Bases de Dados Dedutivas e Datalog

Datalog

- A ideia base do Datalog é usar cláusulas de Horn -- regras “if-then” – como linguagem de consulta a bases de dados relacionais.
- As relações são representadas por *predicados*, e.g. Climbers, Climbs e Routes são interpretados como predicados com aridade fixa.
- Os argumentos têm uma interpretaçao *posicional*; e.g. Climbers(X,”Bridget”, “EXP”,”33”). Os argumentos podem ser constantes (e.g. “Bridget”, “EXP”, e “33”) ou variáveis (e.g. X).
  - Vamos usar letras maiúsculas para var, e minúsculas para constantes

Valores de verdade

- Um predicado é *ground* se todos os seus argumentos forem constantes. Predicados ground têm valores de verdade, que representam o facto do "tuplo" pertencer à relação.

Climbers

CId	Cname	Skill	Age
123	edmund	exp	80
214	arnold	beg	25
313	bridget	exp	33
212	james	med	27

Climbers(313,bridget, exp,33) é verdade.

Climbers(518,jeremy,exp,17) é falso.

- Os predicados também podem ser negados:

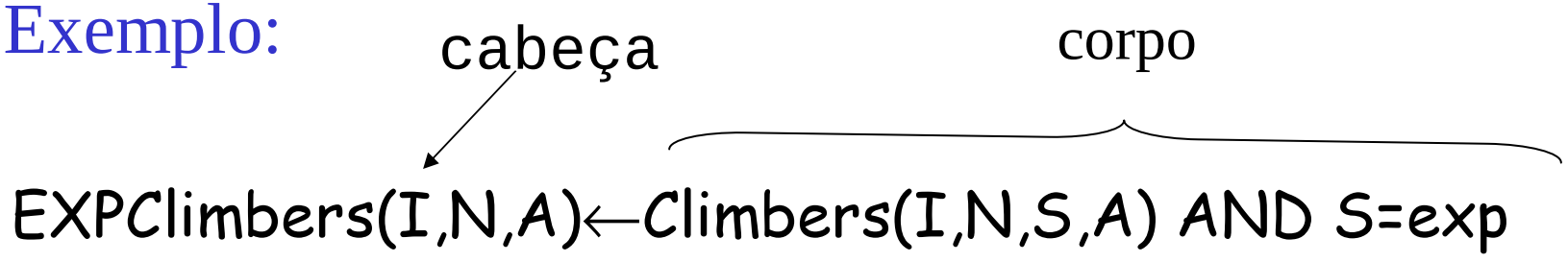
NOT Climbers(518,jeremy,exp,17) é verdadeiro.

Predicados “Aritméticos”

- Vamos representar condições e usaremos os predicados $<, \leq, >, \geq, =, \neq$. I.e. $<(5,7)$ é verdadeiro mas $<(7,5)$ é falso.
- Notem que ao contrário dos predicados “relacionais”, os predicados aritméticos são infinitos!

Regras Datalog

- Uma regra é da forma $p \leftarrow q$, onde p é um predicado relacional chamado *cabeça* e q é uma conjunção de predicados (*subgolos*) chamada corpo.
- Exemplo:



- Quando uma variável não é usada, pode ser substituída por "_" (variável *anónima*).

EXPClimbers(N)←Climbers(_,N,exp,_)

Alguns exemplos…

- Os nomes dos climbers com mais de 32 anos.

  OLD(N) ←Climbers(I,N,S,A) AND A>32

- Os nomes dos climbers que escalaram a via 1.

  Route1(N)←Climbers(I,N,_,_) AND Climbs(I, 1,_,_)

- Os nomes dos climbers com menos de 40 anos que escalaram uma via de grau maior do que 5.

  Rating5(N) ← Climbers(I,N,_,A) AND Climbs(I, R,_,_)
  AND Routes(R,_,_,Ra,_) AND Ra>5 AND A<40

- Notem a interpretação posicional dos atributos!

