BDD - Cours 9

Introduction à Datalog

Celine Kuttler

3 novembre 2015

Histoire

- 1977s : invention de Prolog, dans le contexte des bases de données. Idée : ajouter du calcul récursif aux requêtes relationnelles.
- 1980s : programmation logique populaire pour l'intelligence artificielle. présence industrielle forte, mais pas encore de killer app pour les requêtes récursives.
- 1990s : niches...
- depuis environ 2007 : renaissance de Datalog pour le web.

Datalog est une machine pour construire des nouveaux faits.

Hypothèse du monde clos

Un fait est considéré comme faux s'il n'est ni inclus dans la base de données des faits, ni démontrable en temps fini. Il n'y a pas de monde extérieur qui pourrait contenir des éléments inconnus au programme.
Cette hypothèse motive des contraintes syntaxiques (vues en fin de ce cours, si le temps le permet).

Contenu

Introduction a Datalog

- Faits et règles
- Différentes catégories de règles, et leur significations
- Requêtes récursives : chemins dans un graphe

Requêtes relationnelles en Datalog

- Opérations de l'algèbre relationnelle
- Aggregation et groupage en DES

Chemin le plus court : Datalog vs SQL

Semantique

- Sémantique des points fixes
- Exemple : graphe
- Memoization
- Sûreté
- Differences entre Datalog et Prolog

Appendix

Exemple 1 : des faits et des faits

```
fete.
femme(mia).
femme(jody).
femme(yolanda).
happy(yolanda).
joueAirGuitar(jody).
```

Requêtes :

- tuple trouvé ou non.
- requête avec variables : présence de femmes ? noms des femmes ?
- requête avec conjonction : existence d'une femme qui joue de l'AirGuitar.

Résumé de la syntaxe (1)

- constantes
  - nombres (entiers et réels)
  - séquences de caractères alphanumériques, _ inclus, qui commencent avec une minuscule
- prédicats $p(a_1, a_2, \ldots, a_n)$. Le prédicat p prend n arguments, qui sont des variables ou constantes. Le nom du prédicat doit commencer avec une minuscule.
- variables
  - X, Y (séquences qui commencent avec une majuscule,
  - _ (variable anonyme, tiret bas)

Comment construire des nouveaux faits ?

Une base de connaissances contient :

- des faits et
- des règles.

Modus Ponens

Du fait A, en combinaison avec la règle $A \Rightarrow B$, on déduit B.

Exemple 1 : la fête

- Fait : C'est la fête.
  F
- Règle : Quand c'est la fête, il y a de la musique.
  $F \Rightarrow M$
- Conclusion : Il y a de la musique !
  M

Une première règle en Datalog

```
fete.
musique :- fete.
```

Comment lire une règle Datalog ?

En notation de logique propositionnelle, la règle Datalog

$$B :- A.$$

se lit comme implication dans l'autre direction,

$$A \Rightarrow B$$

Règles avec plusieurs conditions

En notation de logique propositionnelle, la règle Datalog

$$Z :- A1, A2, ..., An.$$

se lit comme implication dans l'autre direction,

$$A1 \text{ and } A2 \text{ and } ... \text{ and } An \Longrightarrow Z$$

- dans la tête d'une règle, toujours un seul fait
- le corps de la règle est une conjonction de n faits
- si toutes ses conditions sont satisfaites, on peut déduire le nouveau fait de la tête, Z

Exemple 2 : règles et leurs interprétations

```
fete.
happy(yolanda).
ecoute2LaMusique(mia).
ecoute2LaMusique(yolanda) :- happy(yolanda).
joueAirGuitar(mia) :- ecoute2LaMusique(mia).
joueAirGuitar(yolanda) :-
                          ecoute2LaMusique(yolanda).
```

Interprétation

- Quand Yoland est contente, elle écoute de la musique.
- Quand Mia écoute de la musique, elle joue de l'AirGuitar.
- Quand Yolanda écoute de la musique, elle joue de l'AirGuitar.

