Predicados dinâmicos

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Predicados dinâmicos

Um predicado dinâmico

é um predicado cuja definição pode ser alterada em tempo de execução, por meio da inclusão ou exclusão de cláusulas (fatos ou regras).

- **`dynamic(p/n)`**: declara que a definição do predicado **`p/n`** pode mudar.
- **`listing(p/n)`**: exibe a definição do predicado **`p/n`**.
- **`asserta(C)`**: insere **`C`** como primeira cláusula da definição do predicado.
- **`assertz(C)`**: insere **`C`** como última cláusula da definição do predicado.
- **`assert(C)`**: idem a **`assertz/1`**.
- **`retract(C)`**: exclui cláusula da definição que casa com **`C`**.
- **`retractall(C)`**: exclui todas as cláusulas da definição que casam com **`C`**.
- **`abolish(p/n)`**: remove a definição do predicado **`p/n`**.

Predicados dinâmicos

Exercício 1. Inclusão de cláusulas

Faça as consultas a seguir e analise os resultados apresentados:

```
?- assertz(p(a)),
   assertz(p(b)),
   assertz(p(c)),
   listing(p/1).
```

```
?- asserta(p(1,a)),
   asserta(p(2,b)),
   asserta(p(3,c)),
   listing(p/2).
```

```
?- listing(p).
```

Predicados dinâmicos

Exercício 2. Exclusão de cláusulas

Faça as consultas a seguir e analise os resultados apresentados:

- `?- retract(p(b)).`
- `?- listing(p/1).`
- `?- retract(p(2,b)).`
- `?- listing(p/2).`
- `?- retractall(p(X)).`
- `?- listing(p).`
- `?- abolish(p/2).`
- `?- listing(p).`

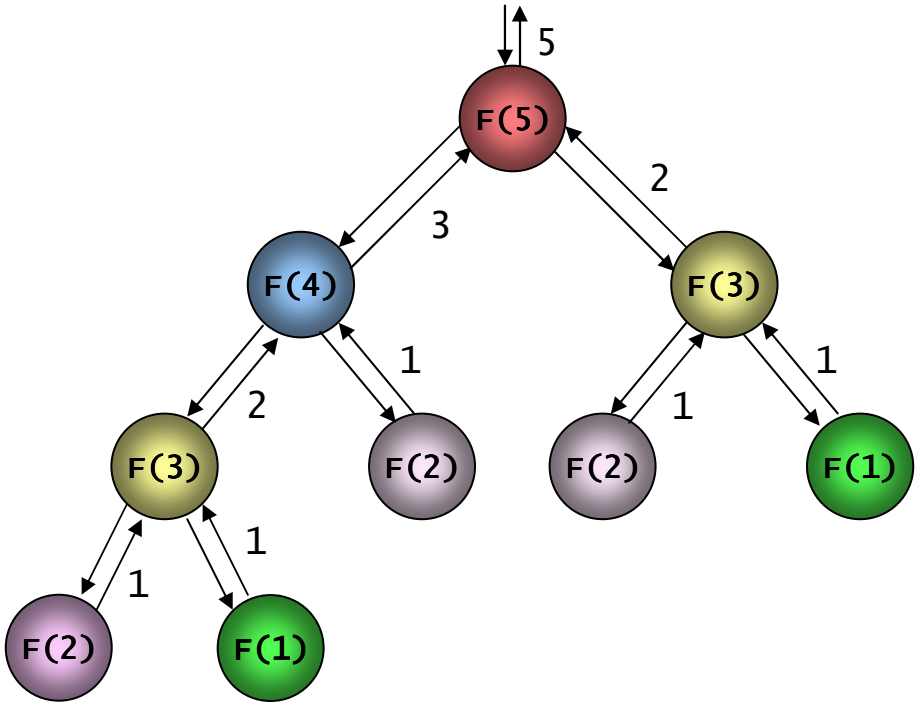
Memoização

Memoização

É uma técnica bastante eficiente para resolver problemas cuja decomposição em subproblemas mais simples apresenta muita redundância. Esta técnica consiste em manter uma tabela com as soluções de subproblemas previamente resolvidos.

