Bases de données

M1 sciences et technologies, mention informatique

récursivité et négation

ou : comment exprimer "peut-on aller de Odéon à Chatelet sans passer par St-Michel ?"

métro	line	station	connexion
	4	St.-Germain	Odéon
	4	Odéon	St.-Michel
	4	St.-Michel	Chatelet
	1	Chatelet	Louvres
	1	Louvres	Palais-Royal
	1	Palais-Royal	Tuileries
	1	Tuileries	Concorde
	9	Pont de Sèvres	Billancourt
	9	Billancourt	Michel-Ange
	9	Michel-Ange	Iéna

patrick.marcel@univ-tours.fr

plan

1. récursivité :
   - syntaxe
   - sémantique théorie des modèles
   - sémantique point fixe/opérationnelle
2. négation et récursivité :
   - syntaxe
   - les problèmes
   - programmes stratifiables

récursivité

exemple

métro	line	station	connexion
	4	St.-Germain	Odéon
	4	Odéon	St.-Michel
	4	St.-Michel	Chatelet
	1	Chatelet	Louvres
	1	Louvres	Palais-Royal
	1	Palais-Royal	Tuileries
	1	Tuileries	Concorde
	9	Pont de Sèvres	Billancourt
	9	Billancourt	Michel-Ange
	9	Michel-Ange	Iéna
	9	Iéna	F.D. Roosvelt
	9	F.D. Roosvelt	République
	9	République	Voltaire

“quelles sont les stations accessibles depuis Odéon ?”

“peut-on aller de Odéon à Chatelet ?”

récursivité

étude de ces requêtes en adoptant

- le point de vue logique
- l'approche non nommée
- le langage à base de règles

le langage étudié s'appelle *datalog*

exemple

accessible(x,x) ←métro(l,x,y)

accessible(x,x) ←métro(l,y,x)

accessible(x,y) ←accessible(x,z), métro(l,z,y)

résultat_1(x) ←accessible("Odéon",x)

résultat_2() ←accessible("Odéon","Chatelet")

syntaxe

un règle datalog est une expression de la forme

$$R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

où

- les R_i sont des noms de relations
- les u_i sont des tuples libres
- chaque variable de u_1 apparaît au moins une fois dans $u_2, \ldots, u_n$

programme

un programme datalog P est un ensemble fini de règles

$adom(P)$ est l'ensemble des constantes de P

$adom(P,I) = adom(P) \cup adom(I)$ pour une instance I

$edb(P)$ l'ensemble de relations apparaissant seulement dans le corps

$idb(P)$ l'ensemble de relations apparaissant dans la tête

$sch(P) = edb(P) \cup idb(P)$

sémantique

plusieurs approches différentes et équivalentes de définir le sens d'un programme

1. théorie des modèles
2. point fixe
3. théorie de la preuve

on étudiera les 2 premières

théorie des modèles

les règles sont vues comme des formules logiques exprimant les propriétés du résultat attendu

le résultat est le plus petit ensemble de faits qui satisfait toutes les formules

théorie des modèles

$q : R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$

I une instance

v une valuation, on note $R(v(u))$ $R(u)$ où chaque variable x est remplacée par $v(x)$

$I \models q$ si pour chaque valuation v telle que
$R_2(v(u_2)) \in I, \ldots, R_n(v(u_n)) \in I$ alors $R_1(v(u_1)) \in I$

exemple

q: résultat(x) ←films(y,x,2005)

I = {résultat(ki-duk), films(locataires,ki-duk,2005)}

J = {films(locataires,ki-duk,2005)}

K = {résultat(ki-duk)}

L = ∅

I et K et L satisfont q, J ne satisfait pas q

modèle d'un programme

soit P un programme et I une instance sur $sch(P)$

$I \models P$ si $\forall q \in P, I \models q$

I est appelé *modèle* de P

pour une instance I sur $edb(P)$, la sémantique de P est la plus petite instance $P(I)$ contenant I telle que $P(I) \models P$

exemple

soit P le programme composé de la règle

q: résultat(x) ←films(y,x,2005)

I = {films(locataires,ki-duk,2005)}

J = {résultat(ki-duk), films(locataires,ki-duk,2005)}

K = {résultat(ki-duk), films(locataires,ki-duk,2005), résulat(lucas)}

J est la plus petite instance contenant I satisfaisant P

questions

est-ce que $P(I)$ peut être infini ?

est-ce que $P(I)$ peut ne pas exister ?

comment calculer $P(I)$?

un modèle

soit $B(P,I)$ l'instance sur $sch(P)$ telle que

- pour chaque $R \in edb(P)$, $R(u) \in B(P,I)$ si $R(u) \in I$
- pour chaque $R \in idb(P)$, tout fait $R(u)$ utilisant des constantes de $adom(P,I)$ est dans $B(P,I)$