Segurança

- Uma regra é *segura* se cada variável ocorre pelo menos uma vez num predicado relacional positivo no corpo.
- Algumas regras inseguras:

```
Likes(X,Y)←Starved(X)
Lazy(X) ←NOT Climbers(_,X,_,_)
```

- Algumas regras seguras:

```
Likes(X,Y)←Starved(X) AND Food(Y)
Lazy(X) ←Person(X) AND NOT Climbers(_,X,_,_)
```

(e todas as outras que vimos até agora).

Consulta Datalog

- Uma *consulta (query)* é um conjunto de uma ou mais regras. Uma regra com corpo vazio é um *facto* (predicado relacional positivo ground).

```
Student (123,j.smith,compsci)
Student(456,k.tappet,french)
Offers(cookery,baking)
Offers(compsci,compilers)
Enroll(123,baking)
Enroll(012,compilers)

InterestedIn(X,S) ←Student(X,Y,S)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(S,Z)
```

A consulta na Álgebra Relacional

- A consulta anterior corresponde à seguinte expressão da álgebra relacional:

$$\pi_{[1][3]}\text{Student} \cup \pi_{[1][4]}(\sigma_{[2]=[3]}(\text{Enroll} \times \text{Offers}))$$

- Qual esperaria que fosse o output desta consulta?

Predicados Intencionais versus Extensionais

- Predicados *extensionais* são aqueles cujas relações estão guardadas na bd; predicados *intencionais* são aqueles que são calculados aplicando uma ou mais regras.
  - Student, Offers, e Enroll são extensionais
  - InterestedIn é intencional
- Predicados Extensionais não podem nunca aparecer na cabeça de uma regra.

Outro exemplo

Parent(mary,jane)
Parent(jane,fred)
Parent(ed,bob)
Parent(bob,fred)
Parent(fred,jill)

Ancestor(X,Y)←Parent(X,Y)
Ancestor(X,Y)←Parent(X,Z) AND Ancestor(Z,Y)

- Podemos traduzir para a álgebra relacional?
- Qual esperaria que fosse o output desta consulta?
- Que predicados são EDB e IDB?

Significado das regras Datalog

- Considerar todas as possíveis atribuições de valores a variáveis. Para cada atribuição que torne todos os subgolos verdadeiros, o tuplo correspondente à cabeça é verdadeiro e adicionado ao resultado.
- Exemplo: X=012, Z=compilers, S=compsci

Offers(compsci,compilers)
Enroll(012,compilers)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(S,Z)
Logo InterestedIn(012,compsci) é acrescentado.

Outra forma de definir significado…

- O método de "atribuição" para definir o significado considera atribuições a variáveis "sem significado". Por exemplo: X=compilers, Z=012, S=f.dunham

InterestedIn(X,S) ← Enroll(X,Z) AND Offers(Z,S)

- Um outro método é considerar apenas os conjuntos de tuplos em cada subgolo relacional não negado, e ver as atribuições a variáveis "consistentes". Se todos os subgolos são verdadeiros (negados e aritméticos tb), então o tuplo na cabeça é acrescentado ao resultado.
- Isto sugere uma implementação usando a AR!

Uma consulta interessante escrita "incorrectamente"

- É fácil escrever consultas que não expressam a nossa intenção. E.g.

  Single(X)←Person(X) AND NOT Married(X,Y)

- Qual o significado desta consulta? Se a intenção era obter todas as pessoas que não estão casadas, como devemos escrever a consulta?
- Esta consulta também não é segura!

AR versus Datalog

- O exemplo Ancestor é chamado *recursivo* porque a definição de ancestor depende de si própria (directamente). Isto não pode ser simulado na AR, e temos que adicionar um operador de ponto fixo à álgebra para o simular.
- Se os subgolos não puderem ser negados, não podemos emular diferença de conjuntos em Datalog.
- No entanto, se os subgolos puderem ser negados podemos simular qualquer expressão da AR em Datalog.

