

¿Te animás a cruzar el puente?: resolviendo acertijos con Prolog

¿Alguna vez te preguntaste cómo utilizar la programación lógica en tu vida cotidiana? Prolog es un lenguaje que suele usarse en la IA pero es útil para distintos tipos de problemas. En esta charla vamos a resolver un acertijo aplicando los conceptos principales del paradigma.



Jessica Saavedra

Estudiante y ayudante de cátedra en Ingeniería en Sistemas de la UTN, ferviente defensora de la educación pública que le dio la profesión. Es desarrolladora de software en 10Pines, colaboró en distintos proyectos de enseñanza de programación y es una apasionada del CSS.

¿Te animás a cruzar el puente? Resolviendo acertijos con Prolog

Jessica Saavedra

Acertijo de las cuatro mujeres y el puente

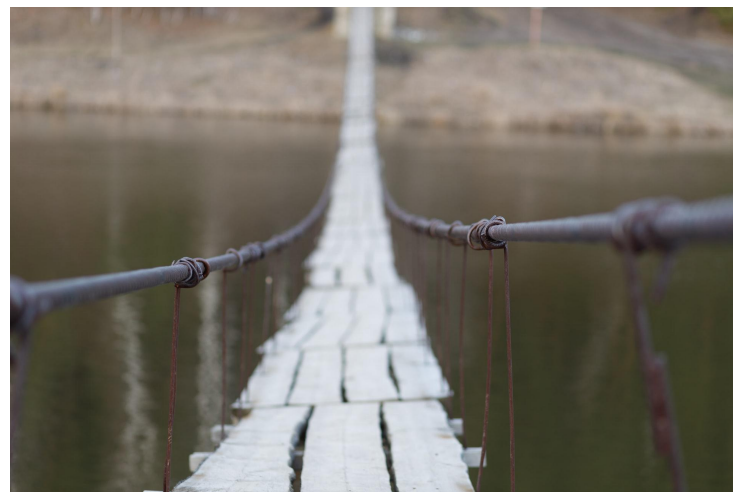
Cuatro mujeres llegan a un río por la noche. Hay un puente estrecho, el cual solo soporta a **dos personas a la vez**. Cuentan con una lámpara, imprescindible si se quiere llegar al otro lado, ya que está todo oscuro. Por lo tanto, **si cruzan dos personas, una debe volver** para que puedan cruzar los demás.



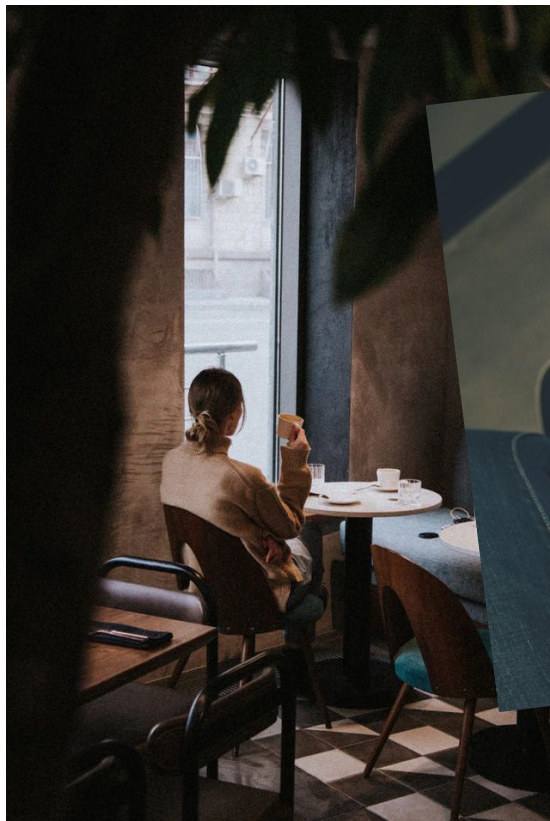
Acertijo de las cuatro mujeres y el puente

Se sabe que **Ana** puede cruzar el puente en **1 minuto**; **Belén**, en **2 minutos**. **Carla y Diana**, en **5 y 8 minutos** respectivamente.

Entonces, ¿pueden cruzar todas en **15 minutos** o menos?



