succ(0)))))
yes
```

a) Prolog has spefic way of answering queries yaitu:
1) Search knowledge base from top to bottom
2) Processes clauses from left to right
3) Backtracking to recover from bad choices

9)

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).
```

```
descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
descend(X,Y):- child(X,Z), descend(Z,Y).
descend(X,Y):- child(X,Y).
```

```
?- descend(A,B).
A=anna
B=bridget
```

```
?- descend(A,B).
A=anna
B=emily
```

Week 4

- Lists = A list is a finite sequence of elements
  - ↳ Enclosed in square brackets
  - ↳ Length of a list = Number of elements it has
  - ↳ All sort of Prolog terms can be elements of a list
  - ↳ Special list : Empty list []

- Non-empty list terdiri atas head dan tail
  - head: first item di list
  - tail: sisa list selain head, yang juga sisanya adalah list

- Empty list gd head & tail.
  - Plays an important role in recursive predicates

- Built-in operator | use to decompose a list into its head & tail

- Underscore = Anonymous variables = Dipakai ketika pakai variable tapi ga ingin prolog instantiates to it.

- Member = Utk tau apakah sesuatu adalah elemen dari list / ga.

- The member/2 predicate works by recursively working its way down a list :
  - Doing something to head
  - Recursively doing same thing to tail

-
```
member(X,[X|T]).
member(X,[H|T]):- member(X,T).
```

```
?- member(zed,[yolanda,trudy,vincent,jules]).
no
?-
```

```
?- member(vincent,[yolanda,trudy,vincent,jules]).
yes
?-
```

6)

```
a2b([],[]).
a2b([a|L1],[b|L2]):- a2b(L1,L2).
```

```
?- a2b([a,a,a,a],[b,b,b,b]).
yes
?- a2b([a,a,a,a],[b,b,b]).
no
?- a2b([a,c,a,a],[b,b,b,t]).
no
?- a2b(X,[b,b,b,b,b,b,b]).
X = [a,a,a,a,a,a,a]
yes
?-
```

7) Empty list has length 0

7) Non-empty list has length 1+ length of its tail

Week 5

- Prolog provides a number of basic arithmetic tools berupa integer and real numbers.

Arithmetic	Prolog
2 + 3 = 5	?- 5 is 2+3.
3 x 4 = 12	?- 12 is 3*4.
5 – 3 = 2	?- 2 is 5-3.
3 – 5 = -2	?- -2 is 3-5.
4 : 2 = 2	?- 2 is 4/2.
1 is the remainder when 7 is divided by 2	?- 1 is mod(7,2).

?- X=3+2 / 2+3=X
X=3+2
yes
?- 2+3 is X
ERROR

- +, -, /, and * do not carry out any arithmetic
- Expression such as 3+2, 4-7, 5/5 are ordinary Prolog terms
  - ↳ Functor : +, -, *, /
  - ↳ Arity : 2
  - ↳ Arguments : Integers
- 3+2 is +(3,2), if want to count then ?- is(X, +(3,2)).
  X=5
  yes

```
len([],0).
len([_|L],N):-
   len(L,X),
   N is X + 1.
```

```
?- len([a,b,c,d,e,[a,x],t],X).
X=7
yes
?-
```

- Accumulators is a good program that is easy to understand and relatively efficient.

- Accumulators are variables that hold intermediate results.

- Defining acclen/3 :
  - Predicate has 3 arguments
  - Length of list is integer
  - An accumulator to keep track of the intermediate values for the length.
  - Initial value of accumulator is 0
  - Add 1 to accumulator each time we can recursively take head of the list
  - When reach empty list, accumulator contains length of list

```
acclen([],Acc,Acc).

acclen([_|L],OldAcc,Length):-
    NewAcc is OldAcc + 1,
    acclen(L,NewAcc,Length).
```

```
?-acclen([a,b,c],0,Len).
Len=3
yes

?-
```

- acclen/3 better than len/2 because acclen/3 is tail-recursive

- Difference :
  - In tail recursive predicates the results is fully calculated once we reach the base clause
  - In tail recursive predicates not tail recursive, there are still goals on the stack when we reach the base clause

Not tail-recursive

```
len([],0).
len([_|L],NewLength):-
    len(L,Length),
    NewLength is Length + 1.
```

Tail-recursive

```
acclen([],Acc,Acc).
acclen([_|L],OldAcc,Length):-
    NewAcc is OldAcc + 1,
    acclen(L,NewAcc,Length).
```

Comparing integers

Arithmetic	Prolog
$x < y$	X < Y
$x \leq y$	X =< Y
$x = y$	X =:= Y
$x \neq y$	X =\= Y
$x \geq y$	X >= Y
$x > y$	X > Y

