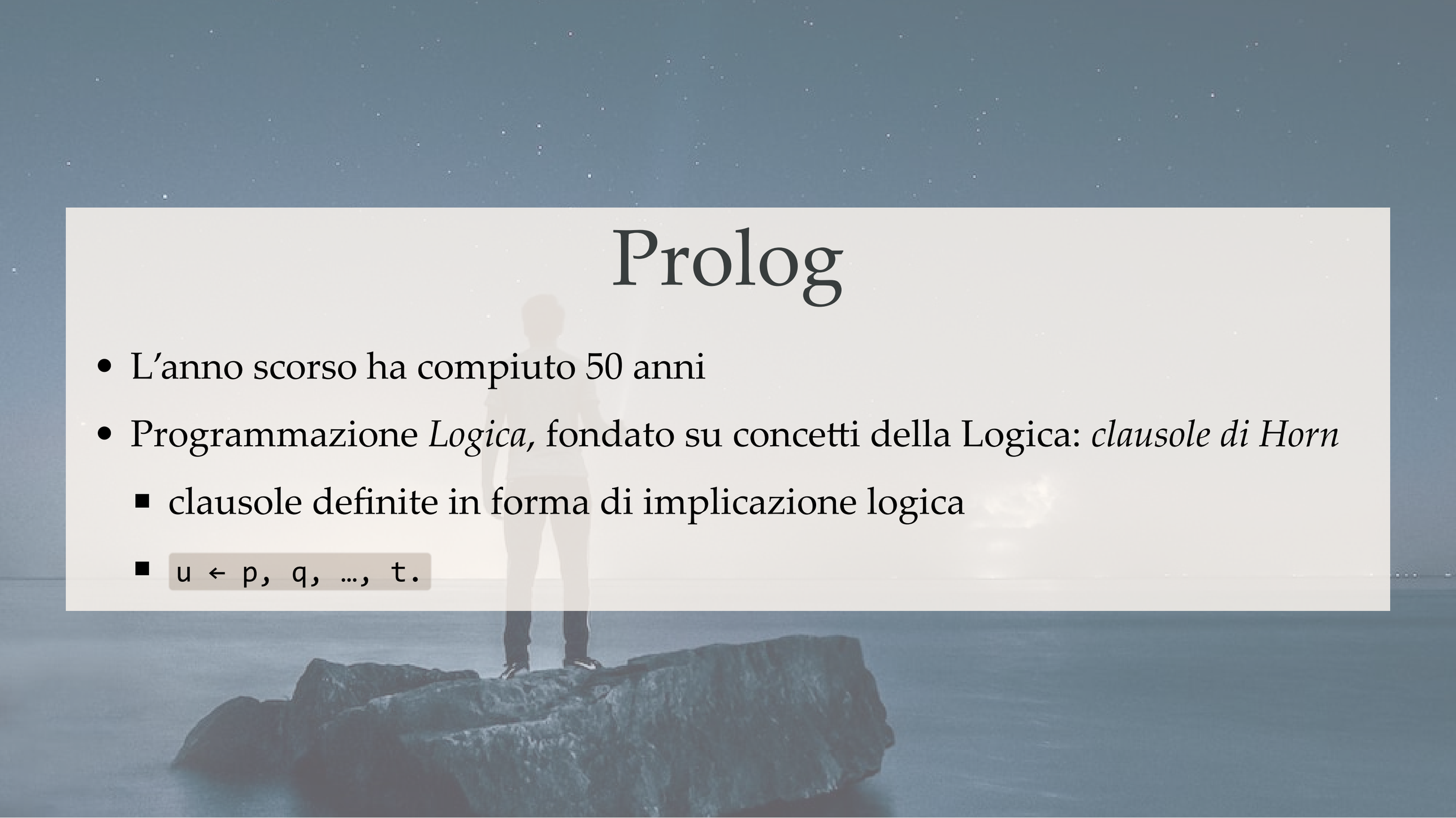
How To Have Fun

con Prolog

MuHackademy 2023

Agenda

- Introduzione di Prolog
- Knowledge rappresentation
- Constraint Logic Programming
- Scripting



Prolog

- L’anno scorso ha compiuto 50 anni
- Programmazione *Logica,* fondato su concetti della Logica: *clausole di Horn*
  - clausole definite in forma di implicazione logica
  - `u ← p, q, …, t.`

Programmi

- Tutti i dati, ed i programmi stessi, sono rappresentati come **termini**
- In Prolog il codice è *"data living in a database"*

Termini

- **Variabili**
  - Le variabili dinamicamente tipizzate
  - Iniziano con una lettera maiuscola o carattere underscore
- **Termini Atomici**
  - numeri, sequenze di caratteri dove il primo è minuscolo
  - sequenze di caratteri comprese tra singoli apici

```
1 %%%% VARIABILI %%%
2 A
3 _CNT
4
5 %%%% TERMINI ATOMICI %%%
6 ciao
7 5
8 sos
9 'Allerta'
```