¿Cómo resolverías este problema?



Paradigmas de programación



Programación
orientada a objetos



Programación lógica



Programación funcional

El asesinato de la tía Agatha



El asesinato de la tía Agatha

El asesino debe
mansion Drez



el mayordomo
más rico que

la su
ella

Tía A
son la
en la

ACOMPANAN:

intive  **10Pines**
Creative Software Development

El asesinato de la tía Agatha

Prolog posee un **motor de inferencia** que es capaz de deducir información nueva a partir de la que nosotros le especificamos



```
SWI-Prolog (AMD64, Multi-threaded, version 7.4.2)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

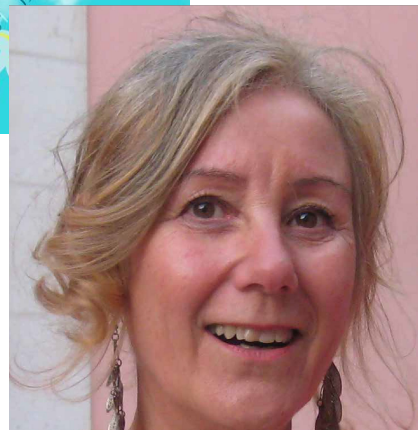
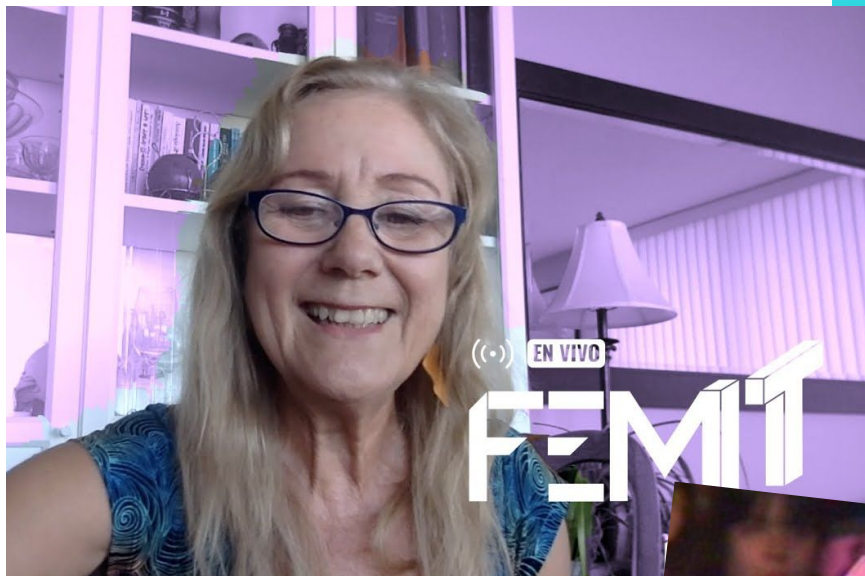
For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
```



El límite no existe



Verónica Dahl



ACOMPañAN: **intive** ■ **R/GA**

Hechos (relaciones)

```
tiempo(ana, 1).  
tiempo(belen, 2).  
tiempo(carla, 5).  
tiempo(diana, 8).
```



Al declarar estos hechos, estamos empezando a construir la **base de conocimiento** que va a utilizar Prolog para hacer deducciones