- Ketika ingin buat predikat berargumen 2, akan true ketika:
  - 1st argument is list of integers
  - 2nd argument is highest integer of list
  - Basic ideanya adalah:
    - Use accumulator
    - Accumulator keeps track of highest value encountered so far
    - If ada lebih tinggi lagi nilainya, accumulator akan di-update.

```
accMax([H|T],A,Max):-
    H > A,
    accMax(T,H,Max).

accMax([H|T],A,Max):-
    H =< A,
    accMax(T,A,Max).

accMax([],A,A).

max([H|T],Max):-
    accMax(T,H,Max).
```

```
?- max([1,0,5,4], Max).
Max=5
yes

?- max([-3, -1, -5, -4], Max).
Max= -1
yes

?-
```

Week 6

- Declaratively, append (L_1, L_2, L_3) is true if List L_3 is the result concatenating the L_1 & L_2 together.

```
?- append([a,b,c,d],[1,2,3,4,5], X).
X=[a,b,c,d,1,2,3,4,5]
yes


?-
```

- Recursive in append/3 :
  - Base Clause : Appending the empty list to any list produces that same list.
  - The recursive step says that when concatenating a non-empty list [H|T] with a list L, the result = list with head and result of concatenating T and L.

```
?- append(X,Y, [a,b,c,d]).
X=[ ]          Y=[a,b,c,d];
X=[a]          Y=[b,c,d];
X=[a,b]        Y=[c,d];
X=[a,b,c]      Y=[d];
X=[a,b,c,d]    Y=[ ];
no
```

- Append juga bisa mencari prefix dan suffix di list
- prefix(P,L):- append(P,_,L).
  - List P is a prefix of some list L when there is some list such that L is result of concatenating P with that list.

```
prefix(P,L):-
    append(P,_,L).
```

```
?- prefix(X, [a,b,c,d]).
X=[ ];
X=[a];
X=[a,b];
X=[a,b,c];
X=[a,b,c,d];
no
```

- suffix(S, L):- append (_, S, L).
  - ↳ List S is a suffix of some list L when there is some list such that L is the result of concatenating that list with S.

```
suffix(S,L):-
    append(_,S,L).
```

```
?- suffix(X, [a,b,c,d]).
X=[a,b,c,d];
X=[b,c,d];
X=[c,d];
X=[d];
X=[];
no
```

- Sub-lists of a list L are simply the prefix of suffixes of L
  - ↳ sublist (Sub, List):- suffix(Suffix, List),
    prefix(Sub, Suffix).

- append/3 is inefficiency because concatenate a list isn't done in simple action, but by traversing down one of the lists.

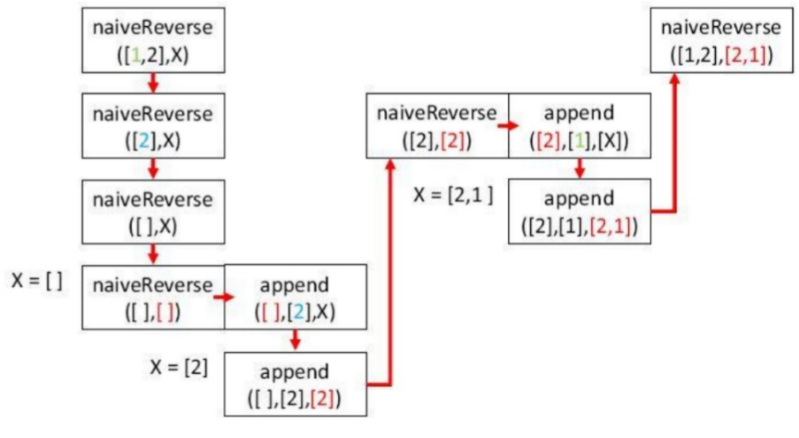
Naive Reverse

- If we reverse empty list, we obtain empty list
- If we reverse list [H|T], we obtain by reversing T and concatenating it with [H]

```
naiveReverse([],[]).
naiveReverse([H|T],R):-
    naiveReverse(T,RT),
    append(RT,[H],R).
```

kurang efektif, makanya pakai accumulator.

↓ menjadi



Use of reverse/2

```
accReverse([],L,L).
accReverse([H|T],Acc,Rev):-
    accReverse(T,[H|Acc],Rev).
reverse(L1,L2):-
    accReverse(L1,[],L2).
```

```
?- reverse([a,b,c,d], [d,c,b,a]).
true

?- reverse([a,b,c,d], [a,c,b,a]).
false
```