Termini

- **Termini Composti**
  - `Fnome(Arg1, …, Argn)`
  - Ogni *argomento* è a sua volta un termine

```
1  %%% VARIABILI %%%
2  A
3  _CNT
4
5  %%% TERMINI ATOMICI %%%
6  ciao
7  5
8  sos
9  'Allerta'
10
11 %%% TERMINI COMPOSTI %%%
12 pazzesco(pane, mortadella)
13 nato('Carlo', data(15, 8, 1769))
```

Costrutti di un programma Prolog

- Fatti

```
cane('Toby').
gatto('Sharon').
parla('Luca', francese).
parla('Alessandro', russo).
parla('Gloria', tedesco).
```

- Regole
  - Una regola è strutturata:

    `Head` `:-` `Body`

```
intelligente(X):-
    persona(X),
    conosce(X, 'Prolog'),
    appartiene(X, 'MuHack').
```

Costrutti di un programma Prolog

- Fatti

```
cane('Toby'):- true.
gatto('Sharon'):- true.
parla('Luca', francese):- true.
parla('Alessandro', russo):- true.
parla('Gloria', tedesco):- true.
```

- Regole
  - Una regola è strutturata:

    `Head` `:-` `Body`

```
prelievo_banca(Somma, X, SaldoFinale)
    conto(X, Saldo),
    Somma < Saldo,
    SaldoFinale is Saldo - Somma.
```

Liste

- Struttura dati che permette di rappresentare sequenze, anche disomogenee, di termini
- Una lista può essere vuota → `[]`
- Mediante il carattere `|` è possibile rappresentare la **testa** e **coda** di una lista
  - `[Testa| Coda]`

```
?- L= [a, b, c, d], L= [H| T].
H= a,
T= [b, c, d].
```

Prolog Top-Level

- In questa presentazione interagiamo con il Top-Level del sistema prolog in uso: SWI-prolog
- Attraverso il Top-Level possiamo eseguire delle **interrogazioni**
- Il sistema Prolog risponde cercando di verificare se è **vera** la nostra interrogazione
- Se nell'interrogazione compaiono variabili cerca di trovare una istanziazione tale da rendere **vera** la nostra richiesta

Unificazione

- Meccanismo attraverso cui il *Sistema Prolog* effettua un processo di pattern matching che determina quali istanziazioni attuare nella risoluzione

Backtracking

Come si programma in Prolog?

- Si cerca di **descrivere** le relazioni tra le entità

Task: "*rimuovere un elemento da una lista*"

- In prolog **non** conviene ragionare in maniera operativa (procedurale)
- Conviene **descrivere/dichiarare** le relazioni tra le entità

Task: *"Mettere in relazione due liste, che sono uguali a meno di uno specifico elemento"*

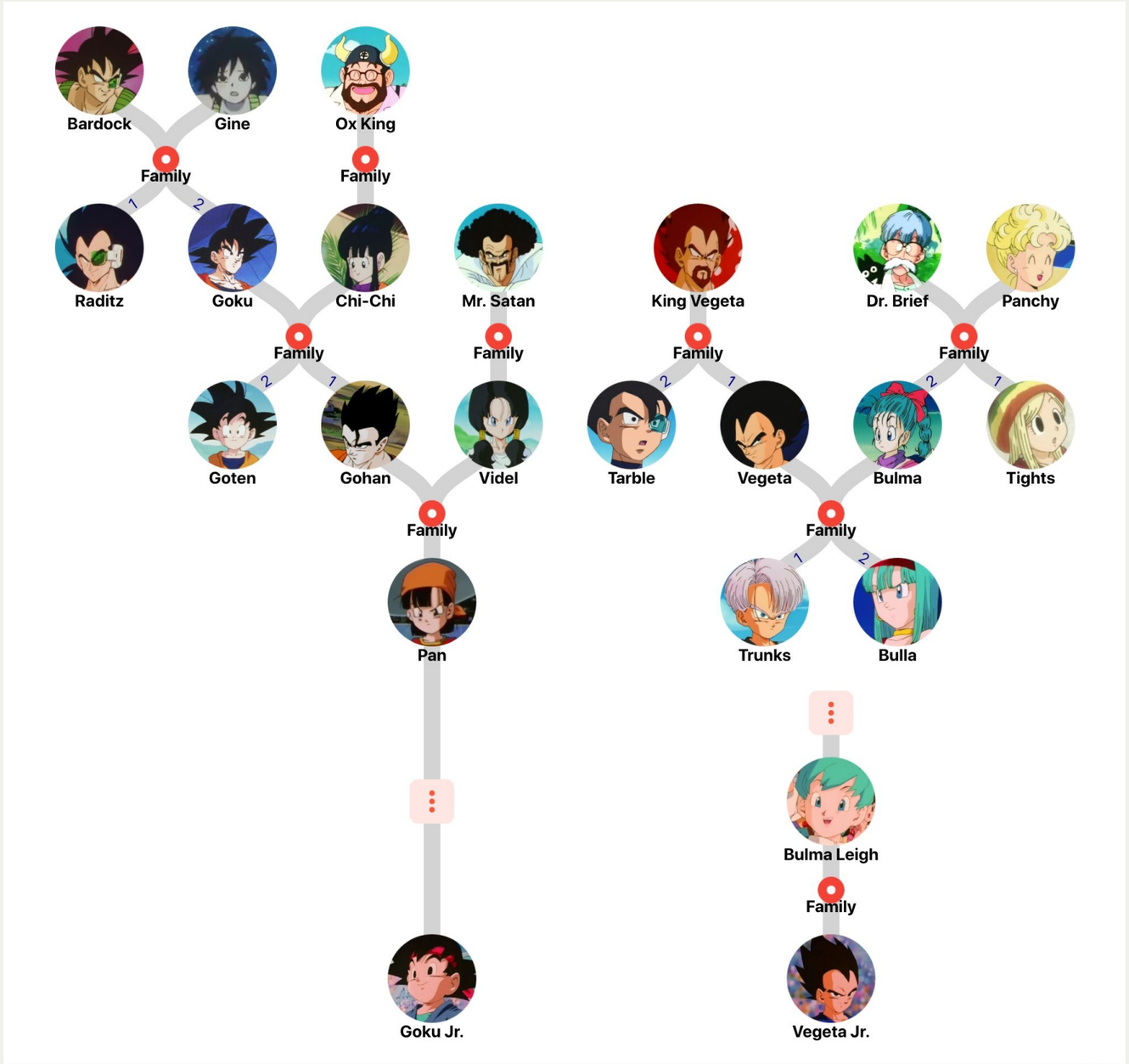
Come si programma in Prolog?

