Lekcija 2

Lūgums uzmest aci uz sadaļu 1.2 no mācību grāmatas, sāksim runāt par predikātu valodām. Izlasiet sadaļu vismaz līdz lpp. 16 sākumam.

Pagājušajā lekcijā esam apskatījuši teoriju L, kuras ietvaros mēs esam spējīgi izveidot algoritmu kas pārbauda, vai dotais apgalvojums ir pierādāms teorijā L. Vai vispārīgi mēs varam izveidot programmu, kas kādai dotai teorijai pārbauda hipotēzi (tas ir, vai tā ir pierādāma)? Patiesībā tādas programmas izveidošana var būt ļoti sarežģīta, vai pat neiespējama. Lielākai daļai no matemātisko teoriju tāds algoritms nav iespējams.

Lekcijas laikā mēs apskatīsim grāmatas sadaļas daļu par predikātu valodām, lekcijas nobeigumā apskatīsim Prolog valodas interpretatoru kā zināšanas bāzes piemēru, kas izmanto predikātu valodas.

1.2. Predikātu valodas

Tika definētas 1870-1880-jos gados.

Apgalvojums "X strādā iestādē Y kā Z (amatā Z)": sanāk predikāts **strādā(X,Y,Z)**.

x=y (infiksa pieraksts) =(x,y) (prefiksa pieraksts). f(x,y)

Šāda pieeja - cilvēku valodas teikumu vienkāršošana uz mainīgajiem, konstantēm, funkcijām, predikātiem un kvantoriem, izrādās, ir pietiekami elastīga, un tajā pašā laikā ir daudz vienveidīgāka salīdzinājumā ar dabīgām cilvēku valodām. Vienotu pieeju ir daudz vieglāk izmantot sadarbībai ar datoriem.

Parasti matemātiķiem patīk definīcijas, bet datoriķiem – piemēri.

Te mums būs piemērs.

1) x,y,z,x_x,x_y,… - mainīgie

2) Pieejamas (atļautas) vērtības: Britney, John, Peter (to vēl sauc par vērtību apgabalu).

3) S(x), V(x), T(x,y), P(x,y) – sieviete, vīrietis, tētis, precējies, …, x=y

Tie ir predikāti. Bez x=y gandrīz neviena predikātu valoda nestrādā.

4) Loģika: ¬F, F∨G, F∧G, F→G, ∀F, ∃F

Programmētājiem parasti nepatīk implikācija, daudzās programmēšanas valodās tā nav paredzēta.

A	B	A→B
0	0	1
0	1	1
1	0	0
1	1	1

¬V(Britney) S(x)∧V(x) – vai šāds izteiciens ir jēdzīgs, ir atkarīgs no mūsu teorijas.

$T(x,y) \rightarrow V(x)$

$\exists x(T(x, \text{Britney}))$

$\forall x(V(x) \vee S(x))\ \forall x\neg(V(x) \wedge S(x))$

$\forall x \forall y1 \forall y2(M(y1,x) \wedge M(y2,x) \rightarrow y1=y2)$

Studenti mēdz izvairīties no implikācijas, šīs piemērs nav pareizs: $\forall x \forall y(T(x,y) \wedge V(x))$. Vajag implikāciju, lai pateiktu, ka ja persona ir tētis, tad tas ir vīrietis.

Piemērs, kur gan jālieto konjunkcija: "x ir y māsa". Pierakstot, ka x un y ir kopīgie vecāki, neuztraucieties par to, ka jāraksta $z1 \neq z2$ (ka vecāki ir divi dažādi cilvēki), pieņemsim, ka tā nav problēma.

$\exists z1 \exists z2(T(z1,x) \wedge T(z1,y) \wedge M(z2,x) \wedge M(z2,y) \wedge S(x))$

Kopsavilkumam – valodas primitīvi:

Objektu mainīgie – x,y,z,…

Objektu konstantes – apzīmē mūsu domēna objektus. Varam saukt vienkārši par konstantēm.

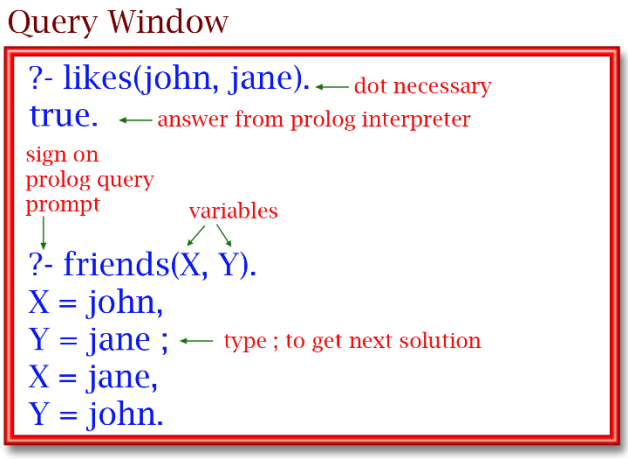
Funkciju konstantes – operācijas ar objektiem. Varam saukt vienkārši par funkcijām.

Predikātu konstantes – objektu īpašības vai attiecības starp objektiem. Varam saukt vienkārši par predikātiem.

Tātad mums ir: mainīgie, konstantes, funkcijas un predikāti.

Programmēšanas valoda Prolog kā piemērs, kur var veidot likumus un veikt vaicājumus par to, vai apgalvojums ir patiess dotās teorijas ietvaros.

Prolog = **Pro**gramming by **Log**ic



Lekcijā mēs apskatīsim sekojošu piemēru:

Saite, kurā mēs izmēģināsim mūsu kodu: https://swish.swi-prolog.org/

Mūsu programma:

```
male('John').
female('Lucy').
male('Peter').
likes('John','Lucy').
likes('Lucy','Peter').
person(X):-male(X).
person(X):-female(X).
```

Vaicājumu piemēri (mēģiniet minēt kādi būs iznākumi):

```
person('John')
person(X)
likes('John',X)
likes(X,'Peter')
```

Šeit mēs varam teikt, ka fakti

```
male('John').
female('Lucy').
male('Peter').
likes('John','Lucy').
likes('Lucy','Peter').
```

ir mūsu aksiomas, bet

```
person(X):-male(X).
person(X):-female(X).
```

ir izveduma likumi.