Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Lenguajes de Programación (6204)
Semestre 1-2025

Proyecto #2 – Prolog
Draft Beers Reloaded

En una fiesta, se cuenta con 3 barriles de cerveza de diferentes capacidades. Los barriles están conectados entre sí mediante tubos que permiten transferir cerveza de uno a otro. Además, cada barril tiene una salida que permite servir cerveza directamente en vasos.

Su objetivo como especialista en programación lógica es diseñar un conjunto de predicados (reglas y hechos) en Prolog que permitan determinar cuántos litros de cerveza deben agregarse entre los barriles para servir exactamente **n** vasos de cerveza desde cualquiera de las salidas. El programa debe considerar las siguientes restricciones:

1. Cada vaso corresponde a un litro de cerveza.
2. Cada barril tiene una capacidad máxima y una cantidad inicial de cerveza.
3. Los barriles están conectados de la siguiente manera: A <-> B <-> C.
4. Se puede agregar cerveza desde los barriles A y C.
5. Los vasos se sirven desde cualquier barril que cumpla con la cantidad de cerveza solicitada.
6. Las transferencias entre barriles deben respetar las capacidades máximas de los mismos.
7. Se debe minimizar la cantidad de cerveza a agregar (mejor solución).

Ejemplo:

- Barril A: Capacidad 1L, contiene 1L.
- Barril B: Capacidad 7L, contiene 2L.
- Barril C: Capacidad 4L, contiene 3L.

Objetivo: Servir exactamente 6 vasos de cerveza desde un solo barril (1L por vaso).

Resultado esperado: agregar 4L de cerveza desde el Barril A para servir los 6 vasos de cerveza desde el Barril B.

Parte 1 – Inicialización de barriles

La representación de un barril se hará mediante el predicado **barrel**(ID, Capacity, Beer). Donde:

1. ID: representa el identificador del barril.
2. Capacity: capacidad total de cada barril.
3. Beer: cantidad de cerveza en litros.

Para representar los barriles de forma dinámica en Prolog, se pueden utilizar hechos que sean modificables en tiempo de ejecución. La idea es implementar el predicado `initialBarrels(Barrels, Capacity, Beer)`, el cual permitirá definir y almacenar en la base de conocimientos la información de los barriles, incluyendo su capacidad y cantidad de cerveza inicial.

Ejemplos:

```
1) ?- initialBarrels(["A", "B", "C"], [10, 7, 3], [10, 0, 0]).
```

Base de conocimientos:

```
barrel("A", 10, 10).

barrel("B", 7, 0).

barrel("C", 3, 0).
```

Parte 2 – Existe solución

Implementa un predicado que permita verificar si existe solución si se agrega cierta cantidad de cerveza desde un barril específico.

```
iSolution(Barrel, Beer, Goal).
```

Base de conocimientos:

```
barrel("A", 10, 1).

barrel("B", 7, 3).

barrel("C", 3, 0).
```

Ejemplos:

```
1) ?- iSolution("A", 1, 2).

true.

2) ?- iSolution("C", 3, 4).

false.
```

Parte 3 – Añadir cerveza

Se necesita implementar un predicado para añadir cerveza a un barril, respetando las capacidades máximas de los barriles y las cantidades actuales de cerveza. Si el barril sobrepasa su capacidad debe devolverse la cantidad de cerveza a transferir al barril vecino.

```
addBeer(Barrel, Beer, Transfer).
```

Base de conocimientos:

```
barrel("A", 20, 15).

barrel("C", 4, 3).
```

Ejemplos:

```
1) ?- addBeer("A", 5, Transfer).

Transfer = 0.

2) ?- addBeer("C", 5, Transfer).

Transfer = 4.
```

Parte 4 – Solución

Implementar un predicado que permita determinar la cantidad de cerveza que debe añadirse a los barriles para alcanzar una cantidad específica de vasos en uno de los tres barriles.

```
findSolution(Goal, SolutionType, Result).
```

Dónde:

- Goal: la cantidad de cerveza que se desea servir.
- SolutionType: indica si se busca la mejor solución (best) o todas las soluciones posibles (all).
- Result: la cantidad de cerveza que se debe agregar y el barril correspondiente.

Base de conocimientos:

```
barrel("A", 10, 3).

barrel("B", 7, 0).

barrel("C", 4, 0).
```

Ejemplos:

```
1) ?- findSolution(4, "best", Result).

Result = (1, "A").

2) ?- findSolution(4, "all", Result).

Result = (1, "A") ;

Result = (4, "C") ;

Result = (8, "C") ;

Result = (11, "A") ;

Result = (12, "C") ;

Result = (18, "A").

3) ?- findSolution(1, "best", Result).

Result = (0, "N/A").
```

Puntuación

- Parte 1, 1pto.
- Parte 2 y 3, 2 ptos c/u.
- Parte 4, 14 ptos.
- Readme, 1 pto.

Consideraciones para la entrega

- Debe utilizar SWI-Prolog.
- Realice las validaciones que considere necesarias.

- Los predicados definidos deben estar dentro de un archivo llamado **`Proyecto2.pl`**. Puede definir los predicados auxiliares con cualquier nombre que describa su propósito, pero los predicados descritos arriba deben mantener su nombre y aridad.
- Documente la solución (representación, hechos definidos, estrategia utilizada, etc.) en un archivo en formato *markdown* (**`README.md`**). No están permitidos documentos en ningún otro formato.
- No está permitido utilizar soluciones de terceros (incluyendo soluciones generadas por LLMs). Cualquier material consultado para ideas de solución o documentación debe ser referenciado en una sección del **`README.md`**, indicando para qué sirvió.
- El proyecto se puede realizar en grupos de **hasta dos estudiantes** y la fecha de entrega será el **lunes 23/06/2025**, hasta las 11:59 pm (VET).
- El proyecto se debe adjuntar por e-mail a la dirección `ldpucv@gmail.com` usando como asunto:
  `"[Proyecto Prolog]" <DatosEstudiante> [; <DatosEstudiante>]`
  donde `<DatosEstudiante> ::= <Apellidos><Nombres>,<CI>`.

José Yvimas, junio 2025