Ayudantía 1

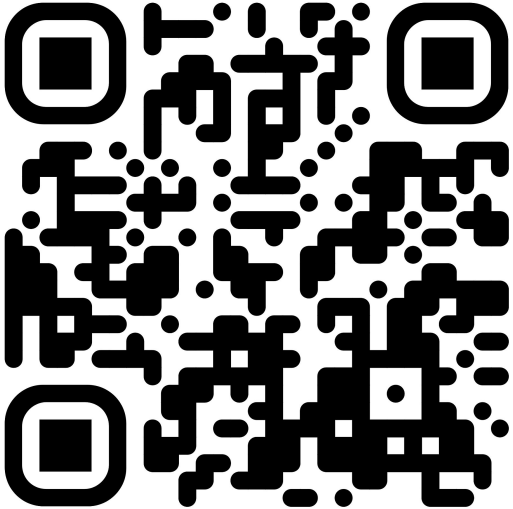
ASP y Clingo, Instalación Clingo, Presentación Tarea 2

Por Pablo González y Felipe Espinoza

Instalación de Clingo

Instalación de Clingo

- ➔ Link a la cápsula en Anuncios de Canvas
- ➔ Cualquier duda/problema por Issues de Github
- ➔ También pueden escanear el siguiente QR



ASP

¿Qué $#!% es ASP?

- ASP = *Answer Set Programming* = Programación de Conjuntos de Respuestas
- Es un paradigma de programación lógica

¿Qué $#!% es ASP?

- ASP = *Answer Set Programming* = Programación de Conjuntos de Respuestas
- Es un paradigma de programación lógica

Enfoque o estilo que define cómo se debe estructurar y escribir el código para resolver problemas en el desarrollo de software.

Paradigma de programación declarativa que utiliza la lógica formal para expresar hechos y reglas sobre un dominio de conocimiento.

¿Qué $#!% es ASP?

- ASP = *Answer Set Programming* = Programación de Conjuntos de Respuestas
- Es un paradigma de programación lógica
- Busca conjuntos de elementos que cumplan con las reglas o restricciones del programa
- Un ejemplo de lógica proposicional que podemos representar en ASP:

"Todos los hombres son mortales" y "Sócrates es hombre" □ "Sócrates es mortal"

¿Qué $#!% es ASP?

Programación declarativa

- **Describe** hechos como
  - *“El Sol existe”*
  - *“El Sol emite calor”*
- ¡Está en lenguajes como Clingo!

Programación imperativa

- Da **órdenes** como
  - *“Suma 6 + 17”*
  - *“Imprime ‘Hello world!’ en consola”*
- Está en lenguajes como Python

```
existe(sol).
```

```
print(“Hello world!”)
```

Aplicaciones de ASP

Muchas situaciones que involucran

- ➔ Problemas de búsqueda
- ➔ Puzzles lógicos
- ➔ Satisfacibilidad de restricciones
- ➔ Problemas con grafos
- ➔ Pensamiento normativo

se pueden resolver usando ASP.

Clingo

Clingo

- ➔ Lenguaje que combina **ASP** con solucionadores de satisfacibilidad **SAT.** Lo usaremos para escribir programas lógicos.
- ➔ Sus archivos tienen **extensión .lp** y, para ejecutarlos, se debe escribir en consola el comando

```
clingo archivo.lp
```

Predicados

- ➔ Constantes que representan una propiedad, relación, o características con sus términos
- ➔ **Siempre comienzan con una minúscula**

```
existe(sol). % Constante simbólica
existe(1).   % Constante numérica
existe(X).   % Variable
```

Ojo: las variables solo existen dentro de los predicados, y siempre comienzan con **mayúscula**

Predicados

Aridad

➔ Corresponde al **número de términos** que reciben.

```
p.                          % Predicado nulario (aridad 0)
ayudante(juanjo).           % Predicado unario  (aridad 1)
amigo(shrek, burro).    % Predicado binario (aridad 2)
```

Átomos/Proposiciones

- ➔ Definen propiedades o reglas que pueden ser **verdaderas** o **falsas**
- ➔ Un mismo predicado puede definir múltiples proposiciones, si se definen con la misma palabra pero distinta aridad

```
p.
p(q).

aprende(estudiante).
aprende(estudiante, profesor).
```

Modelo

- ➔ Es la **solución** del programa lógico
- ➔ Es un **conjunto minimal** de átomos que satisfacen las condiciones lógicas
- ➔ Pueden existir **varios**, así como **ninguno**

Ejemplo de output de *Clingo*

```
Solving...
Answer: 1
p r q
SATISFIABLE

Models       : 1
```

El modelo obtenido

{p, r, q}

Modelo

Minimalidad

- ➔ Solo **son modelos aquellos conjuntos con la mínima cantidad** posible de átomos
- ➔ De lo contrario, podrían existir infinitos modelos
- ➔ Para el ejemplo anterior, si **{p, r, q}** es un modelo, **{p, r, q, s}** no puede serlo

Reglas

Head ⟵ *Body*

- ➔ Si *Body* es verdadero, algo en *Head* también debe serlo
- ➔ Tanto *Head* como *Body* son conjuntos de átomos o proposiciones
- ➔ Se pueden construir hechos a partir de reglas que carezcan de *Body*

```
llueve.
mojado(niño) :- llueve.
enojado(niño) :- mojado(niño).      % lamentable :(
```

¿Cuál o cuáles son los modelos de este programa?

