Devoir Datalog

Jef Wijsen

2 avril 2020

1 Instructions

1.1 Poser une question sur le Forum général

D'abord, chaque étudiant est demandé de *soumettre (au moins une) question* sur le Forum général du cours, en utilisant la plateforme Moodle. Cette question sera soumise avant le vendredi, 1 mai 2020, et portera sur la matière du cours (XML, XSLT, XQuery, datalog, requêtes conjonctives). L'objectif est de lancer une discussion sur le contenu du cours. Voici quelques idées.

- Est-ce que datalog stratifié avec un ordre $\leq$ est plus puissant que datalog stratifié ? Peut-on simuler $\leq$ en datalog stratifié, de même manière que l'on puisse simuler $\neq$ (cf. la section 5 de la note de cours) ?
- Existe-t-il une méthode "facile" pour traduire une requête SPJRUD en datalog ?
- Pourquoi datalog ne contient pas de fonctions arithmétiques (addition, multiplication...) ?
- L'équation (1) dans la note de cours ne montre pas la possibilité d'utiliser $\neq$ ou $=$. Est-ce un oublie ?
- Dans la section 3.1 de la note, on parle de la *Closed World Assumption (CWA)*. Existe-t-il une alternative pour cette hypothèse ?
- Quel est l'intérêt du *Program Dependency Graph (PDG)* ?

1.2 Faire et rendre le devoir décrit ci-après

Les questions 1 à 4 doivent être résolues en utilisant DLV. Le système DLV est décrit dans la note de cours disponible à `http://informatique.umons.ac.be/ssi/teaching/bdIImons/primerDatalog.pdf`

Avant le vendredi, 1 mai 2020, chaque étudiant soumet sur la plateforme Moodle une archive `.zip`. Pour chaque étudiant, le nom de l'archive est composé de son nom de famille suivi de la lettre initiale de son prénom, par exemple, `TrumpD.zip` pour un étudiant qui s'appelle Donald Trump. Pour chaque question j, $j \in \{1, \ldots, 4\}$, l'archive contiendra un fichier `pgm`j`.txt` avec le programme datalog demandé et un fichier `db`j`.txt` avec une base de données qui a été utilisée pour tester le programme. Il est possible d'exécuter un tel programme avec la commande (prenons $j = 1$) :

```
dlv db1.txt pgm1.txt
```

Ces testes devraient être plus élaborés que les bases de données qui figurent à titre d'exemple ci-après. Ajoutez des commentaires dans vos programmes datalog expliquant leur fonctionnement !

Finalement, l'archive contiendra un fichier `q5.txt` avec votre réponse à la dernière question.

2 Questions

Question 1. Soit *Vins*/3 un prédicat EDB tel que *Vins*(v, y, q) signifie que le vin v était de qualité q dans l'année y. Soit *Abus*/3 un prédicat EDB tel que *Abus*(p, v, y) signifie que la personne p a déjà bu le vin v du millésime y. Cette base de données sert aussi comme exemple dans le cours de *Bases de Données I*. La base de données satisfera toujours les contraintes suivantes :

$$\forall v \forall y \forall q_1 \forall q_2 \left((\mathit{Vins}(v, y, q_1) \wedge \mathit{Vins}(v, y, q_2)) \rightarrow q_1 = q_2 \right);$$
$$\forall p \forall v \forall y \left(\mathit{Abus}(p, v, y) \rightarrow \exists q \left(\mathit{Vins}(v, y, q) \right) \right).$$

Vins

1	2	3
Volnay	1983	A
Volnay	1979	B
Chablis	1983	A
Julienas	1986	C

Abus

1	2	3
Jean	Volnay	1983
Jean	Volnay	1979
Pierre	Volnay	1979
Pierre	Julienas	1986
Anne	Volnay	1983
Anne	Chablis	1983

Noter que des crus d'autres qualités ($\texttt{D}, \texttt{E}, \texttt{F}, \ldots$) sont possibles.

