Predicado bagof/3

O predicado **bagof** em Prolog é usado para obter uma "bolsa" de alternativas que poderiam satisfazer um certo objetivo.

O **bagof** se comporta da mesma maneira que o **findall**, mas com um recurso extra.

O predicado **bagof** tem a seguinte assinatura: ***bagof*** *(+* ***Template****,: **Objective**, -**Bag**)*

- ***Template***: a variável que é a alternativa dentro do **Objective**. A variável deve estar presente dentro do objetivo.
- ***Objective***: a consulta / meta que estamos tentando satisfazer.
- ***Bag***: uma lista de alternativas com as quais o **Template** poderia se unir (fazer o match).

Considere a seguinte base de conhecimento:

```
electronics(acer, laptop).
electronics(hp, laptop).
electronics(toshiba, laptop).
electronics(compact, laptop).
electronics(dell, laptop).
electronics(toshiba, tv).
electronics(toshiba, washing_machine).
electronics(hp, desktop).
electronics(hp, mouse).
```

Exemplo1)

Digamos que gostaríamos de descobrir todas as empresas que criam laptops. Primeiro, definimos o **Objective** como:

```
electronics(Company, laptop).
```

Onde a *Company* é o **template** que estamos tentando encontrar todas as suas alternativas.

Finalmente, como estamos tentando encontrar uma lista de empresas, nomeamos a **bag** para **Companies**. Assim, a consulta ficaria do seguinte modo:

```
?- bagof(Company, electronics(Company, laptop), Companies).
Companies = [acer, hp, toshiba, compact, dell].
```

Como podemos ver, a consulta retornou todas as alternativas da empresa que podem se unificar com o predicado ***electronics(Company, laptop).***

Exemplo2)

Na próxima consulta, queremos descobrir todos os componentes eletrônicos a empresa toshiba cria.

```
?- bagof(Device, electronics(toshiba, Device), Electronics).
Electronics = [laptop, tv, washing_machine].
```

Aqui, a consulta localiza todos os valores que podem unificar com a variável **`Device`** no **Objective** e os adiciona à lista **Electronics**.

Exemplo3)

Agora, observe o exemplo.

```
?- bagof(Device, electronics(Company, Device), Electronics).
Company = acer,
Electronics = [laptop] ;
Company = compact,
Electronics = [laptop] ;
Company = dell,
Electronics = [laptop] ;
Company = hp,
Electronics = [laptop, desktop, mouse] ;
Company = toshiba,
Electronics = [laptop, tv, washing_machine].
```

Nessa consulta acima, o segundo parâmetro do predicado electronics / 2 não recebe nenhum valor e apenas a variável Company está presente, o que não é unificado com nenhum valor.

Neste caso, o Prolog primeiro unifica a variável Company, e depois tenta encontrar todas as alternativas do Device.

Como podemos ver no resultado, se pedirmos mais respostas, obtemos cada empresa juntamente com a lista de produtos eletrônicos que fabrica.

Exemplo4)

Uma característica que **bagof** tem sobre **findall** é o operador **^**.

O operador **^** pode ser adicionado no início do objetivo para indicar que não nos importamos com uma variável dentro do objetivo e, portanto, o Prolog não deve tentar unificá-lo com a base de conhecimento.

Considere a seguinte base de conhecimento:

```
city( las_vegas, nevada , usa).
city( miami, florida, usa ).
city( new_jersey, new_york, usa).
city( munich, bavaria, germany).
city( augsburg, bavaria, germany).
```

Suponha que gostaríamos de encontrar os nomes de todas as cidades nos EUA, poderíamos usar a seguinte consulta:

```
?- city( X , _ , usa).
X = las_vegas ;
X = miami ;
X = new_jersey.
```

Mas usar o operador '_' em **bagof** fará com que o predicado retorne, resultando em três listas diferentes.

```
?- bagof(X, city(X, _ , usa), Xs).
Xs = [miami] ;
Xs = [las_vegas] ;
Xs = [new_jersey].
```

Portanto, teremos que substituir **'_'** por uma variável e dizer ao predicado **bagof** que o **template** não faz diferença usando o operador ^, fazendo com que o predicado retorne todas as soluções dentro de uma lista.

```
?- bagof(X, Y^city(X, Y , usa), Xs).
Xs = [las_vegas, miami, new_jersey].
```

Como você pode ver, mesmo que a variável **'Y'** tenha valores diferentes, quando usamos **"Y ^",** informamos ao predicado **bagof** que a variável **Y** não importa, portanto, o predicado tratou a variável **Y** (mais ou menos) como tendo apenas um valor, permitindo-nos acumular as alternativas de **X** na lista **Xs**.