Task: *"Mettere in relazione due liste, che sono uguali a meno di uno specifico elemento"*

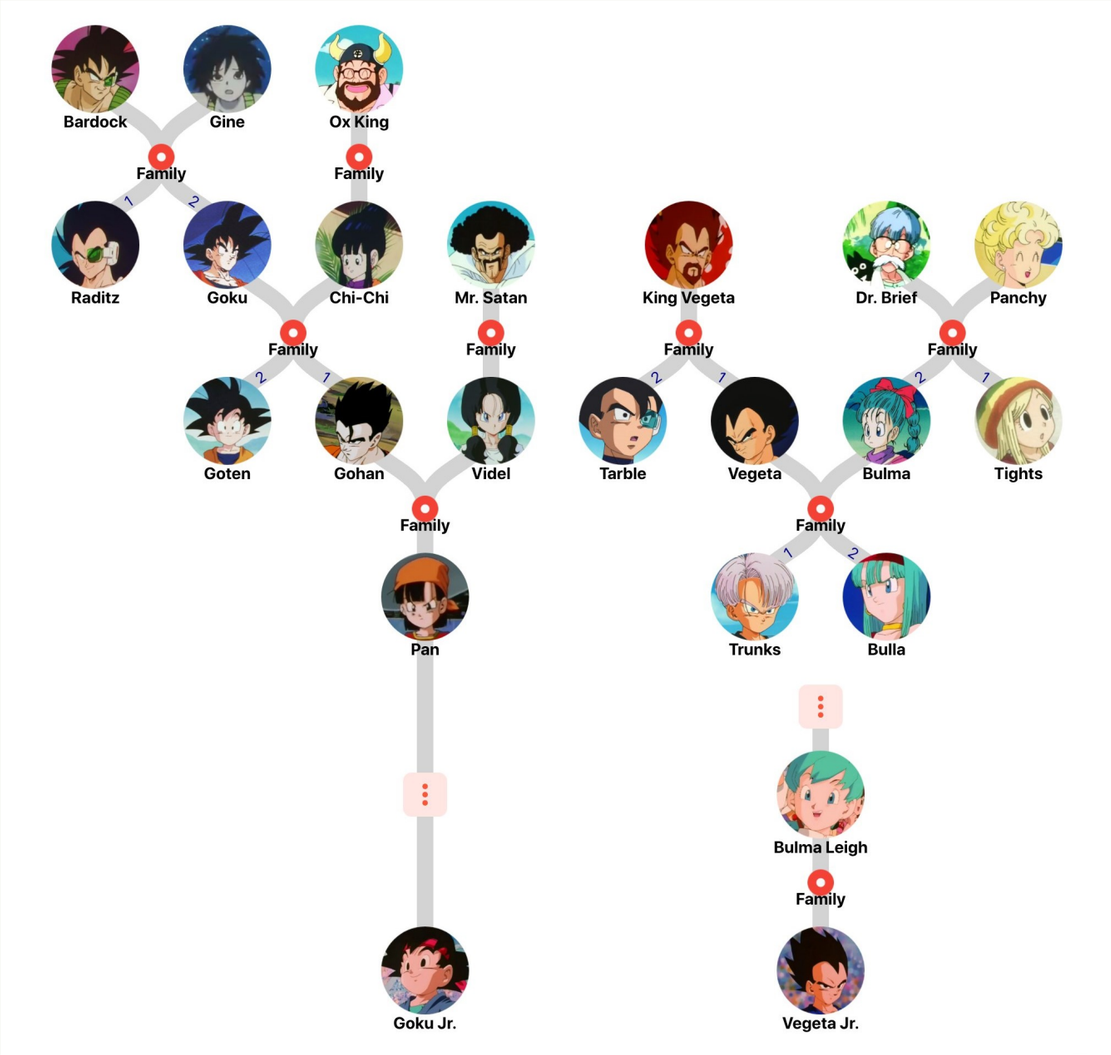
```
lista_senza_elemento([], _, []).
lista_senza_elemento(Lista, Elemento, ListaSenza):-
    Lista= [X| Xs],
    if_(X=E, ListaSenza=Ws, Ys=[X| ListaSenza]),
    lista_senza_elemento(Xs, E, Ws).
```