$B(P,I)$ est un modèle de P contenant I

théorème

soit P un programme, I une instance sur $edb(P)$ et X l'ensemble des modèles de P contenant I

alors

1. $\cap X$ est le modèle minimal de P contenant I
2. $adom(P(I)) \subseteq adom(P,I)$
3. pour tout $R \in edb(P)$, $P(I)(R) = I(R)$

point fixe

la sémantique du programme est la solution d'une équation point fixe

cette approche consiste à "itérer" une requête jusqu'à ce qu'un point fixe soit atteint

conséquence immédiate

soit P un programme, K une instance sur $edb(P)$

un fait A est une *conséquence immédiate* pour K et P si

1. $A \in K(R)$ avec $R \in edb(P)$, ou
2. $A \leftarrow A_1, \ldots, A_n$ est une instanciation d'une règle de P, et chaque $A_i \in K$

opérateur de conséquence immédiate

T_P l'opérateur de conséquence immédiate de P

c'est une fonction de $inst(sch(P))$ dans $inst(sch(P))$ telle que

pour chaque K, $T_P(K)$ est l'ensemble des faits qui sont conséquences immédiates de P et K

exemple

soit P le programme composé de la règle

q: résultat(x) ←films(y,x,2005)

I = {films(locataires,ki-duk,2005)}

$T_P(I)$ = {résultat(ki-duk),films(locataires,ki-duk,2005) }

propriétés d'un opérateur T

1. T est monotone ($I \subseteq J \Rightarrow T(I) \subseteq T(J)$)
2. K est un point fixe de T si $T(K) = K$

lemme

soit P un programme datalog

1. l'opérateur T_P est monotone
2. une instance K sur $edb(P)$ est un modèle de P ssi $T_P(K) \subseteq K$
3. chaque point fixe de T_P est un modèle de P

théorème

quels que soient P et I, T_P a un plus petit point fixe contenant I égal à $P(I)$

calcul du pppf

pour une instance I sur $edb(P)$, on peut calculer $T_P(I), T_P^2(I), T_P^3(I), \ldots$

on a $I \subseteq T_P(I) \subseteq T_P^2(I) \subseteq T_P^3(I) \subseteq \ldots \subseteq B(P,I)$

N est la cardinalité de $B(P,I)$

la séquence $\{T_P^i(I)\}_i$ atteint un point fixe après au plus N étape $(\forall i \geq N, T_P^i(I) = T_P(I))$

appelons ce point fixe $T_P^\omega(I)$

théorème

soit P un programme datalog et I une instance sur $edb(P)$

$$T_P^\omega(I) = P(I)$$

exemple

soit le programme P

$T(x,y) \leftarrow G(x,y)$

$T(x,y) \leftarrow G(x,z),\ T(z,y)$

soit l'instance initiale $I=\{G(1,2),\ G(2,3),\ G(3,4),\ G(4,5)\}$

exemple

on a
$T_P(I) = I \cup \{T(1,2), T(2,3), T(3,4), T(4,5)\}$

$T_P^2(I) = T_P(I) \cup \{T(1,3), T(2,4), T(3,5)\}$

$T_P^3(I) = T_P^2(I) \cup \{T(1,4), T(2,5)\}$

$T_P^4(I) = T_P^3(I) \cup \{T(1,5)\}$

$T_P^5(I) = T_P^4(I)$

donc $T_P^\omega(I) = T_P^4(I)$

récursivité et négation

Hypothèse du monde clos

Closed-World Assumption (CWA)

pour I une instance de base de données :

- tout ce qui se trouve dans I est vrai
- ce qui ne se trouve pas dans I est faux

syntaxe de Datalog$^\neg$

une règle Datalog$^\neg$ est une expression de la forme :

$$S(u) \leftarrow L_1, \ldots, L_n$$

- S est un nom de relation
- u est un tuple libre
- les L_i sont des litéraux de la forme $R(v)$ ou $\neg R(v)$
  - R est un nom de relation
  - v est un tuple libre
- chaque variable apparait dans un litéral positif

un programme Datalog$^\neg$ est un ensemble de règles

sémantique théorie des modèles

exemple : soit P le programme

$$p(a) \leftarrow \neg q(a)$$

$$q(a) \leftarrow \neg p(a)$$

P admet 2 modèles minimaux : $\{p(a)\}$ et $\{q(a)\}$
lequel choisir ?

un programme n'a pas forcément un seul modèle minimal

sémantique point fixe

pour une instanciation d'une règle r, on définit :

$$T_P(I) = \{\text{tête(r)} \mid \text{si } L_i = A_i \text{ alors } A_i \in I, \\ \text{si } L_i = \neg A_i \text{ alors } A_i \notin I\}$$

sémantique point fixe

pour une instanciation d'une règle r, on définit :

$$T_P(I) = \{\text{tête(r)} \mid \text{si } L_i = A_i \text{ alors } A_i \in I, \\ \text{si } L_i = \neg A_i \text{ alors } A_i \notin I\}$$

mais...