Exemple 2 : Requêtes, concernant des faits déduits

```
fete.femme(mia).  ecoute2LaMusique(mia).
femme(yolanda).happy(yolanda).

ecoute2LaMusique(yolanda) :- happy(yolanda).
joueAirGuitar(mia) :- ecoute2LaMusique(mia).
joueAirGuitar(yolanda) :-
                          ecoute2LaMusique(yolanda).
```

Requêtes

- Est-ce que Mia joue de l'AirGuitar ?
- Est-ce que Yolanda joue de l'AirGuitar ?
- Qui joue de l'AirGuitar ?

Exemple 3 : Conjonction logique (et)

```
happy(vincent).
ecoute2LaMusique(paul).
joueAirGuitar(vincent) :-
        ecoute2LaMusique(vincent), happy(vincent).
```

Syntaxe

La virgule exprime la conjonction en Datalog.

Interprétation

Vincent joue de l'AirGuitar, quand il est content et qu'il écoute de la musique.

Requêtes

- Est-ce que Vincent joue de l'AirGuitar ?

Exo : trouver une musicienne

Comment demander s'il existe une femme qui joue de l'AirGuitar ?

[autoview, mot-clé réservé (answer)]

Exemple 4 : Disjonction logique

Paul joue de l'AirGuitar, quand il est content, ou qu'il écoute de la musique.

Syntaxe

Pour exprimer un OU logique en Datalog, on écrit deux règles.

```
ecoute2LaMusique(paul).

joueAirGuitar(paul) :- ecoute2LaMusique(paul).
joueAirGuitar(paul) :- happy(paul).
```

Requête

- Est-ce que Paul joue de l'AirGuitar ?

Résumé de la syntaxe (2)

Comment exprimer les opérateurs logiques en Datalog :

- conjonction : virgule
- disjonction : écrire deux règles
- implication $B \ :- \ A_1, \ldots, A_n \ .$
  - le corps est une conjonction de n faits
  - dans la tête, un seul fait
  - si toutes ses conditions sont satisfaites, on peut déduire le nouveau fait B

Catégories de règles

Nous pouvons distinguer 3 catégories de règles, de plus en plus expressives :

- règles avec faits simples, c.a.d. constantes (quand c'est la fête, il y a de la musique)
- règles avec prédicats + constantes
- règles avec prédicats + variables

Règles avec variables

Le millionnaire

- Tous les millionnaires ont un coffre-fort.
  $\forall x : \mathit{millionnaire}(x) \Rightarrow \mathit{avoir\_coffre\_fort}(x)$
- en Datalog :
  $\mathit{avoir\_coffre\_fort}(X) : -\mathit{millionaire}(X).$
- Fait : Balthazar Picsou est un millionnaire.
  $\mathit{millionnaire}(bp)$
- Conclusion : Balthazar Picsou a un coffre-fort.
  $\mathit{avoir\_coffre\_fort}(X)$

Base : bars

Schémas

sert(bar, biere ,prix)
frequente(personne, bar)
aime(personne,biere)

```
sert(mcevans,lachouffe,3.5).
sert(mcevans,leffe,2.5).
sert(omnia,lachouffe,4.5) .
sert(taverneflamande,lachti,1.9).

frequente(timoleon,mcevans).

aime(timoleon,lachouffe).
```

Exemple de règle avec variables

```
content(X) :-   aime(X, Beer) , frequente(X, Bar),
                sert(Bar, Beer, _).
```

En français

Si quelqu'un aime une certaine bière, et fréquente un bar qui vend cette bière, alors il est content.