Agenda

- Introduzione di Prolog
- Knowledge rappresentation
- Constraint Logic Programming
- Scripting


Bardock
Gine
Ox King
Family
Family
1
2
Raditz
Goku
Chi-Chi
Mr. Satan
King Vegeta
Dr. Brief
Panchy
Family
Family
Family
Family
2
1
2
1
2
1
Goten
Gohan
Videl
Tarble
Vegeta
Bulma
Tights
Family
Family
1
2
Pan
Trunks
Bulla
Bulma Leigh
Family
Goku Jr.
Vegeta Jr.

Knowledge

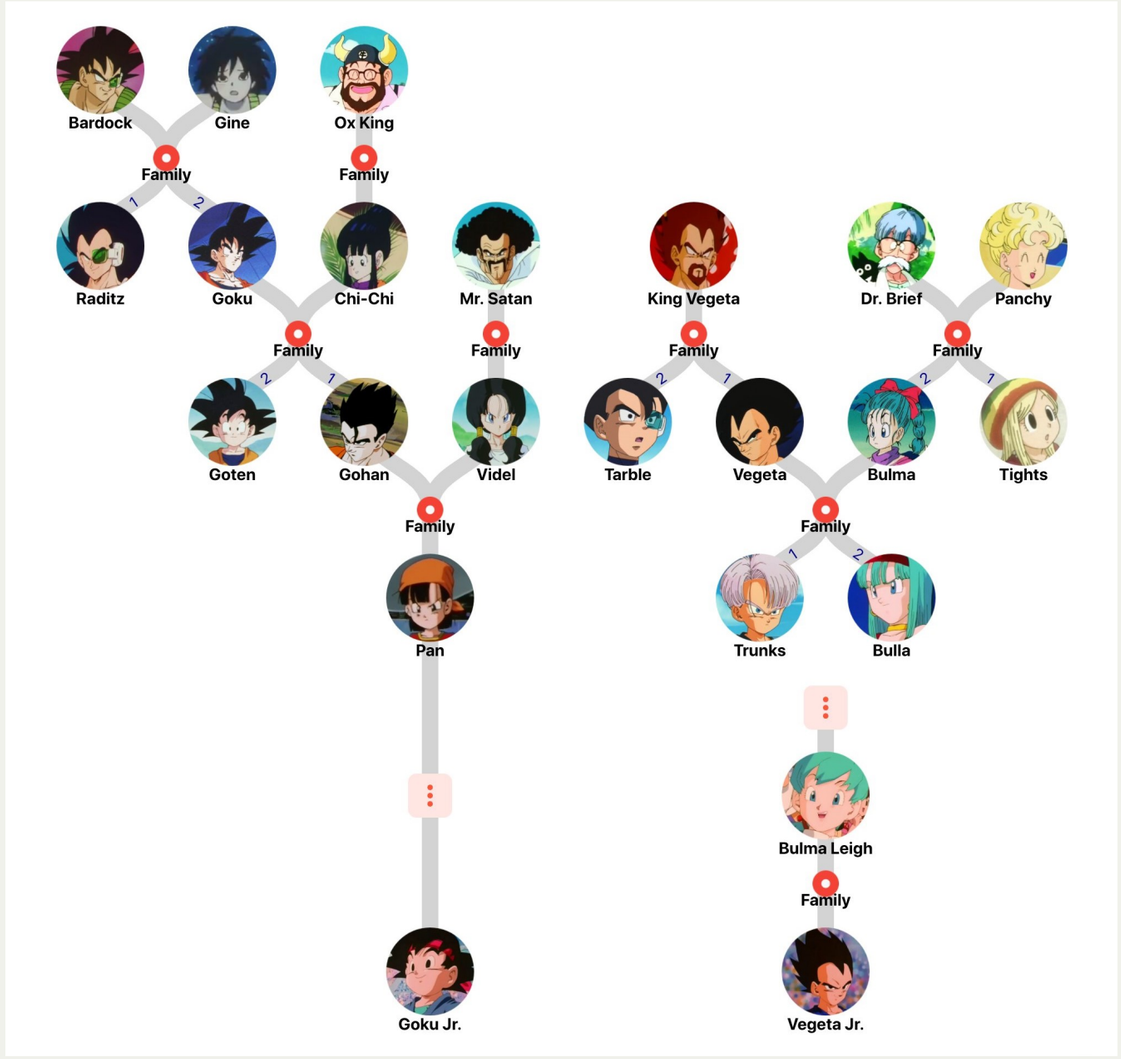



```
padre('Bardock', 'Goku').
padre('Bardock', 'Raditz').
madre('Gine', 'Goku').
madre('Gine', 'Raditz').
padre('Ox King', 'Chi-Chi').

padre('Goku', 'Gohan').
padre('Goku', 'Goten').
madre('Chi-Chi', 'Gohan').
madre('Chi-Chi', 'Goten').
padre('Mr. Satan', 'Videl').

padre('Gohan', 'Pan').
madre('Videl', 'Pan').
madre('Pan', 'Goku Jr.').
padre('King Vegeta', 'Vegeta').
...
```