Exemplo: Na sequência de Fibonacci, os dois primeiros termos são iguais a 1 e, a partir do terceiro, todo termo é igual à soma dos dois anteriores (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...).

```
F(1) = 1
F(2) = 1
F(3) = F(2) + F(1) = 2
F(4) = F(3) + F(2) = 3
F(5) = F(4) + F(3) = 5
...
F(n) = F(n-2) + F(n-1)
```



Memoização

Exemplo 1. Sequência de Fibonacci sem memoização

```
fib(N,1) :-
   N<3, !.
fib(N,F) :-
   A is N-2, fib(A,X),
   B is N-1, fib(B,Y),
   F is X + Y.
```

Exercício 3. Sequência de Fibonacci sem memoização

Digite o predicado definido no Exemplo 1 e faça a consulta a seguir:

```
?- time(forall(between(1,30,N),(fib(N,F),writeln(N -> F)))).
```

Memoização

Exemplo 2. Sequência de Fibonacci com memoização

```
:- dynamic fibmemo/2.

fibmemo(1,1).
fibmemo(2,1).

fibm(N,F) :-
    fibmemo(N,F), !.
fibm(N,F) :-
    A is N-2, fibm(A,X),
    B is N-1, fibm(B,Y),
    F is X + Y,
    assert(fibmemo(N,F)).
```

Exercício 4. Sequência de Fibonacci com memoização

Digite o predicado definido no Exemplo 2 e faça a consulta a seguir:

```
?- time(forall(between(1,30,N),(fibm(N,F),writeln(N -> F)))).
```

Coleta de soluções

Exemplo 3. Coleta de soluções (idem a `findall/3`)

```
:- dynamic solução/1.
coleta(X,C,L) :- forall(C,assertz(solução(X))), coleta(L).
coleta([X|R]) :- retract(solução(X)), !, coleta(R).
coleta([]).
% dados para teste
p(a).
p(b).
p(c).
P(b).
```

Exercício 5. Coleta de todas as soluções de uma consulta

- Digite os predicados definidos no Exemplo 3.
- Faça a consulta: `?- coleta(X,p(X),L).`

Coleta de soluções

Exercício 6. Coleta de todas as soluções de uma consulta, sem repetição

Uma característica do predicado **coleta/3** que pode ser indesejável em algumas aplicações é que ele devolve uma lista de soluções contendo soluções repetidas.

Com base na definição deste predicado, crie um novo predicado **coleta_sr/3**, que devolve uma lista com todas as soluções de uma consulta, sem repetições.

Dica: em vez de usar **assertz/1** diretamente, crie um predicado **anota/1** que inclui uma solução na base de dados dinâmica só se ela não foi incluída anteriormente.

Exercício 7. Coleta de todas as soluções de uma consulta, sem repetição

Faça as consultas e analise os resultados apresentados:

```
?- assert(f(1,a)), assert(f(2,a)), assert(f(3,b)),
   assert(f(3,c)), listing(f).

?- coleta_sr(X,f(X,Y),L).

?- coleta_sr(Y,f(X,Y),L).
```

Simulação de ambiente dinâmico

Um ambiente dinâmico

É um ambiente cujas propriedades se modificam ao longo do tempo.

Exemplo 4. Estado de uma lâmpada

```
:- dynamic lâmpada/1.
lâmpada(acesa).
apaga :- retract(lâmpada(acesa)), assert(lâmpada(apagada)).
acende :- retract(lâmpada(apagada)), assert(lâmpada(acesa)).
```

Exercício 8. Estado de uma lâmpada

- Digite os predicados definidos no Exemplo 4 e faça as consultas a seguir:
- `?- lâmpada(E).`
- `?- apaga, lâmpada(E).`
- `?- acende, lâmpada(E).`

Simulação de ambiente dinâmico

Exemplo 5. Um agente que anda, pega e solta objetos

```
:- dynamic pos/2, seg/1.
pos(agente,garagem).
pos(micro,cozinha).
pos(rádio,quarto).

ande(L) :- pos(agente,L), !.
ande(L) :- retract(pos(agente,P)), asserta(pos(agente,L)),
           format('~nAnda de ~w até ~w',[P,L]), !.

pegue(O) :- seg(O), !.
pegue(O) :- pos(O,P), ande(P),
            retract(pos(O,_)), assert(seg(O)),
            format('~nPega ~w',[O]), !.

solte(O) :- not(seg(O)), !.
solte(O) :- pos(agente,P),
            retract(seg(O)), assert(pos(O,P)),
            format('~nSolta ~w',[O]), !.
```

Simulação de ambiente dinâmico

Exercício 9. Um agente que anda, pega e solta objetos

- Digite o programa do Exemplo 5 e faça as consultas a seguir:
- `?- pegue(rádio), ande(sala), solte(rádio), ande(rua).`
- `?- pegue(rádio), ande(cozinha), solte(rádio).`

Exercício 10. Levar um objeto de um local para outro

Defina o predicado `leve(O,L)`, que faz o agente levar o objeto `O` para o local `L`; em seguida, faça as consultas a seguir:

- `?- leve(micro,quintal).`
- `?- leve(rádio,banheiro).`

Exercício 11. Descobrindo as posições de novos objetos

Modifique o predicado `pegue(O)` de modo que, quando a posição do objeto for desconhecida, o agente pergunte ao usuário a sua localização; depois, faça a consulta:

- `?- leve(notebook,sala), ande(rua), leve(notebook,escritório).`

Memória persistente

- As informações armazenadas na base de dados do Prolog são **voláteis**, ou seja, são descartadas assim que o sistema finaliza a sua execução.
- Assim, para que estas informações sejam persistam de uma execução para outra, é necessário que elas sejam preservadas em um arquivo em disco.