En logique

$\forall X, Bar, Beer, Prix$:
$aime(X, Beer)$ and $frequente(X, Bar)$ and $sert(Bar, Beer, Prix)$
$\rightarrow content(X)$

Questions auxquelles Datalog sait répondre

Exemples

- Timoléon est-il content ?
- Qui est content ?

```
sert(mcevans,lachouffe,3.5).
sert(mcevans,leffe,2.5).
sert(omnia,lachouffe,4.5)
sert(taverneflamande,lachti,1.9)

frequente(timoleon,mcevans).

aime(timoleon,lachouffe).

content(X) :- aime(X,Beer), frequente(X,Bar),
    sert(Bar,Beer,_).
```

Exemple de règle avec comparaison

Syntaxe

comparaisons *A op B*, ou *A* et *B* sont des constantes ou variables, et *op* un opérateur de comparaison

```
bonmarche(B)  :-  sert(B, _ ,P), P =< 2.0 .
```

Interprétation de la règle

Un bar est bon marché, s'il sert une bière à moins de deux euros.

Astuce

L'occurrence de la variable P dans le prédicat *sert* permet une liaison de cette variable. Uniquement après cette liaison, la variable peut être comparée.

Signification des règles Datalog

Datalog est une machine pour construire des nouveaux faits.

Première approximation de la sémantique pour règles avec variables, non-récursives.

- prends les valeurs de variables qui rendent le corps de la règle vrai (il faut rendre vrai chacun des sous-buts)
- considère les valeurs que peuvent prendre les variables de la tête
- ajoute le tuple crée dans l'étape précédente, à la relation en tête de règle

Extension vs intension

Prédicats

- extension : les prédicats dont les relations sont enregistrées dans la base, comme faits.
- intention : des prédicats définis par des règles (c.a.d. en tête)

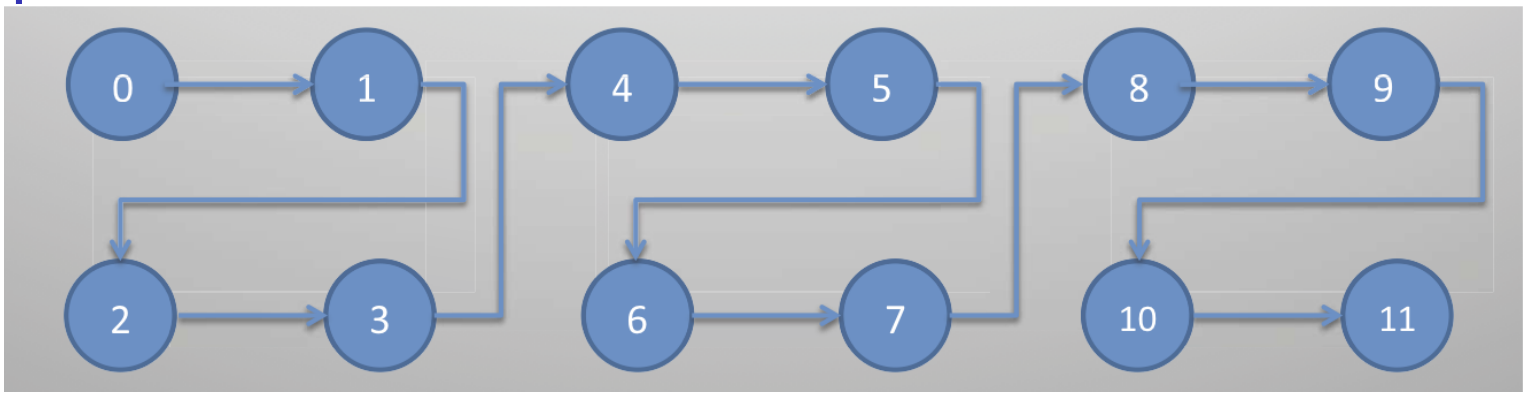
Terminologie

- EDB (extensional database) : collection de relations extensionelles
- IDB (intensional database) : collection de relations intensionnelles

Exemple

Quels prédicats du schéma *bar* sont extensionnels, quels intensionnels ?