Knowledge



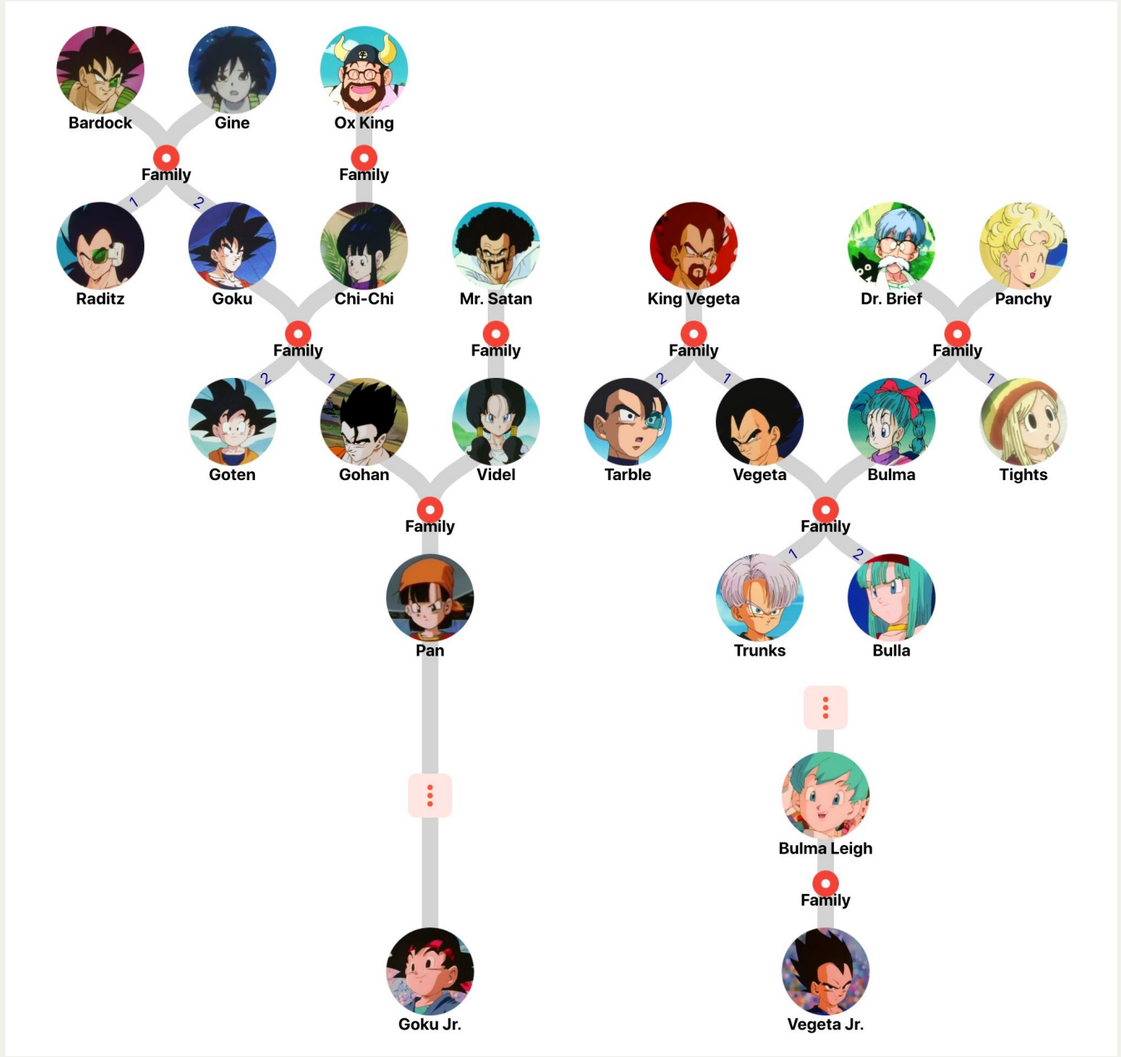

```
?- padre('Goku', 'Bulma').
false.
```

```
?- padre('Goku', Figlio).
Figlio = 'Gohan' ;
Figlio = 'Goten'.
```

```
?- madre(Mamma, 'Goten').
Mamma = 'Chi-Chi'.
```