- `tell(A)`: redireciona a saída padrão para o arquivo de nome `A`.
- `told`: volta a enviar informações para a saída padrão (vídeo).
- `see(A)`: redireciona a entrada padrão para o arquivo de nome `A`.
- `seen`: volta a ler informações da entrada padrão (teclado).

Exercício 12. Redirecionamento de entrada/saída padrão

Faça as consultas a seguir e analise os resultados obtidos:

- `?- tell('arq.pl'), writeln('um.'), writeln('dois.'), told.`
- `?- see('arq.pl'), read(X), read(Y), read(Z), seen.`

Memória persistente

Exemplo 6. Agente com memória persistente

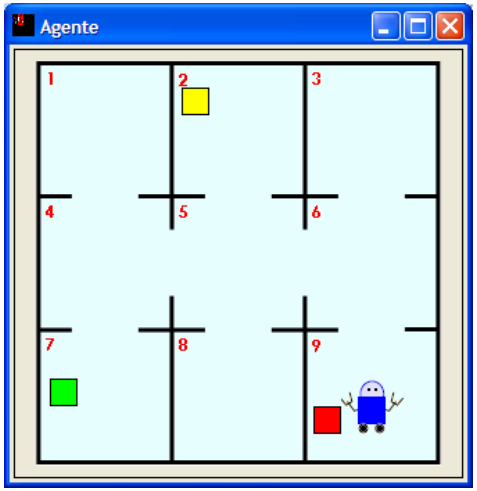
```
memorize :-
    tell('dados.pl'),
    forall(pos(X,Y), format('~w.~n',[pos(X,Y)])),
    forall(seg(X), format('~w.~n',[seg(X)])),
    told.
relembre :-
    retractall(pos(_,_)),
    retractall(seg(_)),
    see('dados.pl'),
    repeat,
      read(C),
      assert(C),
      C=end_of_file,
    retract(end_of_file), !,
    seen.
```

Exercício 13. Agente com memória persistente

Acrescente as definições acima no programa do robô e teste seu funcionamento.

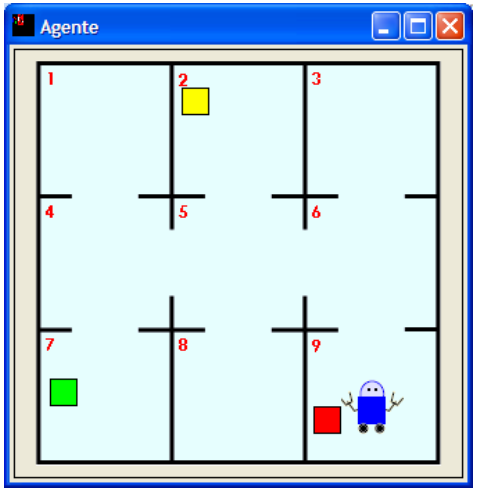
Bônus: interface gráfica

```
agente :-
    inicia,
    new(D,dialog('Agente')),
    send(D,append,bitmap(image('ambiente.bmp'))),
    send(D,display,new(@o1,box(20,20))),
    send(@o1,fill_pattern,colour(yellow)),
    send(@o1,move,point(20,30)),
    send(D,display,new(@o2,box(20,20))),
    send(@o2,fill_pattern,colour(green)),
    send(@o2,move,point(20,50)),
    send(D,display,new(@o3,box(20,20))),
    send(@o3,fill_pattern,colour(red)),
    send(@o3,move,point(20,70)),
    send(D,append,new(@a,bitmap(image('agente.bmp')))),
    send(@a,move,point(235,237)),
    new(@t,timer(1)),
    send(@t,start),
    send(D,open).
```