Graphe



Le prédicat $e/2$ exprime un lien direct entre deux sommets (e pour anglais : edge) :

```
e(0,1). e(1,2). e(2,3).
e(3,4). e(4,5). e(5,6).
e(6,7). e(7,8). e(8,9).
e(9,10). e(10,11).
```

But : tester l'existence d'un chemin entre deux noeuds

- Requête pour tester l'existence des chemin de longueur fixe, p.ex. 2 et 3, entre deux noeuds.
- Peut-on poser les requêtes correspondantes en SQL ?

Chemins dans un graphe

- Comment tester si deux sommets (edge) sont connectés, quelque soit la longueur du chemin ?
- Définir un prédicat $p/2$, qui exprime un chemin (path) entre deux sommets passant par un nombre arbitraire de liens.

```
p(X,Y) :- e(X,Y).

p(X,Z) :- e(X,Y), p(Y,Z).
```

Contenu

Introduction a Datalog

- Faits et règles
- Différentes catégories de règles, et leur significations
- Requêtes récursives : chemins dans un graphe

Requêtes relationnelles en Datalog

- Opérations de l'algèbre relationnelle
- Aggregation et groupage en DES

Chemin le plus court : Datalog vs SQL

Semantique

- Sémantique des points fixes
- Exemple : graphe
- Memoization
- Sûreté
- Differences entre Datalog et Prolog

Appendix

Datalog et l'algèbre relationnelle

- les mêmes relations qu'on peut définir en algèbre relationnelle, ou SQL, peuvent être définies en Datalog
- Nous supposons un nom de relation (symbole de prédicat) r en Datalog, pour chaque tableau R
- Nous montrons comment exprimer, en Datalog, des requêtes sur une base de données relationnelle, du type SELECT-FROM-WHERE, ou des requêtes en algèbre relationnelle

Exemple : la boutique de BDD

- articles(*aid* : *int*, *anom* : *string*, *acoul* : *string*)
- fournisseurs(*fid* : *int*, *fnom* : *string*, *fad* : *string*)
- catalogue(*fid* : *int*, *aid* : *int*, *prix* : *real*)
- déclaration de types (create table) + ajoutà la EDB (inserts) :

```
:-type(articles(aid:int,anom:string,acoul:string)).
:-type(fournisseurs(fid:int,fnom:string,fad:string)).
:-type(catalogue(fid:int,aid:int,prix:real)).

articles(1,'Left Handed Toaster Cover','rouge').
articles(2,'Smoke Shifter End','noir').
...
fournisseurs(1,'kiventout','59 rue du Chti,
                F-75001 Paris').
fournisseurs(2,'Big Red Tool and Die','4 My Way,
                Bermuda Shorts, OR 90305, USA').
...
catalogue(1,1,36.10).
catalogue(1,2,42.30).
```

Projection $\pi_{Acoul}(Articles)$

Afficher toutes les couleurs d'articles

```
couleur(C) :- articles(_,_,C).
```

Requête et résultat :

```
DES> couleur(X)
{
   couleur(argente),   couleur(cyan),
   couleur(magenta),   couleur(noir),
   couleur(opaque),   couleur(rouge),
   couleur(superjaune),   couleur(vert)
}
Info: 8 tuples computed.
```

```
CREATE VIEW couleur AS
SELECT acoul FROM articles
```

Selection σ

Afficher tous les articles verts. Afficher tous les articles rouges.

```
art_vert(Anom)
  :- articles(Aid,Anom,'vert').
art_rouge(Anom)
  :- articles(Aid,Anom,Acoul),Acoul='rouge'.

art_vert(X).
art_rouge(X).
```

```
CREATE VIEW art_rouge(anom) as
 SELECT anom FROM articles WHERE acoul='rouge';

CREATE VIEW art_vert(anom) as
 SELECT anom FROM articles WHERE acoul='vert';

select anom from art_rouge;
select anom from art_vert;
```

Jointure

Qui vend quel article a quel prix ?

```
tout(Anom, Fid, Prix)
  :- articles(Aid, Anom, _),
     catalogue(Fid, Aid, Prix).
```

```
CREATE VIEW tout
SELECT anom, fid, prix
FROM articles join catalogue using aid;
```