Simulação da AR em Datalog

- Intersecção: simular por uma regra com cada relação como subgolo. E.g. relembrar Climbers e Hikers definidos anteriormente.

$$\text{Climbers} \cap \text{Hikers}$$

seria escrito como

CandH(I,N,S,A)←Climbers(I,N,S,A) AND Hikers(I,N,S,A)

- Diferença: simular por uma regra com cada relação como subgolo, com o segundo subgolo negado. Logo Climbers - Hikers fica

CnotH(I,N,S,A)←Climbers(I,N,S,A) AND NOT Hikers(I,N,S,A)

Simular AR, cont.

- União: simular por duas regras, cada uma com um único subgolo consistindo numa das relaçoes. Assim seria escrito como $\text{Climbers} \cup \text{Hikers}$

$$\text{CorH}(I,N,S,A) \leftarrow \text{Climbers}(I,N,S,A)$$
$$\text{CorH}(I,N,S,A) \leftarrow \text{Hikers}(I,N,S,A)$$

- Projecção: simular por uma regra, cuja cabeça usa as variáveis correspondendo aos atributos a projectar. Assim $\pi_{Name,Age}\text{Climbers}$ seria escrito como $\text{Result}(N,A) \leftarrow \text{Climbers}(\_,N,\_,A)$

Simulação da Selecção

- Se a condição é uma conjunção de átomos aritméticos, torna-se fácil: juntar cada átomo como subgolo. Assim

fica $\sigma_{Name="Bridget" \wedge Age>33}\text{Climbers}$

Result(I,N,S,A)←Climbers(I,N,S,A) AND N= bridget AND Age>30

- OU pode ser simulado usando a união; relembrem a equivalência

$$\sigma_{C \vee D} R \equiv \sigma_C R \cup \sigma_D R$$

Simulação da Selecção, cont.

- Agora, relembrem da lógica que qualquer expressão envolvendo e, ou e não pode ser posta na *forma normal disjuntiva*: um OU de conjunções, cada qual é um E de comparações.

$$\sigma_{\neg(Age\leq 30 \vee Name=\text{"}Bridget\text{"}) \vee Skill=EXP} \text{Climbers} \approx$$

$$\sigma_{(\neg Age\leq 30 \wedge \neg Name=\text{"}Bridget\text{"}) \vee Skill=EXP} \text{Climbers} \approx$$

$$\sigma_{(Age>30 \wedge Name\neq\text{"}Bridget\text{"}) \vee Skill=EXP} \text{Climbers} \approx$$

$$\sigma_{Age>30 \wedge Name\neq\text{"}Bridget\text{"}} \text{Climbers} \cup \sigma_{Skill=EXP} \text{Climbers}$$

Simulação de Produto e Junção

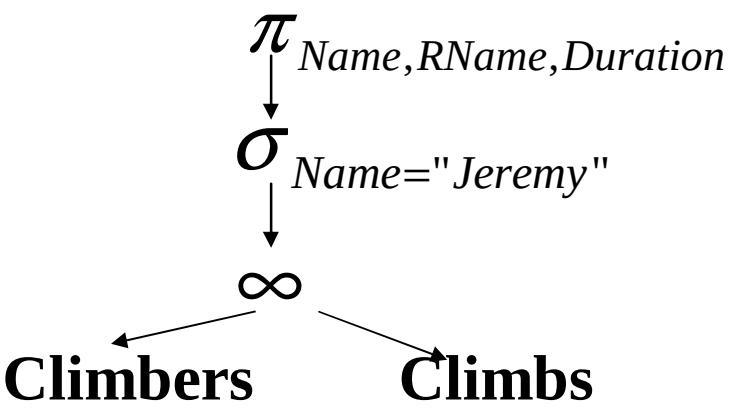
- O produto de duas relações é expresso por uma única regra com ambas as relações como subgolos; todas as variáveis das relações aparecem na cabeça:
  R(A,B)←S(A) AND T(B)
- Uma junção é uma selecção por igualdade e projecção sobre um produto; a igualdade pode ser expressa reusando as variáveis:

  R(A,B,C) ←S(A,B) AND T(D,C) AND B=D ou
  R(A,B,C) ←S(A,B) AND T(B,C)

Simulação operações múltiplas da AR

- Criação da “árvore de operadores”:

$$\pi_{Name,RName,Duration}\sigma_{Name="Jeremy"}(\text{Climbers}\infty\text{Climbs})$$