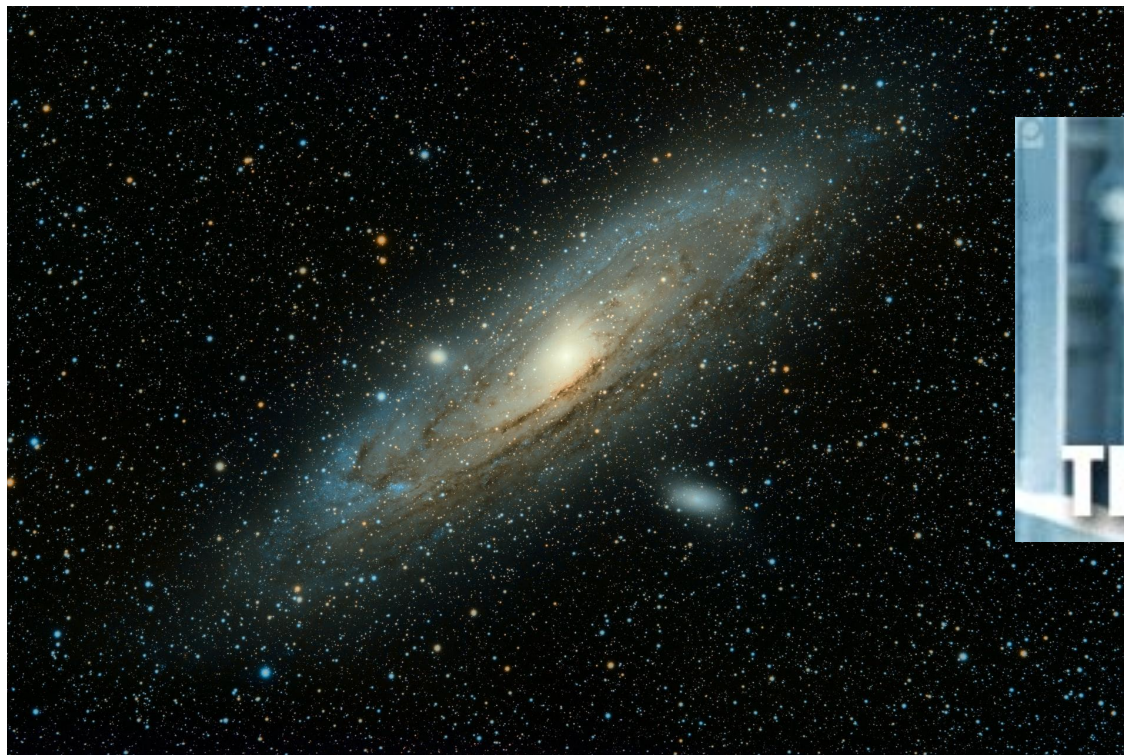
Unificación de variables

No existe la asignación como conocemos en otros lenguajes

$$\begin{cases} 2x_1 + x_2 = 7 \\ x_1 + x_2 - 3x_3 = -10 \\ 6x_2 - 2x_3 + x_4 = 7 \\ 2x_3 - 3x_4 = 13 \end{cases}$$

Principio de universo cerrado

Lo desconocido es **falso**



Lógica proposicional

Todo humano es mortal

$$\forall (x)(p(x) \Rightarrow q(x))$$

p: es humano

q: es mortal

mortal(Persona) :-
humano(Persona).

Reglas en Prolog

consecuente(Persona):-
 antecedente1(Persona),
 antecedente2(Persona),
 antecedente3(Persona).

aproboMateria(Persona):-
 aproboPrimerParcial(Persona),
 aproboSegundoParcial(Persona),
 entregoTPs(Persona).

Reglas

```
tiempo(Personas, TiempoTotal) :-  
    findall(Tiempo,  
            (member(Persona, Personas), tiempo(Persona, Tiempo)),  
            Tiempos),  
    max_list(Tiempos, TiempoTotal).
```

