Programação em Lógica

Profa. Juliana Vizzotto

juvizzotto@inf.ufsm.br

Disciplina de Paradigmas de Programação

Roteiro

- Fatos
- Regras
- Computação

Prolog

- Fato: especifica que uma tupla satisfaz um predicado
- Regra: especifica sob quais condições uma tupla de valores satisfaz um predicado

Prolog

Recursão

- Exemplo grafos: gostaríamos de definir um predicado geral o qual é satisfeito por um par de nodos, caso exista um caminho entre eles no grafo (de qualquer tamanho).
- Recursão:
  1. Caso básico: existe um caminho de um nodo para outro se existe um arco entre eles;
  2. Recursão: existe um arco do nodo até um nodo intermediário e existe um *caminho* do nodo intermediário até o nodo destino.

Prolog

Recursão

```
path(Node1,Node2) :-
   edge(Node1,Node2).

path(Node1,Node2) :-
   edge(Node1,SomeNode),
   path(SomeNode,Node2).
```

Prolog

Ground Queries

- Utilização de indentificadores de valores como argumentos dos predicados, e.g, `edge(a,b).`
- Quando uma *ground query* é executada espera-se observar *yes* ou *no* como resposta.
- Como a resposta é determinada:
  - O sistema prolog procura no programa por um fato ou regra que faça um *match* com a query.
  - Como a query `path(a,b)` é processada?

Prolog

Ground Queries

- Ainda temos uma *ground query*, entretanto não encontramos uma regra que faça um *match* exato.
- O *match* para regras trabalha com identificadores fazendo *match* em variáveis.

Prolog

Non Ground Queries

- São *queries* as quais têm variáveis como parâmetros.
- Por exemplo: `tedge(a,X)`
- O átomo nesta *query* contém uma variável e uma constante como argumentos.
- Esta *query* é verdadeira?
- Depende! Se substituimos `X` por `d`, então é verdadeira.

Prolog

Satisfiability

- Dizemos que a *query* é *satisfiable* com relação ao programa, pois existe uma substituição para a sua variável, a qual faz o átomo ser verdadeiro.
- O Processo:
  1. *match* na regra `tedge`
  2. `Node1=a` e `X=Node2`
  3. então temos uma nova *query*: `edge(a,Somenode)`, `edge(Somenode, Node2)`

Prolog

Satisfiability

- O Processo (cont):
  1. Precisamos provar duas *queries*
  2. Consideramos primeiro `edge(a,SomeNode)`
  3. Primeiro fato que *satisfaz* é `edge(a,b)`, com *substituição* `SomeNode =b`
  4. Assim, a primeira *query* é satisfeita!
  5. Agora temos que aplicar o mesmo processo para `edge(b,Node2)`
  6. Da mesma maneira, encontramos `Node2=d`, o qual satisfaz a *query*.
  7. Resumindo: seguindo as substituições `X=Node2`, `Node2=d` descobrimos que `X=d` satisfaz a *query* original.

Prolog

Unificação/Resolução

- *Unificação* é o processo de pegar dois átomos (uma *query* e outro sendo um fato ou a *cabeça* de uma regra) e verificar se existe uma substituição a qual torna a *query* **verdadeira**.
- A segunda atividade é a *Resolução*: quando um átomo a partir da *query* foi unificado com um fato ou cabeça de uma regra, o processo de *resolução* substitui o átomo com o corpo da regra (ou nada, no caso do fato) e então aplica substituição a nova *query*.

Prolog

Unificação/Resolução

Por exemplo: `tedge(a,X)`

1. Faz a *unificaçã* de `tedge(a,X)` com `tedeg(Node1,Node2)`.
   1. Então aplica *substituição* `Node1=a` e `X=Node2`
   2. *Resolução* substitui `tedge(a,X)` por `edge(Node1,someNode)` e `edge(someNode,Node2)` e aplica a substituição nas novas *queries*: `edge(a,someNode)` e `edge(someNode,Node2)`

Prolog

Unificação/Resolução

Por exemplo: `tedge(a,X)`

1. Selecionamos o átomo `edge(a,someNode)`
2. Aplicamos *unificação* de `tedge(a,X)` com `tedge(a,b)` com a substituição `X=b`
3. *Resolução* substitui `tedge(a,b)` por nada já que unificamos com um fato. Agora, pela substituição temos `edge(b,Node2)`.

Prolog

Unificação/Resolução

Por exemplo: `tedge(a,X)`

1. O último átomo `edge(b,Node2)`
2. Aplicamos *unificação* de `tedge(b,Node2)` com `tedge(b,d)` com a substituição `Node2=d`
3. *Resolução* substitui `tedge(b,d)` por nada já que unificamos com um fato.
4. Final!

Prolog

Backtracking

- Existem outras soluções para o átomo `tedge(a,X)`.
- Poderíamos refazer o processo computacional anterior considerando `X=b` ou `X=c` ou ainda `X=d`
- Prolog faz um *backtracking* na unificação mais recente para determinar se existe *outro* fato ou regra o qual também sucede na unificação.
- Caso exista, uma solução adicional pode ser encontrada.
- Prolog continua com o *backtracking* até que todas as soluções sejam encontradas.

Prolog

Exercício

- Considere a *query* `tedge(g,X)`, assumindo o grafo do exercício da aula anterior. Escreva os passos do processo de unificação/resolução claramente marcando os lugares onde mais de uma regra ou fato podem ser verificados através de unificação (lembra que cada um pode levar a uma nova solução). Então, continue o processo de unificação/resolução com *backtracking* para determinar todas as soluções possíveis.