

Proyecto II: Prolog

Jaque Mate, Jack

(15 pts)

A pesar de haber sido derrotado en la ciudad de New Haskell, Jack Lambda pudo escapar de la justicia gracias a un sistema de backtracking que tenía en su guarida. Dos semanas más tarde, ustedes como tupla estelar de investigadores, reciben una pista de que Jack Lambda se ha trasladado a la ciudad de SWI Francisco, bajo el alias de Carl Cut. Unificando sus fuerzas, lo logran acorralar en su base de conocimientos, pero en ese momento descubren que quedaron atrapados en una iteración y colocó cortes en todas las salidas posibles, por lo que nadie puede escapar. Finalmente, Carl Cut les propone un desafío peculiar, tras el cual promete entregarse a las fuerzas de la ley procedural. Dado que ya demostró que su suerte no valía para vencerles, les plantea una serie de acertijos de ajedrez, donde se debe encontrar la mejor manera de terminar una partida en situaciones particulares.

Sabiendo de antemano del ingenio de su adversario, usted decidió implementar el siguiente conjunto de predicados que simulan un juego de ajedrez en el lenguaje de programación Prolog.

1. `valido\1`

Primero, se desea tener una forma de saber si el tablero observado es válido. Para ello, usted se valdrá del predicado `valido\1`, definido de la siguiente forma:

`valido(Tablero)`

Donde `Tablero` es una lista de piezas en sus posiciones, descritas de la siguiente manera:

- `peon(Jugador,Fila,Columna)`
- `torre(Jugador,Fila,Columna)`
- `caballo(Jugador,Fila,Columna)`
- `alfil(Jugador,Fila,Columna)`
- `dama(Jugador,Fila,Columna)`
- `rey(Jugador,Fila,Columna)`

Donde `Jugador` puede ser `blancas` o `negras`, y `Fila` y `Columna` son enteros tales que $1 \leq Fila, Columna \leq 8$. Estas restricciones deben verificarse para que un tablero sea válido, y además debe verificarse que:

- No haya dos piezas ocupando la misma posición.

- Cada jugador debe tener a lo sumo 16 piezas.
- Por cada jugador, la cantidad p de peones no debe ser mayor a 8, y además, para cubrir la posibilidad de que un peón sea promovido:
 - Máximo $10 - p$ torres por jugador.
 - Máximo $10 - p$ caballos por jugador.
 - Máximo $10 - p$ alfiles por jugador.
 - Máximo $9 - p$ damas por jugador.
- Cada jugador debe tener en todo momento exactamente 1 rey.

Ejemplo de un tablero válido:

```
?-valido([torre(negras,1,1),peon(negras,2,2),peon(negras,2,4),rey(negras,1,4),
caballo(negras,4,6),rey(blancas,8,5),peon(blancas,7,5),dama(blancas,6,4)]
True
```

2. mostrar\1

Por comodidad y para ver fácilmente las jugadas, debe implementar un “predicado” `mostrar\1` de manera que `mostrar(Tablero)` muestre en pantalla una representación en texto del estado del `Tablero`. Si el tablero no es válido, el predicado debe fallar.

Ejemplo válido de mostrar:

```
?-mostrar([torre(negras,1,1),peon(negras,2,2),peon(negras,2,4),rey(negras,1,4),
caballo(negras,4,6),rey(blancas,8,5),peon(blancas,7,5),dama(blancas,6,4)]

--|--|--|--|--|--|--|--
TN|  |  |RN|  |  |  |
-----+-----+-----+-----
  |PN|  |PN|  |  |  |
-----+-----+-----+-----
  |  |  |  |  |  |  |
-----+-----+-----+-----
  |  |  |  |CN|  |  |
-----+-----+-----+-----
  |  |  |  |  |  |  |
-----+-----+-----+-----
  |  |  |DB|  |  |  |
-----+-----+-----+-----
  |  |  |PB|  |  |  |
-----+-----+-----+-----
  |  |  |RB|  |  |  |
--|--|--|--|--|--|--|--

True
```

Ejemplo inválido de mostrar:

```
?-mostrar([torre(negras,1,1),peon(negras,2,2),peon(negras,2,2),rey(negras,1,4),
caballo(negras,4,6),rey(blancas,8,5),rey(blancas,6,3)]
False
```

Nota: es un ejemplo inválido ya que hay dos piezas en una misma posición, y aparte hay más de un rey blanco.

3. mover\3

Ahora que se puede saber que un tablero es válido, se debe saber si un movimiento es posible para un jugador o no. Para ello, debe crear un predicado `mover\2` de la siguiente manera:

```
mover(Jugador,Anterior,Actual)
```

Este predicado debe triunfar si, desde el tablero en `Anterior` es posible, con una pieza perteneciente al `Jugador`, realizar un movimiento que resulte en el tablero en `Actual`. El predicado debe estar escrito de tal manera que, al unificar con `Actual`, pueda generar todos los movimientos posibles por *backtracking*. A continuación, el listado de movimientos disponibles que puede hacer cada jugador.