Reglas

Head ⟵ *Body*

- ➔ Si *Body* es verdadero, algo en *Head* también debe serlo
- ➔ Tanto *Head* como *Body* son conjuntos de átomos o proposiciones
- ➔ Se pueden construir hechos a partir de reglas que carezcan de *Body*

```
llueve.
mojado(niño) :- llueve.
enojado(niño) :- mojado(niño).      % lamentable :(
```

El modelo es **{llueve, mojado(niño), enojado(niño)}**

Reglas

Body con varios átomos

➔ Generan una **conjunción** de proposiciones; es decir, se debe cumplir todo en *Body* para que la regla se exija

```
a.                  % a se encuentra en el modelo
b.                  % b se encuentra en el modelo
c :- a, b.          % c está sólo si a y b lo están
d :- a, m.          % d está sólo si a y m lo están
```

¿Cuál o cuáles son los modelos de este programa?

Reglas

Body con varios átomos

➔ Generan una **conjunción** de proposiciones; es decir, se debe cumplir todo en *Body* para que la regla se exija

```
a.                  % a se encuentra en el modelo
b.                  % b se encuentra en el modelo
c :- a, b.          % c está sólo si a y b lo están
d :- a, m.          % d está sólo si a y m lo están
```

El modelo es **{a, b, c}**

Reglas

Head con varios átomos

- ➔ Generan una **disyunción** de proposiciones; es decir, cuando se cumple el *Body*, se cumple sólo uno de los átomos del *Head*
- ➔ A excepción de que se fuerce la presencia de más átomos

```
p.
q, r, k :- p.
```

¿Por qué sucede esto?

💡Hint: minimalidad

Reglas

Predicados con variables

➔ Permiten definir múltiples proposiciones de manera simultánea.

```
pajaro(carpintero).
pajaro(martin_pescador).
pajaro(condor).
vuela(carpintero).
vuela(martin_pescador).
vuela(condor).
```

Esto...

```
pajaro(carpintero).
pajaro(martin_pescador).
pajaro(condor).
vuela(Z) :- pajaro(Z).
```

...es equivalente a esto

Reglas

Filtros

➔ Los filtros son un tipo de restricciones, que siguen la sintaxis:

```
:- condicion.
```

➔ Se interpreta como: **“No puede existir un modelo en el que la condición se cumpla”**

Regla: Si la condición se cumple, entonces algo más debe cumplirse.
Filtro: Si la condición se cumple, entonces el modelo es inválido.

Reglas

Filtros (Ejemplos)

```
estudiante(pedro).
estudiante(maria).
estudiante(ana).

:- estudiante(X), X != pedro.
```

Este filtro elimina todos los modelos donde exista un estudiante distinto de pedro

```
llueve.
mojado(niño) :- llueve.
:- llueve, not mojado(niño).
```

Este filtro asegura que no puede existir un modelo donde llueva y el niño no esté mojado.

Restricciones de cardinalidad

- ➔ En el contexto de la Head de una regla, estas permiten elegir **distintas combinaciones** de átomos o predicados para que aparezcan en los modelos.
- ➔ Por ejemplo, para el programa:

```
p.
{q; r} :- p.     % Si p está en el modelo,
                 % alguna combinación entre q y r también lo está
```

¿Qué combinaciones de átomos pueden generarse desde la restricción?

Restricciones de cardinalidad

- En el contexto de la Head de una regla, estas permiten elegir **distintas combinaciones** de átomos o predicados para que aparezcan en los modelos.
- Por ejemplo, para el programa:

```
p.
{q; r} :- p.     % Si p está en el modelo,
                 % alguna combinación entre q y r también lo está
```

Las combinaciones pueden ser **{p}**, **{p,q}**, {**p,r}** y **{p,q,r}**.

Restricciones de cardinalidad

Limitando combinaciones

- ➔ Por defecto, Clingo prueba con todas las combinaciones posibles.
- ➔ Puede limitarse el número de elementos a incluir rodeando con números el conjunto de la restricción:

```
p.
1 {q; r; s} 2 :- p.  % Si p está en el modelo, alguna combinación
                     % de 1 a 2 elementos entre q, r y s
                     % también lo está
```

¿Cuántos modelos genera este programa?

Restricciones de cardinalidad

Limitando combinaciones

- ➔ Por defecto, Clingo prueba con todas las combinaciones posibles.
- ➔ Puede limitarse el número de elementos a incluir rodeando con números el conjunto de la restricción:

```
p.
1 {q; r; s} 2 :- p.  % Si p está en el modelo, alguna combinación
                     % de 1 a 2 elementos entre q, r y s
                     % también lo está
```

Ahora, las combinaciones pueden ser **{p;q}**, **{p;r}**, **{p;s}**, **{p,q;r}**, **{p;r;s}** y **{p;q;s}** **6 modelos**

Restricciones de cardinalidad

- ➔ Las restricciones igual pueden escribirse de forma paramétrica con variables, para evitar enumerar manualmente cada constante.
- ➔ Actualmente tenemos programas de este estilo:

```
robot(r1).
robot(r2).
robot(r3).

1 { encendido(r1); encendido(r2); encendido(r3) } 1.
```

Pero si creamos un nuevo robot, tenemos que cambiar también la restricción de cardinalidad...

Restricciones de cardinalidad

➔ Este programa es equivalente al anterior, pero mejor:

```
robot(r1).
robot(r2).
robot(r3).

1 { encendido(R) : robot(R) } 1.
```

➔ Aquí `R` recorre todos los robots definidos por el programa.

➔ Si quisiéramos agregar `robot(r4).` la restricción de cardinalidad ya lo considera.

Así nuestro código es más escalable.

Ejercicio en Menti

Para ganar un premio

Presentación Tarea 2

Ejercicio de ejemplo