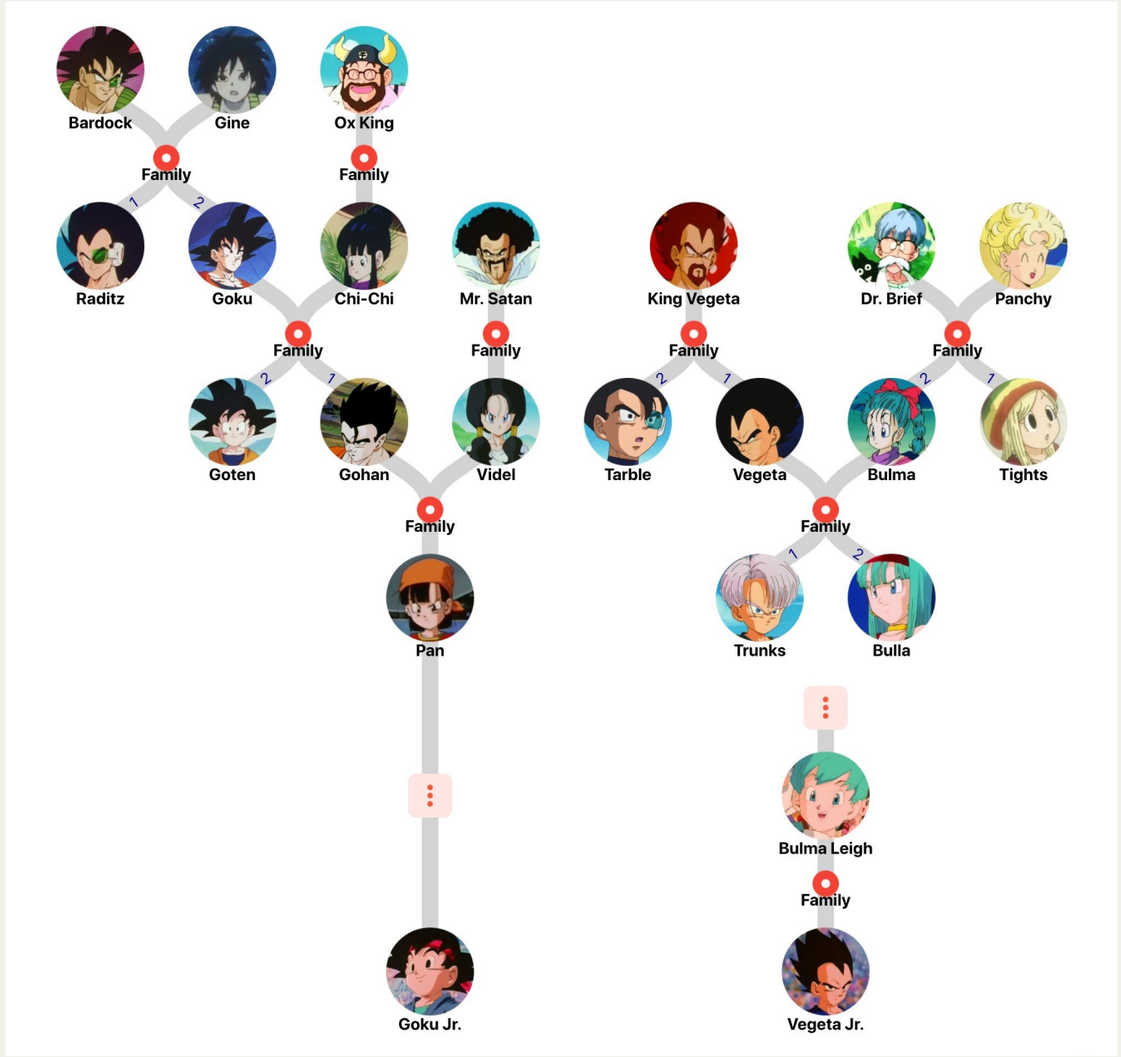
```
?- madre(X, Y).
X = 'Gine',
Y = 'Goku';
X = 'Gine',
Y = 'Raditz';
X = 'Chi-Chi',
Y = 'Gohan'
```