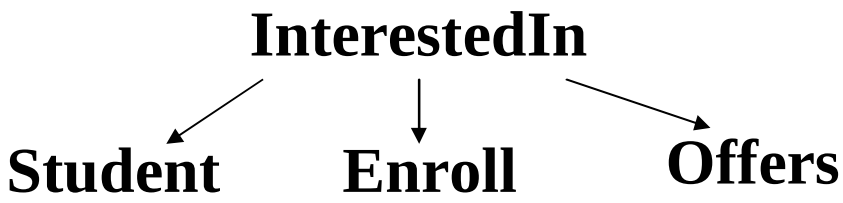


- Criar um predicado IDB para cada nó interior, e escrever a regra correspondente.
- O IDB correspondente à raíz é o resultado.

Datalog: Grafo de Dependências

- As folhas correspondem a predicados relacionais; Existe um ramo de A para B se B aparece como subgolo (positivo ou negativo) numa regra com cabeça A. O ramo é anotado com "-" para subgolos negados:

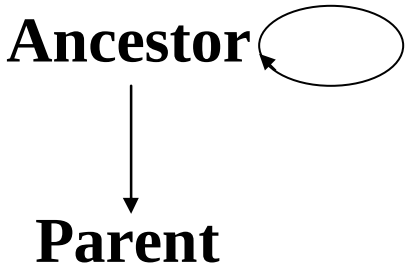
InterestedIn(X,S) ←Student(X,Y,S) InterestedIn(X,S) ← Enroll(X,Z) AND Offers(Z,S)



Grafos de Dependências Cíclicos

- Um grafo de dependências cíclico indica uma consulta recursiva.
- Relembre o exemplo do ancestor:

Ancestor(X,Y)←Parent(X,Y)
Ancestor(X,Y)←Parent(X,Z) AND Ancestor(Z,Y)



Avaliação de Datalog na AR

- Assuma por agora programas Datalog não recursivos e sem predicados negados.
- A avaliação de predicados IDB procede de forma “bottom-up” começando nas folhas do grafo de dependências para os subgolos estarem completamente avaliados quando são usados. (Notem que as folhas correspondem aos predicados EDB!)

Avaliação de Datalog na AR, cont.

- Procedimento para avaliar uma regra (sem negação):
  - Tomar o produto dos subgolos relacionais (não-aritméticos)
  - Formar uma selecção do produto com uma condição que iguale os argumentos com a mesma variável e todos os predicados aritméticos
  - Projectar nas variáveis que aparecem na cabeça
- Para cada predicado IDB R, tomar a união das expressões de regras com cabeça R.

Exemplos

Result(N)←Climbers(I,N,_,A) AND A<40 AND
Climbs(I,R,_,_) AND Routes(R,_,_,Ra,_) AND Ra>5

$$\pi_{[2]}(\sigma_{[1]=[5]\wedge[6]=[9]\wedge[4]<40\wedge[12]>5}(\text{Climbers} \times \text{Climbs} \times \text{Routes}))$$

InterestedIn(X,S) ←Student(X,Y,S)
InterestedIn(X,S) ← Enroll(X,Z) AND Offers(Z,S)

$$\pi_{[1][3]}\text{Student} \cup \pi_{[1][4]}(\sigma_{[2]=[3]}(\text{Enroll} \times \text{Offers}))$$

NotSingle(X) ←Married(X,Y)
NotSingle(X) ←Married(Y,X)

$$\pi_{[1]}\text{Married} \cup \pi_{[2]}\text{Married}$$

Tratamento de subgolos negados

- Suponha que NOT R(X,Y,Z) aparece como subgolo negado numa consulta. Seja DOM={qualquer símbolo que ocorra na regra} U {qualquer símbolo que apareça em qualquer instancia relacional que apareça num subgolo da regra}.
- É suficiente "avaliar" NOT R(X,Y,Z) como (DOM X DOM X DOM)-R
- No entanto, existem problemas quando a negação é combinada com a recursão!