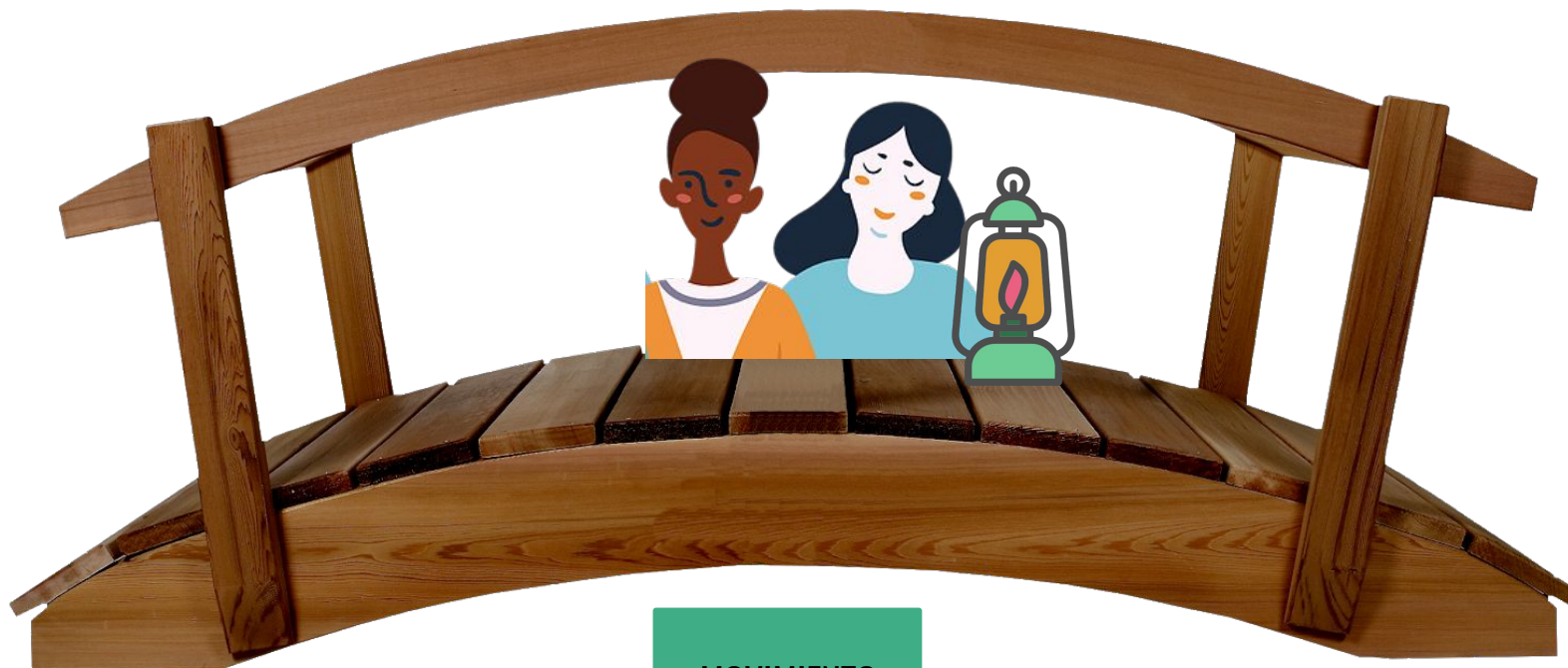
Modelando nuestro problema

ORIGEN

DESTINO



Modelando nuestro problema



MOVIMIENTO



ACOMPÑAN:

rípío ■ R/GA

Modelando nuestro problema

PERSONAS EN
ORIGEN (ANTES)

PERSONAS EN
DESTINO (ANTES)



Modelando nuestro problema

TIEMPO QUE LLEVA



DIRECCIÓN (DESTINO
A ORIGEN)



ACOMPAAÑAN: **rípío** ■ R/GA

Modelando nuestro problema

PERSONAS EN
ORIGEN (DESPUÉS)

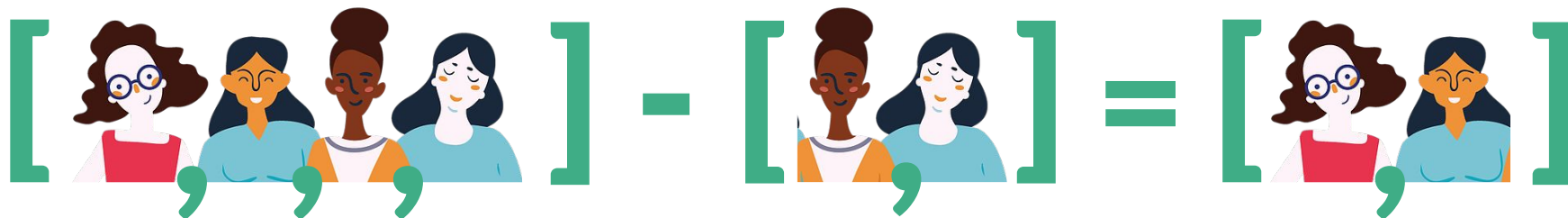
PERSONAS EN
DESTINO (DESPUÉS)