Knowledge




```
?- madre(X, Y).
X = 'Gine',
Y = 'Goku';
X = 'Gine',
Y = 'Raditz';
X = 'Chi-Chi',
Y = 'Gohan';
X = 'Chi-Chi',
Y = 'Goten' ;
X = 'Videl',
Y = 'Pan' ;
X = 'Pan',
Y = 'Goku Jr.' ;
...
X = 'Panchy',
Y = 'Tights'.
```

Knowledge



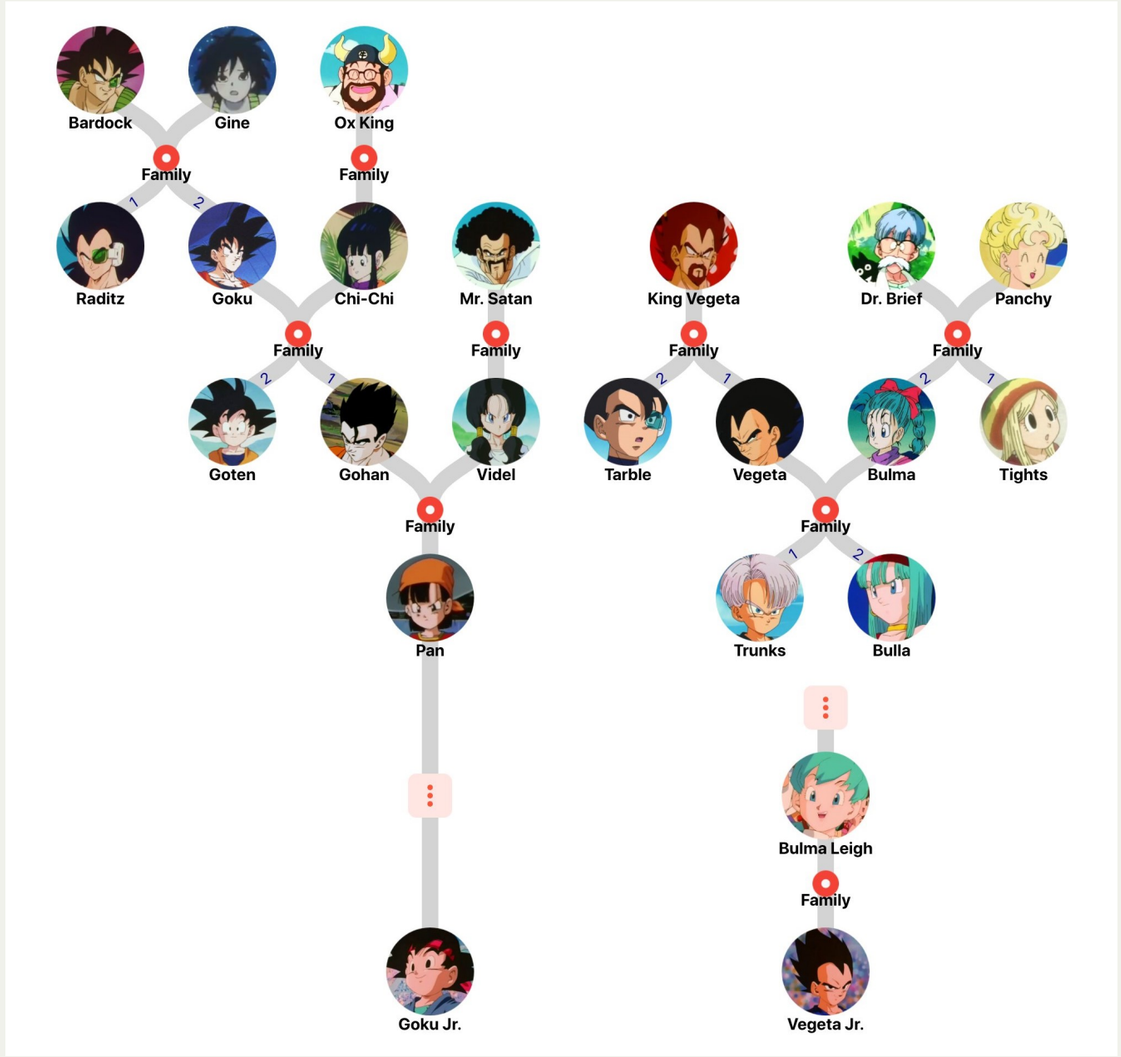

```
genitore(Genitore, Figlio):-
    madre(Genitore, Figlio)
    ;
    padre(Genitore, Figlio).
```

In maniera **equivalente**

```
genitore(Genitore, Figlio):-
    madre(Genitore, Figlio).

genitore(Genitore, Figlio):-
    padre(Genitore, Figlio).
```

Knowledge




```
fratelli(F1, F2):-
    dif(F1, F2),
    padre(Padre, F1),
    padre(Padre, F2),
    madre(M, F1),
    madre(M, F2).
```

```
zio(Zio, Nipote):-
    genitore(Genitore, Nipote),
    fratelli(Genitore, Zio).
```

```
cugino_primo_grado(Cugino0, Cugino1):-
    genitore(Gen1, Cugino1),
    zio(Gen1, Cugino0).
```