Predicado setof/3

O predicado **setof** no Prolog é usado para obter um "conjunto" de alternativas que poderiam satisfazer um determinado objetivo. O predicado **setoff** tem a seguinte assinatura: ***setof(+Template, :Goal, -Set)***

- ***Template***: a variável que é a alternativa dentro do **Objective**. A variável deve estar presente dentro do objetivo.
- ***Objective***: a consulta / meta que estamos tentando satisfazer.
- ***Bag***: um conjunto de alternativas com as quais o **Template** poderia se unir (fazer o match).

O predicado **setof** / 3 funciona quase exatamente como o predicado **bagof** / 3.

A única diferença é que o predicado **setof** retorna um conjunto ordenado de alternativas, livre de duplicatas.

Considere a seguinte base de conhecimento:

```
electronics(acer, laptop).
electronics(hp, laptop).
electronics(toshiba, laptop). %duplicate
electronics(toshiba, laptop). %duplicate
electronics(compact, laptop).
electronics(dell, laptop).
electronics(toshiba, tv).
electronics(toshiba, washing_machine).
electronics(hp, desktop).
electronics(hp, mouse).
```

Observe as duplicatas no código acima. Agora, vamos criar uma consulta para encontrar todas as empresas que fabricam laptops. Como fizemos no exemplo **bagof**, vamos definir o objetivo como:

```
electronics( Company, laptop).
```

Exemplo 1)

Compare as duas consultas, com bagof e setof:

```
?- bagof(Company, electronics(Company, laptop), Companies).
Companies = [acer, hp, toshiba, toshiba, compact, dell].


?- setof(Company, electronics(Company, laptop), Companies).
Companies = [acer, compact, dell, hp, toshiba].
```

Ao usar o **bagof**, o Prolog simplesmente faz backtrack (retrocede) para encontrar todas as alternativas possíveis para a Company, independentemente de sua presença já na lista.

- Note que a lista Companies retornadas por **bagof** contém o elemento "toshiba" duas vezes, e "toshiba" ocorre antes de "compact".
- Por outro lado, a lista de empresas retornadas por **setof** é ordenada e não contém duplicatas (o elemento "toshiba" ocorre apenas uma vez).

Exercícios

Considere a base de conhecimento a seguir.

```
cidade(curitiba, parana, brasil).
cidade(blumenau, santa_catarina, brasil).
cidade(joenville, santa_catarina, brasil).
cidade(foz_do_iguaçu), parana, brasil).
cidade(londrina, parana, brasil).
cidade(maringa, parana, brasil).
cidade(ushuaia, terra_do_fogo, argentina).
cidade(santa_rosa, la_pampa, argentina).
cidade(cordoba, cordoba, argentina).
```

1) Encontre uma lista ordenada de todas as cidades da Argentina.

```
Cidades = [ushuaia, santa_rosa, cordoba]
```

2) Encontre uma lista com as cidades de cada estado do Brasil.

```
Cidades = [curitiba, londrina, maringa],
Estado = parana;
Cidades = [blumenau, joenville],
Estado = santa_catarina.
```

3) Encontre uma lista ordenada das cidades de cada estado da base de conhecimento.

```
Cidades = [cordoba],
Estado = cordoba;
Cidades = [santa_rosa],
Estado = la_pampa;
Cidades = [curitiba, londrina, maringa],
Estado = parana;
Cidades = [blumenau, joenville],
Estado = santa_catarina;
Cidades = [ushuaia],
Estado = terra_do_fogo
```

4) Encontre uma lista ordenada das cidades de cada país da base de conhecimento.

```
Cidades = [cordoba, la_pampa, terra_do_fogo],
Pais = argentina;
Cidades = [parana, santa_catarina],
Pais = brasil
```

5) Encontre uma lista com repetições e não ordenada das cidades de cada país da base de conhecimento.

```
Cidades = [terra_do_fogo, la_pampa, cordoba],
Pais = argentina;
Cidades = [parana, santa_catarina, santa_catarina, parana, parana],
Pais = brasil
```