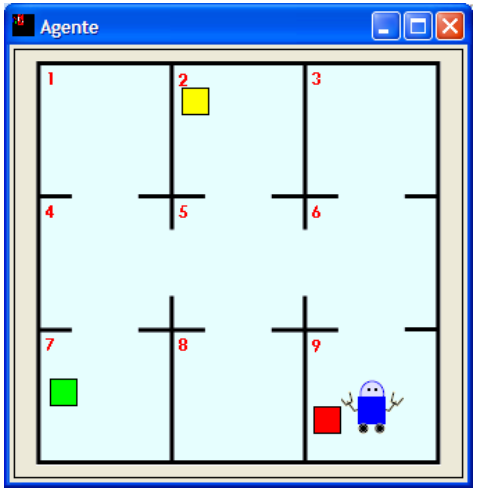
Bônus: interface gráfica

```
:- dynamic pos/2, seg/1.

inicia :-
   forall(member(X,[@t,@a,@o1,@o2,@o3]),free(X)),
   retractall(pos(_,_)),
   retractall(seg(_)),
   assert(pos(agente,9)),
   assert(pos(1,1)),
   assert(pos(2,1)),
   assert(pos(3,1)).

porta(1,4).
porta(2,5).
porta(3,6).
porta(4,5).
porta(4,7).
porta(5,6).
porta(5,8).
porta(6,9).
```



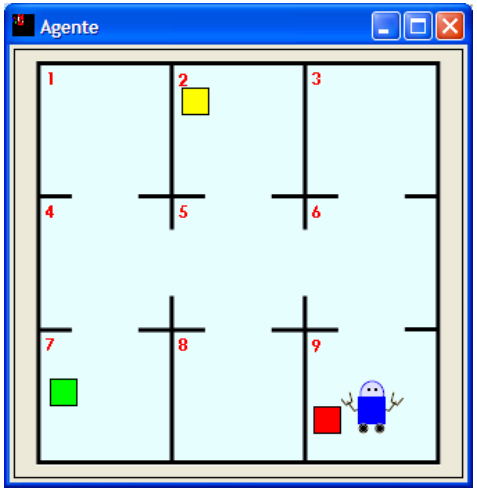
Bônus: interface gráfica

```
passagem(X,Y) :- porta(X,Y).
passagem(X,Y) :- porta(Y,X).

rota(X,X,[X]) :- !.
rota(X,Y,[X|R]) :- passagem(X,Z), rota(Z,Y,R).

% comandos para o agente

ande(L) :-
    pos(agente,L), !.
ande(L) :-
    retract(pos(agente,P)),
    assert(pos(agente,L)),
    length(R,_),
    rota(P,L,R),
    siga(R), !.
```



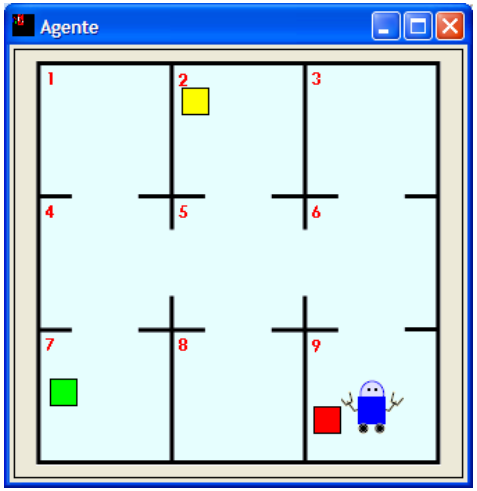
Bônus: interface gráfica

```
pegue(O) :-
    seg(O), !.
pegue(O) :-
    pos(O,P),
    ande(P),
    retract(pos(O,_)),
    assert(seg(O)),
    get(@a,position,X),
    obj(O,No,_),
    send(No,move,X), !.
solte(O) :-
    not(seg(O)), !.
solte(O) :-
    pos(agente,P),
    not(member(P,[5,6])),
    retract(seg(O)),
    assert(pos(O,P)),
    get(@a,position,point(X,Y)),
    obj(O,No,Yo), X1 is X-20, Y1 is Y+Yo,
    send(No,move,point(X1,Y1)), !.
```



Bônus: interface gráfica

```
siga([]).
siga([S|R]) :- mova(S), send(@t,delay), siga(R).

mova(S) :-
   sala(S,X,Y),
   forall(seg(O),(obj(O,No,_),
          send(No,move,point(X,Y)))),
   send(@a,move,point(X,Y)).

sala(1,  45,  47).
sala(2, 140,  47).
sala(3, 235,  47).
sala(4,  45, 142).
sala(5, 140, 142).
sala(6, 235, 142).
sala(7,  45, 237).
sala(8, 140, 237).
sala(9, 235, 237).

obj(1, @o1, -20).
obj(2, @o2,   0).
obj(3, @o3, +20).
```



Fim