Exemplo

A versão correcta do exemplo que obtivesse todas as pessoas solteiras na base de dados é:

NotSingle(X) ←Married(X,Y)
NotSingle(X) ←Married(Y,X)
Single(X)←Person(X) AND NOT NotSingle(X)

Como é que avaliamos esta consulta?

Seja DOM={td. simb. em Person}U{td. simb. em NotSingle}

$$\pi_{[1]}(\sigma_{[1]=[2]}(\text{Person}\times(\text{DOM}-\text{NotSingle})))$$

Consultas recursivas (sem negação) – avaliação “Naïve”

- Sejam R, S... predicados IDB ocorrendo num único ciclo do grafo de dependências.

R=∅, S= ∅
while there is a change to R, S,… do
  R=R ∪ {evaluation of R}
  S=S ∪ {evaluation of S}

Exemplo

Parent(mary,jane)
Parent(jane,fred)
Parent(ed,bob)
Parent(bob,fred)
Parent(fred,jill)

Ancestor(X,Y)←Parent(X,Y)

Ancestor(X,Y)←Parent(X,Z)
AND Ancestor(Z,Y)

Avaliaçao de Ancestor:

$$\text{Parent} \cup (\pi_{[1],[4]} \sigma_{[2]=[3]} (\text{Parent} \times \text{Ancestor}))$$

1. Ancestor = ø
2. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
(fred,jill)}

Exemplo, cont.

$$\text{Parent} \cup (\pi_{[1],[4]} \sigma_{[2]=[3]} (\text{Parent} \times \text{Ancestor}))$$

3. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
(fred,jill), (mary, fred),(jane,jill),(ed.fred)
(bob,jill)}
4. Ancestor = {(mary,jane),(jane,fred),(ed,bob),(bob,fred),
(fred,jill), (mary, fred),(jane,jill),(ed,fred)
(bob,jill), (mary,jill), (ed,jill)}

Negação em Regras Recursivas

- Problema: qual é a semântica?

  Por exemplo, suponha um predicado IDB de R(0):

  $P(X) \leftarrow R(X)$ AND NOT $Q(X)$

  $Q(X) \leftarrow R(X)$ AND NOT $P(X)$

- Existem duas respostas "correctas": {R(0), P(0), ∅} e {R(0),Q(0), ∅}. Ambas são mínimas no sentido em que não podemos eliminar nada e obter uma resposta correcta (i.e. {R(0)} é inconsistente).

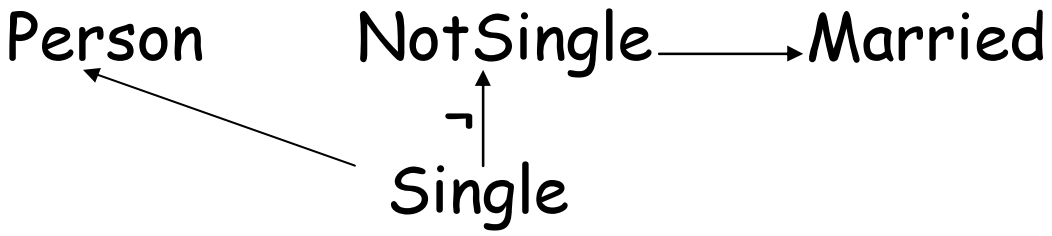
Negação Estratificada

- Técnica para atribuir um único significado a *alguns* programas Datalog seguros com negação.
- Funciona pela divisão dos predicados em “estratos”, que são ordenados linearmente. Cada estrato tem que estar completamente avaliado antes de se avaliar o próximo estrato.
- Não poderia tratar o programa anterior pois este tem um ciclo de ramos “negativos” no grafo de dependências.

Negação estratificada (cont.)

- Um programa está estratificado sse sempre que existe uma regra com cabeça p onde q ocorre como subgolo negado, não existe um caminho de q para p no grafo de dependências.

