Resumos PLOG

MIEIC

16 de fevereiro de 2021

Conteúdo

1 Introdução

Este documento contém resumos dos conteúdos lecionados em PLOG no ano letivo 2019/2020. Este documento contém apenas resumos dos conteúdos abordados na unidade curricular e não substitui o estudo mais aprofundado destes conteúdos.

2 Fundações

Programa: conjunto de axiomas.

Computação: prova construtiva de um objetivo a partir do programa.

Interpretação procedimental de uma cláusula de Horn:
A se B_1 e B_2 e ... e B_n

- Leitura declarativa: A é verdade se os B_i são verdade.
- Leitura procedimental: para resolver A, resolver B_1, B_2 e ... e B_n.

- Interpretador da linguagem: prova usando o algoritmo da unificação.

3 Conceitos

Factos: afirmar que uma relação entre objetos é verdadeira.

- Exemplo: father(abraham, isaac)
  - Relação ou predicado: father.
  - Objetos ou indivíduos: abraham, isaac.

Convenções sintáticas:

- Nomes de predicados e objetos começam com minúscula (átomos).
- Frases terminam com ponto final.

Um conjunto finito de factos é um programa em lógica.

Variável Lógica: representa um indivíduo não especificado.

- Não é uma posição de memória onde se coloca um valor!
- Convenção: começa por maiúscula.

Termos: constantes, variáveis, estruturas (termos complexos).

Estruturas: functor. Exemplos: s(0), hot(milk), name(john, doe)...

- nome(um átomo)
- aridade(número de argumentos): s/1, hot/1, name/2

Termos:

- sem variáveis: ground (totalmente instanciado)
- com variáveis: nonground

Conjunção de objetivos: a vírgula corresponde ao "e"lógico ($\wedge$).

Procedimento: coleção de regras com o mesmo predicado na cabeça.

Um programa em lógica é um conjunto finito de regras. O significado de um programa em lógica P, M(P), é o conjunto de objetivos totalmente instanciados dedutíveis de P (se o programa apenas tem factos, o seu significado é o próprio programa).

Uma base de dados em lógica contém:

- Factos: permitem definir relações, como em bases de dados relacionais.
- Regras: permitem definir perguntas relacionais complexas (vistas).

Programa recursivo linear: o corpo da regra recursiva tem apenas um objetivo recursivo.

```
ancestor(Ancestor, Descendant) :-
    parent(Ancestor, Descendant).
ancestor(Ancestor, Descendant) :-
    parent(Ancestor, Person), ancestor(Person, Descendant).
```

Uma lista é uma estrutura de dados binária: .(X, Y)

- 1^{o} argumento (cabeça): elemento; 2^{o} argumento (cauda): resto da lista.
- Símbolo constante para fim da recursão: lista vazia (nil ou []).
- Sintaxe alternativa: $.(X, Y) \equiv [X|Y]$
- Os seus elementos podem ser quaisquer termos.
- Predicados importantes:
  - Membro de uma lista: member(X, L)
    - Verifica se um elemento X é membro da lista L.
    - Obtém um elemento da lista L.
    - Obtém uma lista que contém um elemento X.
  - Prefixo: prefix(L1, L)
  - Sufixo: suffix(L1, L)
  - Sublista: sublist(L1, L)
  - Concatenação de listas: append(L1, L2, L)
  - Inversão de uma lista: reverse(L1, L)
  - Comprimento de uma lista: length(L, Length)
  - Eliminar elementos: delete(L, X, NL)
  - Selecionar elemento: select(X, L1, L)
  - Permutações: permutation(L1, L)

Frase consistente: tem uma instância verdadeira.

Frase inválida: tem uma instância falsa.

Um unificador de dois termos é uma substituição que os torna idênticos, portanto, um unificador encontra uma instância comum.

Condições de unificação:

- Variável com Variável: unificam sempre.
- Atómico com Atómico: unificam se os valores forem iguais.
- Atómico ou Estrutura com Variável: unificam sempre.
- Estrutura com Estrutura: unificam se os functores são iguais, o número de argumentos é igual e os argumentos unificam dois a dois.

Regras de Dedução:

- 1ª - Identidade: de P deduzir P?
  - Uma pergunta é uma consequência lógica de um facto idêntico.
- 2ª - Generalização: de Pθ deduzir P?
  - Uma pergunta existencial é uma consequência lógica de uma instância sua.
- 3ª - Instanciação: de P deduzir Pθ
  - De um facto quantificado universalmente, deduz-se uma instância sua.
- 4ª - Modus Ponens Universal: de $A \leftarrow B_1, ..., B_n$ e de $B_1\theta, ..., B_n\theta$ deduzir $A\theta$
  - Da instanciação do corpo de uma regra podemos deduzir a instanciação da cabeça.