- Una pieza de tipo `peon`, solo puede desplazarse UNA casilla de manera frontal en el tablero (no puede retroceder, solo avanzar). Esto quiere decir que los peones blancos solo pueden restar filas y los negros solo pueden sumar filas. En caso de tener una ficha enemiga en la siguiente casilla diagonal, puede atacar esa pieza, avanzando en diagonal y sacando dicha pieza del juego. En caso de estar en su posición inicial (fila 7 para peones blancos y fila 2 para peones negros), el `peon` puede avanzar hasta DOS casillas frontalmente. Adicionalmente, si un `peon` está a una casilla de recorrer todo el tablero (esto es, el un peón blanco en fila 2 o un peón negro en fila 7), entonces puede ser promovido y su movimiento válido es moverse a la fila final y ser reemplazado por cualquier otra pieza, excepto un rey u otro peón.
- Una pieza de tipo `torre`, puede desplazarse cualquier cantidad de casillas horizontal y verticalmente en el tablero, hasta encontrar otra pieza. En caso de ser enemiga, puede atacar, reemplazando dicha pieza.
- Una pieza de tipo `alfil`, puede desplazarse cualquier cantidad de casillas diagonalmente en el tablero, hasta encontrar otra pieza. En caso de ser enemiga, puede atacar, reemplazando dicha pieza.
- Una pieza de tipo `dama`, puede desplazarse cualquier cantidad de casillas horizontal, vertical y diagonalmente en el tablero, hasta encontrar otra pieza. En caso de ser enemiga, puede atacar, reemplazando dicha pieza.
- Una pieza de tipo `rey`, puede desplazarse UNA casilla horizontal, vertical o diagonalmente en el tablero. En caso de tener una pieza enemiga contigua, puede atacar, reemplazando dicha pieza.
- La pieza de tipo `caballo` tiene el movimiento más particular de todos en forma de L. Esto quiere decir que puede moverse dos casillas seguidas horizontales y una vertical, o dos casillas seguidas verticales y una horizontal. Además, puede saltar piezas sin importar el color. Si hay una pieza enemiga en la casilla final de su movimiento, puede atacarla, reemplazando dicha pieza.

Por último, se debe considerar que un movimiento es válido siempre y cuando el jugador que esté haciendo el movimiento no deje a su propio Rey en Jaque.

3.1. `jaque\2`

El predicado `jaque(Jugador,Tablero)` triunfará si el `Jugador` se encuentra en jaque para el `Tablero` proporcionado, es decir, si alguna de las piezas del oponente puede realizar un movimiento hacia la casilla donde tiene su rey.

3.2. `mate\2`

El predicado `mate(Jugador,Tablero)` triunfará si el `Jugador` no tiene ningún movimiento legal en el `Tablero` actual, es decir, está en jaque mate.

4. `puedeGanar\3`

Para saber si un jugador puede ganar en N turnos o menos, debe realizar el predicado `puedeGanar\4`, de la siguiente forma:

`puedeGanar(Jugador,Actual,Final,N)`

De manera que triunfe si el `Jugador`, partiendo del tablero `Actual`, puede llegar en N turnos o menos, con N un entero mayor o igual a 0, a una situación en la que el jugador contrario esté en jaque mate, con el tablero en `Final`.

5. `leer\1`

Para ahorrar tiempo cargando el tablero a tu programas, debe implementar un predicado `leer\1`, de tal forma que `leer(Tablero)` pida un nombre de archivo al usuario y lea, a partir del contenido de dicho archivo, la estructura de un tablero (con el mismo formato con el que se escribiría en el intérprete de *SWI-Prolog*). Finalmente, dicho tablero debe quedar unificado en `Tablero`.

Note que `leer\1` no es en realidad predicado, sino un procedimiento imperativo impuro (su intención no es la de hacer cumplir un predicado, sino la de alterar el estado del programa con información externa al mismo).

Detalles de la Entrega

La entrega del proyecto consistirá de un único archivo `p2-<carné1>&<carné2>.tar.gz`, donde `<carné1>` y `<carné2>` son los números de carné de los integrantes de su equipo. Por ejemplo, si el equipo está conformado por 00-00000 y 11-11111, entonces su entrega debe llamarse: `p2-00-00000&11-11111.tar.gz`. Este archivo debe contener únicamente:

- Un archivo `ajedrez.pl` que contenga la implementación de las funcionalidades solicitadas.
- Un archivo `Readme` con los datos de su equipo y cualquier detalle relevante a su implementación.

El proyecto deberá ser entregado a ambos profesores encargados del curso (Carlos Infante y Alexander Romero), únicamente a sus direcciones de correo electrónico institucionales: (13-10681@usb.ve y 13-11274@usb.ve) a más tardar el Domingo 29 de Marzo, a las 11:59pm. VET.

C. Infante & A. Romero / Enero – Marzo 2020