Movimientos

```
movimiento(PersonasEnOrigenAntes, PersonasEnOrigenDespues,  
            PersonasEnDestinoAntes, PersonasEnDestinoDespues,  
            TiempoQueLleva, origen_a_destino,  
            cruzaronDesde(origen_a_destino, Personas)):-  
    PersonasEnOrigenAntes \= [],  
    estadoPostMovimiento(...).
```

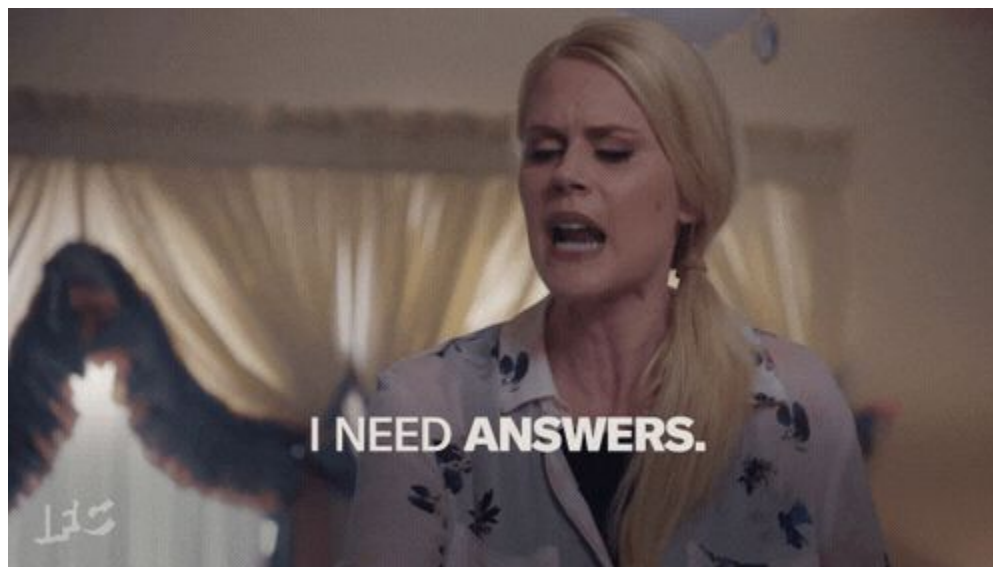
Estado post movimiento



Estado post movimiento

```
estadoPostMovimiento(PersonasEnInicioAntes, PersonasEnInicioDespues,  
    PersonasEnFinalAntes, PersonasEnFinalDespues,  
    PersonasPosibles, TiempoQueLleva) :-  
    posibilidad(PersonasEnInicioAntes, PersonasPosibles),  
    subtract(PersonasEnInicioAntes, PersonasPosibles, PersonasEnInicioDespues),  
    append(PersonasEnFinalAntes, PersonasPosibles, PersonasEnFinalDespues),  
    tiempo(PersonasPosibles, TiempoQueLleva).
```

Pero... ¿cómo obtengo la solución?



Predicado para probar el programa

?- **prueba**([ana, belen, carla, diana], ..., Movimientos).

Movimientos = [...]



Caso Base

```
prueba([], PersonasEnDestino, TiempoAcumulado, TiempoMaximo, _, []):-  
    member(ana, PersonasEnDestino),  
    member(belen, PersonasEnDestino),  
    member(carla, PersonasEnDestino),  
    member(diana, PersonasEnDestino),  
    TiempoAcumulado =< TiempoMaximo.
```

Caso Recursivo


```
prueba(PersonasEnOrigen, PersonasEnDestino,  
        TiempoAcumulado, TiempoMaximo, Direccion,  
        [MovimientoActual|ProximoMovimiento]):-  
    TiempoAcumulado =< TiempoMaximo,  
    movimiento(...),  
    NuevoTiempoAcumulado is TiempoAcumulado + TiempoQueLleva,  
    siguiente(Direccion, SiguienteDireccion),  
    prueba(PersonasEnOrigenDespues, PersonasEnDestinoDespues,  
           NuevoTiempoAcumulado, TiempoMaximo, SiguienteDireccion,  
           ProximoMovimiento).
```

¡A probarlo!






¡GRACIAS!

 @jess_saavedra

 @jess_tr

 @jessica-saavedra-valenzuela