NotSingle(X) ←Married(X,Y)
NotSingle(X) ←Married(Y,X)
Single(X)←Person(X) AND NOT NotSingle(X)



Algoritmo de marcação estratificada

```
for each predicate p do stratum(p)=1
repeat until no changes to any stratum or some stratum
                    exceeds the number of predicates
    for each rule r with head p do begin
        for each negated subgoal of r with predicate q do
            stratum(p):= max(stratum(p), 1+stratum(q))
        for each positive subgoal or r with predicate q do
            stratum(p):= max(stratum(p), stratum(q))
        end for
end repeat
```

Stratum(1): Person, Married, NotSingle

Stratum(2): Single

Resumo: AR versus Datalog

- Se os subgolos puderem ser negados podemos simular qualquer expressão da AR em Datalog. (sem subgolos negados não podemos tratar a “-”.)
- Para simular consultas Datalog “recursivas”, temos que adicionar um operador de ponto fixo à AR.
- Datalog com subgolos negados E recursão pode resultar em programas com mais do que um “modelo mínimo” – a negação estratificada permite avaliar alguns destes programas.

Outro exemplo

sibling(X,Y) :- parent(X,Z), parent(Y,Z), X≠Y.

cousin(X,Y) :- parent(X,Xp), parent(Y,Yp), sibling(Xp,Yp).

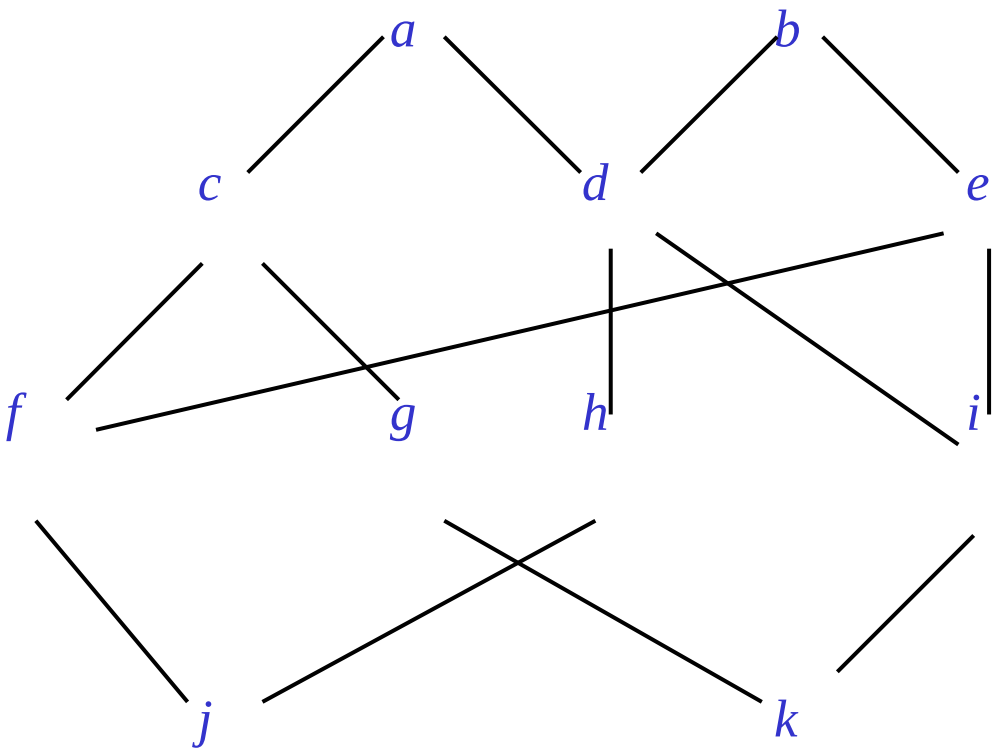
cousin(X,Y) :- parent(X,Xp), parent(Y,Yp), cousin(Xp,Yp).

related(X,Y) :- sibling(X,Y).

related(X,Y) :- related(X,Z), parent(Y,Z).

related(X,Y) :- related(Z,Y), parent(X,Z).

Factos para o predicado *parent*



Monotonia das operações da AR

- Para garantir a convergência do algoritmo as operações envolvidas tem que ser monótonas.

  Teorema: As operações reunião, selecção, projecção e produto são monótonas.