4 Prolog

Execução sequencial, da esquerda para a direita, dos objetivos da resolvente.

Pesquisa sequencial de uma cláusula unificável e retrocesso (backtracking)

- Escolhe a primeira cláusula cuja cabeça unifica com o objetivo.
- Se não houver, a computação é desfeita até à última escolha (ponto de escolha), e é escolhida a cláusula unificável seguinte.

A maior parte das implementações de Prolog:

- Pesquisa a árvore até encontrar a primeira solução
- Permite ao utilizador indicar que quer mais soluções através do símbolo ;

Heurísticas de Ordenação

- Colocar testes primeiro
- Colocar primeiro os objetivos com menos soluções (depende da base de dados)
- Colocar primeiro os objetivos mais instanciados (depende do uso)
- Objetivo: falhar o mais rápido possível!
  - falhar significa podar a árvore de pesquisa, levando mais depressa à solução

Avaliador aritmético: is(Value, Expression)

- Value is Expression
- A expressão Expression é avaliada e o resultado é unificado com Value
  - a expressão não pode conter variáveis não instanciadas.
- Tem sucesso se a unificação tiver sucesso. Exemplos:
  - X is 3 + 5? X = 8
  - 8 is 3 + 5? yes
  - 3 + 5 is 3 + 5? false

No Prolog, a recursividade é usada para especificar algoritmos recursivos e iterativos

- Cláusula iterativa: chamada recursividade é o último objetivo do corpo.
- Procedimento iterativo: se contiver apenas factos e cláusulas iterativas.

O cut (!) permite afetar o comportamento procedimental dos programas

- Principal função: reduzir o espaço de procura podando dinamicamente a árvore de pesquisa
  - Reduz tempo de computação
  - Reduz espaço, pois alguns pontos de escolha deixam de ser necessários

O cut sucede e compromete o Prolog com todas as escolhas feitas desde que o objetivo pai foi unificado com a cabeça da cláusula onde o cut ocorre.

Em caso de retrocesso no cut, a pesquisa continuará a partir da última escolha feita antes da escolha desta cláusula.

fail: predicado que provoca falha.

Cut verde:

- não altera o significado do programa: o mesmo conjunto de soluções é encontrado com ou sem o cut.
- corta apenas caminhos de computação que não levam a novas soluções.

Cut vermelo:

- se retirado, altera o significado do programa: o conjunto de soluções será diferente.
- a ordem das cláusulas passa a ser fixa.

A omissão de condições transforma cuts verdes em vermelhos:

- Cut verde:

```
minimum(X, Y, X) :- X =< Y, !.
minimum(X, Y, Y) :- X > Y, !.
```

- Cut vermelho:

```
minimum(X, Y, X) :- X =< Y, !.
minimum(X, Y, Y).
```

Inspeção de estruturas:

- functor(Term, F, Arity) / arg(N, Term, Arg)
  - Decomposição de termos
  - Criação de termos
- Term =.. List
  - Construir um termo a partir de uma lista
  - Construir uma lista a partir de um termo

Predicados meta-lógicos:

- var(Term) / nonVar(Term)
  - Verificam se Term está ou não instanciado
- == / \==
  - Verificam se dois termos são ou não idênticos

Meta-variável: variável usada como um objetivo no corpo de uma cláusula. Durante a computação, aquando da sua invocação a variável deverá estar instanciada com um termo (se não, erro).

Strings e Códigos ASCII

- String (entre aspas) corresponde a uma lista de inteiros que são os códigos ASCII de cada carácter na string
- name(X, Ys): converte átomo X na lista Ys com códigos ASCII dos caracteres de X
- put(N): escreve carácter cujo código ASCII é N
- get0(N): lê carácter e unifica o seu código ASCII com N
- get(N): lê carácter não branco

Ficheiros:

- see(F): abre canal de leitura para o ficheiro F. Leituras passam a ser feitas a partir de F.
- tell(F): abre canal de escrita para o ficheiro F. Escritas passam a ser feitas para F.
- seeing(F)/telling(F): F é unificado com o nome do ficheiro do canal corrente.
- seen/told: fecha o canal corrente.

Acesso e Manipulação do Programa

- assertz(Clause)/asserta(Clause): adiciona Clause como última/primeira cláusula do procedimento.
- retract(C): remove a primeira cláusula que unifica com C.
- consult(File): lê e adiciona (assert) as cláusulas do ficheiro File.
- reconsult(File): faz retract das cláusulas antes de consult.
- clause(Head, Body): procura cláusula cuja cabeça unifica com Head.

