Datalog avec négation (notes de cours) - MLM

Rappels

Une **instance de base de données (BD)** relationnelle est un ensemble de tables (donnant, pour chaque relation du schéma de la BD, une liste des tuples satisfaisant cette relation). On supposera ici qu'une BD n'a pas de valeurs manquantes (NULL). On peut donc voir une BD comme une **base de faits** instanciée (c'est-à-dire sans variable, un fait étant alors un atome instancié), avec l'interprétation du **monde clos** (tout fait non présent dans la BD est considéré comme faux).

Exemple (plan de métro) : Table de la relation "Direct" ("sur la ligne L, on va directement de S1 à S2")

Ligne	*Station 1*	*Station 2*
4	St Germain	Odéon
4	Odéon	Saint Germain
10	Odéon	Cluny
…	…	…

BF = { Direct (4, SG, O), Direct (4, O, SG), Direct(10, O, C) ∧ … }

Datalog positif

Une **requête Datalog** (ou **programme Datalog**) est une suite de règles permettant de définir de nouvelles relations (qui ne seront pas stockées dans les tables de la BD). Les prédicats apparaissant dans le programme Datalog sont séparés en deux sortes :

- *L'ensemble des prédicats intensionnels* (**IDB**): ceux qui apparaissent en tête de règle ;
- *L'ensemble des prédicats extensionnels* (**EDB**): ceux qui font partie du schéma de la BD.

On considère aussi un prédicat IDB spécial (noté ici **Ans**) qui définit la notion de réponse à la requête Datalog et qui n'apparaît qu'en tête de certaines règles. Etant donnée une BD D et un programme Datalog ∏, l'ensemble des réponses à ∏ sur D est l'ensemble des atomes de prédicat Ans dans la saturation de D par ∏.

Sans perte de généralité[1], on supposera que les ensembles de prédicats intensionnels et extensionnels sont disjoints.

Notation Datalog : une règle Datalog est habituellement donnée sous la forme "tête ← corps", où la tête est réduite à un atome, le corps est une conjonction d'atomes, et toute variable de la tête apparaît aussi dans le corps (cf. règles "range-restricted"). Vous pouvez adopter la notation "corps —> tête" si vous préférez.

Exemples de requêtes :

"Trouver les stations atteignables directement à partir d'Odéon"

Ans(y) ← Direct(x, O, y) (c'est une requête conjonctive simple)

"Trouver les stations directement connectées à Odéon"

Ans(y) ← Direct (x, O, y)
Ans(y) ← Direct (x, y, O)
(ce n'est plus une requête conjonctive mais l'union de deux requêtes conjonctives)

[1] Montrer qu'on peut transformer tout programme Datalog ∏ en un programme Datalog ∏' satisfaisant la condition IDB ∩ EDB = ∅, tout en préservant les réponses pour toute BD.

"Trouver les stations atteignables à partir d'Odéon, directement ou indirectement"

Connecté (y, z) ← Direct (x, y, z)
Connecté (x, z) ← Connecté (x, y), Connecté (y, z)
Ans(x) ← Connecté (x, O)

(on utilise ici la récursivité permise par Datalog ; cette requête n'est pas exprimable sous la forme d'une union de requêtes conjonctives, ni plus généralement sous la forme d'une requête SQL).

Lien avec les règles « Datalog » ou « range-restricted » définissant une ontologie : en Datalog, les règles sont utilisées pour augmenter l'expressivité du langage de requêtes et font donc partie de la requête. Mais on peut partitionner une requête Datalog en deux ensembles de règles : celles définissant de nouveaux prédicats autres que Ans et celles avec tête Ans : on obtient d'une part une **ontologie** de règles « range-restricted » et d'autre part une **requête** qui est une union de requêtes conjonctives.

Considérons une base de connaissances composée d'une base de faits (ou base de données) et d'un ensemble de règles Datalog. Une telle base de connaissances a la propriété d'avoir un **unique plus petit modèle** [un plus petit modèle est un modèle qui n'en inclut strictement aucun autre]. On peut le voir comme l'intersection de tous les modèles de la base de connaissances. C'est donc un **modèle universel (et donc suffisant pour répondre à des requêtes conjonctives, ou union de requêtes conjonctives)**. Ce plus petit modèle est isomorphe à la base de faits saturée en chaînage avant par les règles (rappel : l'application des règles sur une base de faits jusqu'à obtention d'un point fixe est un processus qui se termine et qui produit une base de faits saturée unique, quel que soit l'ordre d'application des règles).

Datalog$^{\neg}$: Datalog avec négation (du monde clos)

Du fait de l'hypothèse du monde clos, la négation ne peut intervenir **que** dans le corps des règles (sinon, on produirait des faits "négatifs", ce qui n'a pas de sens avec cette hypothèse).