Écrivez un programme en datalog stratifié avec $\neq$ pour le prédicat IDB *FineBouche*/1 tel que *FineBouche*(`p`) est vrai si la personne `p` n'a bu que des crus de qualité A. Pour la base de données ci-dessus, *FineBouche*(`Anne`) est vrai, *FineBouche*(`Jean`) est faux, et *FineBouche*(`Pierre`) est faux.

DLV utilise `!=` pour $\neq$.

Question 2. Soit *Prérequis*/3 un prédicat EDB, tel que *Prérequis*(`c1`, `c2`, `univ`) est vrai si à l'université `univ`, le cours `c1` est un prérequis pour le cours `c2`. Écrivez un programme en datalog stratifié pour le prédicat IDB *Acyclique*/1 tel que *Acyclique*(`univ`) est vrai si à l'université `univ`, le graphe des prérequis est acyclique, comme il devrait l'être. Pour la table ci-dessous, *Acyclique*(`ucla`) est vrai, mais *Acyclique*(`mit`) est faux.

Prérequis

1	2	3
logique	algèbre	mit
logique	analyse	mit
algèbre	topologie	mit
analyse	topologie	mit
topologie	logique	mit
algèbre	topologie	ucla
analyse	topologie	ucla
topologie	database	ucla

Question 3. Soient *Rouge Cy* et *Bleu Cy* deux compagnies d'autobus qui utilisent des prédicats *Rouge*/2 et *Bleu*/2 pour stocker leurs connexions directes. Écrivez un programme en datalog stratifié avec $\neq$ pour le prédicat IDB *Critique*/2 tel que *Critique*(`v1`, `v2`) est vrai si les conditions suivantes sont toutes les deux satisfaites :

1. Une des deux compagnies, mais pas les deux, assure une connexion de `v1` à `v2`. Donc, soit *Rouge*(`v1`, `v2`) est vrai et *Bleu*(`v1`, `v2`) est faux, soit *Rouge*(`v1`, `v2`) est faux et *Bleu*(`v1`, `v2`) est vrai ; et
2. si la connexion de `v1` à `v2` est supprimée, il ne sera plus possible d'atteindre `v2` à partir de `v1`.

Par exemple, pour la base de données ci-dessous, *Critique*(`mons`, `dour`) est vrai. *Critique*(`huy`, `ath`) n'est pas vrai : si la connexion *Rouge*(`huy`, `ath`) est supprimée, il est encore possible d'aller de `huy` à `ath` en utilisant, par exemple, les connexions *Bleu*(`huy`, `dour`) et *Rouge*(`dour`, `ath`).

Rouge

1	2
ath	mons
mons	dour
dour	ath
dour	huy
huy	ath

Bleu

1	2
mons	ath
ath	mons
huy	dour
dour	huy

Question 4. Écrivez un programme en datalog stratifié avec $\neq$ pour le prédicat IDB *PassePar*/3 tel que *PassePar*(`v1`, `v2`, `v3`) est vrai si les conditions suivantes sont toutes les trois satisfaites :

1. `v1` $\neq$ `v3` et `v2` $\neq$ `v3`.
2. Il existe un voyage en autobus qui a `v1` comme lieu de départ et `v2` comme lieu d'arrivée. Il est possible que `v1` = `v2`.
3. Il n'est pas possible de voyager de `v1` à `v2` sans passer par `v3`.

Pour la base de données ci-dessus *PassePar*(`huy`, `mons`, `ath`) est vrai. *PassePar*(`mons`, `mons`, `ath`) est aussi vrai, car aucun voyage qui part de Mons ne peut retourner à Mons sans passer par Ath. Par contre, le fait *PassePar*(`dour`, `dour`, `huy`) n'est pas vrai, car le tour suivant ne passe pas par `huy` : *Rouge*(`dour`, `ath`), *Bleu*(`ath`, `mons`), *Rouge*(`mons`, `dour`).