5 Prolog Avançado

findall(Term, Goal, Bag): unifica Bag com a lista das instâncias de Term para as quais Goal é satisfeito.

- Todos os X para os quais call(Goal), X=Term? é satisfeito.
- Term e Goal tipicamente partilham variáveis.

bagof(Term, Goal, Bag): idêntico ao findall, mas são encontradas soluções alternativas para as variáveis em Goal.

setof(Term, Goal, Bag): soluções ordenadas, sem duplicados (conjunto).

Meta-Interpretador: interpretador de uma linguagem escrito na própria linguagem.

- Em Prolog, é fácil construí-los porque não há distinção entre programa e dados.
- Interesse em desenvolvê-los:
  - Implementar diferentes estratégias de pesquisa da solução.
  - Incluir capacidade de explicação.
  - Incluir facilidades acrescidas de traçagem, teste e debugging.

Definição de operadores:

- Nome (um átomo), tipo (classe e associatividade) e prioridade (inteiro entre 1 e 1200).
- :- op(Prioridade, Tipo, Nome).
- Tipos de Operadores: fx, fy, xfx, xfy, yf, xf (y tem prioridade; x e y indicam o lado da associação).

6 Programação em Lógica com Restrições

A PLR, ou CLP (Constraint Logic Programming), é uma classe de linguagens de programação combinando:

- Declaratividade da programação em lógica.
- Eficiência da resolução de restrições.

Aplicações principais na resolução de problemas de pesquisa/otimização combinatória.

Um Problema de Satisfação de Restrições - PST, ou CSP (Constraint Satisfaction Problem) - é modelado através de:

- Variáveis representando diferentes aspetos do problema, juntamente com os seus domínios.
- Restrições que limitam os valores que as variáveis podem tomar dentro dos seus domínios.

A solução de um CSP é uma atribuição de um valor (do seu domínio) a cada variável, de forma a que todas as restrições sejam satisfeitas.

Mais formalmente, um CSP é um tuplo <V, D, C>:

- V = $\{x_1, x_2, ..., x_n\}$ é o conjunto de variáveis.
- D é uma função que mapeia cada variável de V num conjunto de valores (domínio).
- C = $\{C_1, C_2, ..., C_n\}$ é o conjunto de restrições que afetam um subconjunto arbitrário de variáveis de V.

As restrições de um CSP/COP podem ser:

- Rígidas (Hard Constraints): são aquelas que têm obrigatoriamente de ser cumpridas. Todas as restrições num CSP são deste tipo.
- Flexíveis (Soft Constraints): são aquelas que podem ser quebradas.

Resolução de um CSP:

- Declarar as variáveis e os seus domínios (finitos).
- Especificar as restrições existentes.
- Pesquisar para encontrar a solução.

Restrições - construção:

- Retrocesso: "generate and test"
- Propagação: "forward checking"

Atribuição de valores a variáveis:

- Label: uma label é um par Variável-Valor, onde Valor é um dos elementos do domínio da Variável.

Solução parcial em que algumas das variáveis já têm valores atribuídos:

- Compound Label: conjunto de labels incluindo variáveis distintas.

A aridade de uma restrição C é o número de variáveis sobre o qual a restrição está definida, ou seja, a cardinalidade do conjunto Vars(C). Todas as restrições podem ser convertidas em restrições binárias.

Conversão para restrições binárias: uma restrição n-ária C, definida por k compound labels nas suas variáveis X_1 a X_n, é equivalente a n restrições binárias, B_i, através da adição de uma nova variável Z, cujo domínio é o conjunto 1 a k.

Forward Checking:

- Verifica as restrições entre a variável corrente (e anteriores) e as variáveis futuras.
- Quando um valor é atribuído à variável corrente, qualquer valor de uma variável futura que entre em conflito com esta atribuição é (temporariamente) removido do seu domínio.

7 Programação em Lógica com Restrições no SICStus Prolog

Domínio das variáveis:

- Uma variável pode ter o seu domínio declarado usando in/2 e um intervalo (Constant Range).
- Pode ainda ser usado in_set/2. O seu segundo argumento é um Finite Domain Set, que pode ser obtido a partir de uma lista com o predicado list_to_fdset(+List, -FD_Set).
- Para declara o mesmo domínio a uma lista de variáveis pode ser usado o predicado domain(+List_of_Variables, +Min, +Max).