"Trouver les stations qui ne sont pas atteignables à partir d'Odéon, directement ou indirectement"

Connecté (y, z) ← Direct(x, y, z)

Connecté (x, z) ← Connecté (x, y), Connecté (y, z)
Ans(x) ← ¬ Connecté (x, O), Station(x) *// Rappel : "non" signifie ici "n'est pas présent"*
// on suppose ici qu'on a une relation Station, qui peut être un prédicat EDB ou IDB

De plus, on se limite à des **règles sûres** (*safe*) : toute variable de la règle apparaît dans au moins un littéral *positif* du corps de la règle. Par exemple, quel serait l'ensemble des valeurs possibles pour x dans la requête ci-dessous ?

Ans(x) ← ¬ Connecte(x, O)

Soit on considère le domaine « actif » de la BD interrogée (les valeurs, ou constantes, apparaissant dans les tables), mais dans ce cas une modification dans une table autre que Connecte *peut modifier le résultat de la requête (bof). Soit on considère le domaine du schéma de la BD, c'est-à-dire l'ensemble de toutes les valeurs autorisées dans des tables, et ce domaine étant généralement infini, on obtient un ensemble de réponses infini.*

Le corps d'une règle est défini par son **corps positif B^+** (l'ensemble de ses littéraux positifs) et son **corps négatif B^-** (l'ensemble des atomes qui apparaissent dans des littéraux négatifs). Une règle $H \leftarrow B^+, B^-$ est **applicable** sur une base de faits F s'il existe un homomorphisme h de B^+ dans F tel que $h(B^-) \cap F = \emptyset$ (autrement dit, aucun atome du corps négatif ne vient « **bloquer** » l'application de la règle selon h). Appliquer la règle consister à ajouter h(H) à F.

La logique classique est **monotone** : si une formule f est conséquence d'un ensemble de formules E, alors f reste conséquence si l'on ajoute à E de nouvelles formules. La négation du monde clos rend la déduction **non-**

monotone : par exemple, soit E = { R(a) ; Q(x) ⇐ R(x), ¬P(x) } ; de E on déduit Q(a), mais Q(a) n'est plus déductible de E ∪ {P(a)}). On ne peut donc pas traduire un programme Datalog¬ en logique classique.

Problème : L'**ordre des règles** influe sur ce que l'on peut produire (déduire) à partir d'une même base de faits : en particulier, on ne calcule pas les mêmes réponses au programme selon l'ordre dans lequel on considère les règles. Or, pour pouvoir donner une sémantique déclarative à un programme Datalog¬ (autrement dit, pour définir ce qui est conséquence du programme vu comme un *ensemble* de formules logiques) il faut que l'on puisse s'affranchir de l'ordre des règles.

Exemple 1 :

(R1) C(x) ⇐ ¬B(x) , A(x) (R2) B(x) ⇐ A(x) (R3) Ans(x) ⇐ C(x)

BD = { A(a) }
Avec l'ordre R1 puis R2 : on déduit C(a) par R1, puis B(a) par R2, puis Ans(a) par R3
Avec l'ordre R2 puis R1 : on déduit B(a), puis plus rien.

Dans cet exemple, l'application de R1 avant R2 est intuitivement choquante : on suppose que B(a) est absent pour appliquer R1, mais par la suite R2 produit B(a), ce qui remet en cause l'application de R1. On dira qu'une dérivation[2] à partir d'une certaine base de faits est **satisfaisante** si toute règle appliquée par un homomorphisme h à une certaine étape **reste applicable par le même homomorphisme** h dans la suite de la dérivation (autrement dit, aucune application de règle n'utilise l'absence d'un atome qui sera produit par la suite).

D'autre part, même si on se restreint à des dérivations satisfaisantes, on peut avoir plusieurs résultats :

Exemple 2 :

(R1) P(x) ⇐ ¬Q(x) , A(x) (R2) Q(x) ⇐ ¬P(x) , A(x) (R3) Ans(x) ⇐ P(x)

BD = { A(a) }
Avec l'ordre R1 puis R2 : on déduit P(a) (par R1), puis Ans(a) (par R3)
Avec l'ordre R2 puis R1 : on déduit Q(a), puis plus rien

Dans cet exemple, les deux dérivations satisfaisantes produisent des bases saturées différentes. De façon générale, la saturation d'une base de faits par des règles Datalog¬ produit toujours un **modèle** de la base de connaissances, cependant le résultat n'est pas unique.

Nous allons maintenant voir des conditions qui permettent d'obtenir un **résultat unique** quelle que soit la BD considérée.

1. Programme **semi-positif** : on n'autorise la négation que sur les prédicats extensionnels (EDB). L'ordre des règles n'a alors aucune importance puisque par définition elles ne produisent aucun nouvel atome de prédicat EDB (rappel : IDB ∩ EDB = ∅). Toutes les dérivations sont satisfaisantes et donnent le même résultat.