Knowledge

```
genitore(Genitore, Figlio):-
    madre(Genitore, Figlio)
    ;
    padre(Genitore, Figlio).
```

```
fratelli(F1, F2):-
    dif(F1, F2),
    padre(Padre, F1),
    padre(Padre, F2),
    madre(M, F1),
    madre(M, F2).
```

```
zio(Zio, Nipote):-
    genitore(Genitore, Nipote),
    fratelli(Genitore, Zio).
```

```
cugino_primo_grado(Cugino0, Cugino1):-
    genitore(Gen1, Cugino1),
    zio(Gen1, Cugino0).
```

```
figli(Genitore, Figli):-
    bagof(
        F,
        (
            padre(Genitore, F)
        ;
            madre(Genitore, F)
        ),
        Figli
    ).
```

Agenda

- Introduzione di Prolog
- Knowledge rappresentation#
- Constraint Logic Programming
- Scripting

Constraint Logic Programming

- Constraint satisfaction problem (CSP)
  - I problemi CSP (Constraint Satisfaction Problems) sono una classe di problemi nell'ambito dell'informatica e dell'intelligenza artificiale che coinvolge la ricerca di assegnamenti di valori a un insieme di variabili soggette a vincoli specifici in modo che tali assegnamenti soddisfino tutti i vincoli imposti.
  - Variabili con dominio booleano: **SAT**
  - Variabili con dominio intero: **ClpZ**

SAT Solving in Prolog

- Libreria `CLP(B)` di Markus Triska
- Le variabili possono avere solo valori **Booleani**
  - **0 ← falso**
  - **1 ← vero**
- Predicato `sat(Expr)`

`0`	false
`1`	true
variable	unknown truth value
atom	universally quantified variable
~ *Expr*	logical NOT
Expr + *Expr*	logical OR
Expr * *Expr*	logical AND
Expr # *Expr*	exclusive OR
Var ^ *Expr*	existential quantification
Expr =:= *Expr*	equality
Expr =\= *Expr*	disequality (same as #)

Esempio: Knights and Knaves

- Puzzle logico in cui lo scopo è individuare chi è cavaliere e chi furfante basandosi sulle affermazioni di chi *incontri*
- Ci troviamo su un'isola in cui gli abitanti possono essere o **cavalieri** (Knights) o **furfanti** (Knaves).
- I cavalieri dicono sempre la verità
- I furfanti mentono sempre

Associamo 1 ai cavalieri e 0 per i furfanti

Puzzle 1

Incontri 2 abitanti, A e B.

A dice: "*O io sono un furfante o B è un cavaliere*"

A	**B**	**L'affermazione**
0	0	1
0	1	1
1	0	0
1	1	1

Puzzle 2

Incontri 2 abitanti, A e B.

A dice: "*Io sono un furfante, B non lo è*"

A	**B**	**L'affermazione**
0	0	0
0	1	1
1	0	0
1	1	0

Puzzle 3

Incontri 3 abitanti: A, B e C.

A dice: "*Siamo tutti furfanti*"

B dice: "*Solo uno di noi è un cavaliere*"

CLPZ Solving

- Le variabili possono avere solo valori **Interi**
- Libreria `CLP(FD)` di Markus Triska
- predicato `all_distinct(Lista)`
- operatore `ins(Variabile, Dominio)`

Esempio: Sudoku

Regole

1. Ogni cella deve contenere un numero compreso tra 1 e 9
2. Ogni blocco quadrato 3x3 non può contenere due numeri diversi
3. Ogni riga non può contenere due numeri uguali
4. Ogni colonna non può contenere due numeri uguali

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Codice Sudoku

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).
sudoku(Rows):-
        length(Rows, 9), maplist(same_length(Rows), Rows),
        append(Rows, Vs), Vs ins 1..9,
        maplist(all_distinct, Rows),
        transpose(Rows, Columns), maplist(all_distinct, Columns),
```

Codice Sudoku

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).
sudoku(Rows):-
        length(Rows, 9), maplist(same_length(Rows), Rows),
        append(Rows, Vs), Vs ins 1..9,
        maplist(all_distinct, Rows),
        transpose(Rows, Columns), maplist(all_distinct, Columns),
        Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],
        blocks(As, Bs, Cs), blocks(Ds, Es, Fs), blocks(Gs, Hs, Is).

blocks([], [], []).
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-
        all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]),
        blocks(Ns1, Ns2, Ns3).
```

Agenda

- Introduzione di Prolog
- Knowledge rappresentation#
- Constraint Logic Programming#
- Scripting

Scripting example: WebScraping

- Attività il cui fine è ottenere delle informazioni da pagine web

Ringraziamenti

- MarkusTriska The Power Of Prolog
- MuHack, Ettore Fodrigo


?- the_end(X).
X= true.