Aide : Si vous rencontrez des difficultés, essayez d'abord d'écrire ce programme pour une valeur fixée de `v3`. Par exemple, écrivez un programme pour *PassePar*(`v1`, `v2`, `Mons`), qui est vrai s'il existe un voyage de `v1` à `v2` et tout voyage de `v1` à `v2` passe par `Mons`.

Question 5. Soient $R/2$ et $S/2$ deux prédicats EDB. Est-ce que le programme suivant est stratifié ? Si oui, donnez une stratification. Si non, détaillez pourquoi le programme n'est pas stratifié.

```
RlvantR(X,Y) :- R(X,Y), S(Y,X).
RlvantS(Y,X) :- R(X,Y), S(Y,X).
GarbageS(Y) :- S(Y,X), not RlvantR(X,Y).
GarbageR(X) :- R(X,Y), not RlvantS(Y,X).
%---
GarbageR(X) :- R(X,Y), S(Y,X), GarbageS(Y).
GarbageS(Y) :- R(X,Y), S(Y,X), GarbageR(X).
%---
Eq(X,Y,X,Y) :- R(X,Y), S(Y,X).
%---
E(X1,Y1,X2,Y2) :- R(X1,Y1), S(Y1,X1), R(X2,Y2), S(Y2,X2), X1=X2, not Eq(X1,Y1,X2,Y2).
E(X1,Y1,X2,Y2) :- R(X1,Y1), S(Y1,X1), R(X2,Y2), S(Y2,X2), Y1=Y2, not Eq(X1,Y1,X2,Y2).
%--- arXiv contains a problem
UCon(X1,Y1,X1,Y1,X3,Y3) :- R(X1,Y1), S(Y1,X1), R(X3,Y3), S(Y3,X3),
                           not Eq(X1,Y1,X3,Y3), not E(X1,Y1,X3,Y3).
%--- Two symmetric rules.
UCon(X1,Y1,X2,Y2,X3,Y3) :- UCon(X1,Y1,U,V,X3,Y3), E(U,V,X2,Y2),
                           not Eq(U,V,X3,Y3), not E(U,V,X3,Y3),
                           not Eq(X2,Y2,X3,Y3), not E(X2,Y2,X3,Y3).
UCon(X1,Y1,U,V,X3,Y3) :-   UCon(X1,Y1,X2,Y2,X3,Y3), E(U,V,X2,Y2),
                           not Eq(U,V,X3,Y3), not E(U,V,X3,Y3),
                           not Eq(X2,Y2,X3,Y3), not E(X2,Y2,X3,Y3).
%---
InLongCycle(X1,Y1) :- E(X0,Y0,X1,Y1), E(X1,Y1,X2,Y2), E(X2,Y2,X3,Y3), E(X3,Y3,X4,Y4),
                      UCon(X0,Y0,X4,Y4,X2,Y2),
                      not Eq(X1,Y1,X2,Y2), not Eq(X1,Y1,X3,Y3), not Eq(X2,Y2,X3,Y3),
                      not E(X1,Y1,X3,Y3).
GarbageR(X) :- InLongCycle(X,Y).
GarbageS(Y) :- InLongCycle(X,Y).
%---
KeepR(X,Y) :- R(X,Y), not GarbageR(X).
KeepS(Y,X) :- S(Y,X), not GarbageS(Y).
%---
Link(Y1,Y2) :- KeepR(X1,Y1), KeepS(Y1,X1), KeepR(X2,Y2), KeepS(Y2,X2), X1=X2.
Link(Y1,Y2) :- KeepR(X1,Y1), KeepS(Y1,X1), KeepR(X2,Y2), KeepS(Y2,X2), Y1=Y2.
Trans(Y1,Y2) :- Link(Y1,Y2).
Trans(Y1,Y2) :- Trans(Y1,Y3), Link(Y3,Y2).
Trans(Y1,Y3) :- Trans(Y1,Y2), Link(Y3,Y2).
```