On étend naturellement cette notion en celle de programme stratifié :

2. Programme **stratifié** : l'idée est de partionner l'ensemble des règles en des « paquets » totalement ordonnés (appelés « strates »), avec un ordre qui assure que lorsqu'on utilise un littéral négatif ¬p(...) dans un corps de règle, on a déjà produit tous les faits (littéraux positifs) portant sur ce prédicat p. Ensuite, on sature la base de faits strate par strate, dans l'ordre croissant des strates.

   A chaque prédicat intensionnel p on associe un entier positif $\alpha(p)$. Les règles sont ensuite rangées en strates suivant le numéro α associé au prédicat de leur tête. Chaque strate peut donc être vue comme « définissant »

[2] Une dérivation est une séquence d'application de règles (voir la définition formelle dans le cours sur les règles existentielles).

un certain nombre de prédicats intensionnels. On exécute les règles en marche avant par ordre croissant des strates : à l'étape i, on sature la base calculée à l'étape i-1 (pour i = 1, c'est la base de données de départ) avec la règles de la strate i.

La numérotation α vérifie les conditions suivantes :

1. Pour toute règle de la forme p'(...) ← … p(...)…, on α (p) ≤ α (p')
   (le ≤ permet la récursivité entre règles)
2. Pour toute règle de la forme p'(...) ← …¬p(...)…, on a α (p) < α (p')
   (si le prédicat p est utilisé sous la forme négative, il faut absolument qu'on ait produit tous les atomes ayant ce prédicat p dans une strate précédente).

Un programme stratifié est ainsi découpé en une séquence de **sous-programmes semi-positifs :** le sous-programme de l'étape i est semi-positif par rapport à la base de données calculée à l'étape i-1 ; en effet, les littéraux négatifs du sous-programme de l'étape i ne portent que sur des prédicats qui n'apparaissent pas en tête de règle, du fait de la définition de α.

Un programme est **stratifiable** si on peut le transformer en un programme stratifié, c'est-à-dire s'il existe une numérotation α de ses prédicats intensionnels vérifiant les conditions 1 et 2 ci-dessus. Tous les programmes Datalog¬ ne sont pas stratifiables (voir Exemple 2).

Outil utile : le **graphe (de précédence) des prédicats intensionnels**. Les sommets de ce graphe sont les prédicats intensionnels, et on a un arc de p à p' si p apparaît dans le corps d'une règle de tête p' ; cet arc est étiqueté + (respectivement -) si p apparaît dans un littéral positif (respectivement négatif). On peut montrer qu'un programme est stratifiable si et seulement si son graphe des prédicats n'admet aucun circuit avec un arc négatif. Une stratification s'obtient alors en associant à chaque composante fortement connexe une strate de façon compatible avec l'ordre partiel sur les c.f.c. : étant données deux c.f.c. Ci et Cj, s'il y a un arc négatif (respectivement positif) d'un prédicat de Ci vers un prédicat de Cj, alors strate(Ci) < strate(Cj) (respectivement strate(Ci) ≤ strate(Cj)).

Si un programme est **stratifiable**, alors **toutes les dérivations qui suivent une stratification de ce programme** (n'importe laquelle) produisent la **même saturation** à partir d'une BD donnée. Remarquons que ces dérivations sont satisfaisantes, par définition des strates. D'autre part, toutes les dérivations satisfaisantes complètes produisent le même résultat (même lorsqu'elles ne suivent pas les strates).

La saturation strate par strate d'une BD donnée avec un ensemble de règles ∏ stratifiable définit donc un unique modèle de la base de connaissances (BD,∏). Si ∏ est vu comme une requête, l'ensemble de ses réponses sur BD est l'ensemble des *(a1…ak)* tels que *Ans(a1…ak*) appartient à la saturation de BD par ∏, autrement dit l'ensemble des *Ans(a1…ak*) vrais dans le modèle associé.

A suivre ...

Il y a eu un certain nombre de propositions dotant de sémantique *tout* programme Datalog¬ et pas seulement les stratifiables. Dans la suite de ce cours, nous allons voir la notion de *« modèle stable »* qui a donné lieu au langage de programmation logique ASP (*Answer Set Programming*) : un programme (associé à une instance de BD, qui est vue comme un ensemble de règles à corps vide dans le cadre d'ASP) peut avoir un, aucun, ou plusieurs modèles stables, et chaque modèle stable correspond à une « solution » possible.

On a la propriété suivante : si un programme Datalog¬ est **stratifiable**, alors quelle que soit la base de connaissances K composée d'une base de faits (ou instance de BD) et des règles de ce programme, K a un **unique modèle stable** qui est celui que l'on obtient avec la **saturation strate par strate**. La sémantique des modèles stables peut donc être vue comme une généralisation de la sémantique des programmes Datalog¬ stratifiables.