- T_P peut ne pas avoir de point fixe
  exemple : $P = \{p(a) \leftarrow \neg p(a)\}$

sémantique point fixe

pour une instanciation d'une règle r, on définit :

$$T_P(I) = \{\text{tête(r)} \mid \text{si } L_i = A_i \text{ alors } A_i \in I, \\ \text{si } L_i = \neg A_i \text{ alors } A_i \notin I\}$$

mais...

- T_P peut ne pas avoir de point fixe
  exemple : $P = \{p(a) \leftarrow \neg p(a)\}$
- T_P peut avoir plusieurs points fixes, sans converger
  exemple : $P = \{p(a) \leftarrow \neg q(a); q(a) \leftarrow \neg p(a)\}$

programmes stratifiables

un programme Datalog$^{\neg}$ est *stratifiable* si

pour chaque relation R, il est possible de calculer R avant que sa négation soit utilisée

programmes stratifiables

un programme Datalog$^{\neg}$ est *stratifiable* si

pour chaque relation R, il est possible de calculer R avant que sa négation soit utilisée

exemple : le programme P suivant est stratifiable

$T(x,y) \leftarrow G(x,y)$
$T(x,y) \leftarrow G(x,z), T(z,y)$
$CT(x,y) \leftarrow \neg T(x,y)$

stratification

la stratification d'un programme Datalog$^\neg$ P est *une suite de programme Datalog*$^\neg$ $P_1, \ldots, P_n$

telle que pour une fonction σ de $idp(P)$ dans [1,n]

- $\{P_1, \ldots, P_n\}$ est une partition de P
- pour chaque relation R, les règles définissant R sont dans $P_{\sigma(R)}$
- si $R(u) \leftarrow \ldots, R'(v) \ldots$ est une règle de P et R' une relation de l'idb, alors $\sigma(R') \leq \sigma(R)$
- si $R(u) \leftarrow \ldots, \neg R'(v) \ldots$ est une règle de P et R' une relation de l'idb, alors $\sigma(R') < \sigma(R)$

stratification

- chaque P_i est appelé une *strate* de P
- σ est la fonction de stratification
- tout programme n'est pas stratifiable
- un programme stratifiable admet plusieurs stratifications
- toutes les stratifications sont équivalentes

exemple

$$
\begin{array}{lllll}
r_1 & S(x) & \leftarrow & R_1(x), \neg R(x) \\
r_2 & T(x) & \leftarrow & R_2(x), \neg R(x) \\
r_3 & U(x) & \leftarrow & R_3(x), \neg T(x) \\
r_4 & V(x) & \leftarrow & R_4(x), \neg S(x), \neg U(x)
\end{array}
$$

ce programme admet 5 stratifications

1. $\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}$
2. $\{r_2\}, \{r_1\}, \{r_3\}, \{r_4\}$
3. $\{r_2\}, \{r_3\}, \{r_1\}, \{r_4\}$
4. $\{r_1, r_2\}, \{r_3\}, \{r_4\}$
5. $\{r_2\}, \{r_1, r_3\}, \{r_4\}$

exemple

le programme suivant n'est pas stratifiable

$$p(x) \leftarrow q(x)$$
$$q(x) \leftarrow p(x)$$

graphe de précédence

pour vérifier si un programme P est stratifiable

construire le *graphe de précédence* G_P de P

- les noeuds de G_P sont les relations de l'idb,
- si $R(u) \leftarrow \ldots, R'(v) \ldots$ est une règle de P alors (R', R) est une arrête de G_P labellisée $+$ (arrête positive),
- si $R(u) \leftarrow \ldots, \neg R'(v) \ldots$ est une règle de P alors (R', R) est une arrête de G_P labellisée $-$ (arrête négative)

un programme est stratifiable ssi son graphe de précédence ne contient aucun cycle contenant une arrête négative.

sémantique des programmes stratifiables

sémantique d'un programme P de stratification $P_1, \ldots, P_n$ sur une instance I :

- $I_0 = I$
- $I_i = I_{i-1} \cup P_i(I_{i-1})$ pour $i \in [1,n]$

sémantique bien fondée

comment donner un sens aux programmes non stratifiables ?
construire un modèle pour lequel

- certains faits seront vrais
- certains faits seront faux
- certains faits seront inconnus

on utilise non plus 2, mais 3 valeurs de vérité...

exemple

soit le programme P
$w(x) \leftarrow m(x,y), \neg w(y)$

et l'instance
$\{m(a,b), m(b,a), m(b,c), m(c,d)\}$

le modèle bien fondé de P est

- $W^+ = \{m(a,b), m(b,a), m(b,c), m(c,d), w(c)\}$
- $W^- = \{w(d)\}$
- $W^? = \{w(a), w(b)\}$