Produit cartésien

toutes les combinaisons de noms de fournisseurs et noms d'articles.

```
cart(Anom,Fnom) :-
    articles(_,Anom,_), fournisseurs(_,Fnom,_).
```

Requête et résultat :

```
DES> cart(X,Y)
{   cart('7 Segment Display','Alien Aircaft Inc.'),
    cart('7 Segment Display','Autolux'),
    ...
    cart('Smoke Shifter End','Vendrien'),
    cart('Smoke Shifter End','kiventout')}
Info: 54 tuples computed.
```

```
CREATE VIEW cart AS
SELECT anom, fnom FROM articles, fournisseurs
```

Intersection

articles existant en rouge et en vert

```
rouge_et_vert(X) :- art_rouge(X), art_vert(X).
```

```
create view rouge_et_vert as
(
art_rouge intersect art_vert
)
```

Différence

les articles existant en rouge, mais pas en vert

```
rouge_pas_vert(X) :-
    art_rouge(X), not(art_vert(X)).
```

```
CREATE VIEW rouge_pas_vert AS
(art_rouge MINUS art_vert)
```

Union

les articles rouges ou verts

```
rouge_ou_vert(X) :-  art_rouge(X).
rouge_ou_vert(X) :-  art_vert(X).
```

```
CREATE VIEW rouge_ou_vert AS
    (art_rouge UNION art_vert)
```

Quantification existentielle

Articles offerts par au moins un fournisseur

```
vendu(Anom) :-
    catalogue(_,Aid,_),articles(Aid,Anom,_).
```

```
SELECT anom FROM articles a WHERE exists
  (select * from catalogue c where c.aid=a.aid)
```

Quantification universelle

Il faut passer par la négation, et donc des strates d'ordre supérieur. Nous n'avons pas le temps de voir la stratification, une contrainte syntaxique, imposee en Datalog pour utiliser la negation.

Fonctionnalités supplémentaires de DES

Les opérations montrées jusqu'ici peuvent être faites avec n'importe quel DATALOG. Pour augmenter son attractivité, DES offre :

- fonctions d'agrégation : count, min,max,avg,sum
  - Versions avec 1,2 ou 3 arguments pour différents contextes.
- group by - having
- différentes jointures

Nous présentons ici count/2 et avg. Puis nous utiliserons min dans un autre exemple.

count/2

Combien d'articles ?

```
DES> count(articles(_,_,_),C)

Info: Processing:
  answer(C) :-
    count(articles(_,_,_),[],C).
{
  answer(13)
}
Info: 1 tuple computed.
```

Prix moyen du fournisseur X (avg/3)

```
prix_moyen_fid(X,R) :-
   avg(catalogue(X,_,P),P,R).
```

Requete : prix moyen du fournisseur 1

```
DES> prixmoyen(1,P)
```

SQL : prix moyen du fournisseur 1

```
SELECT avg(prix)
FROM catalogue
WHERE fid = 1
```

Contenu

Introduction a Datalog

- Faits et règles
- Différentes catégories de règles, et leur significations
- Requêtes récursives : chemins dans un graphe

Requêtes relationnelles en Datalog

- Opérations de l'algèbre relationnelle
- Aggregation et groupage en DES

Chemin le plus court : Datalog vs SQL

Semantique

- Sémantique des points fixes
- Exemple : graphe
- Memoization
- Sûreté
- Differences entre Datalog et Prolog

Appendix

Chemin le plus court en Datalog

```
path(X,Y,1) :- edge(X,Y).

path(X,Y,L) :-
   path(X,Z,L0),
   edge(Z,Y),
   count(edge(A,B),Max),
   L0<Max,
   L is L0+1.

% requete:
shortest_paths(X,Y,L) :-
    min(path(X,Y,Z),Z,L).
```

La condition $L0 < Max$ assure la terminaison (elle interdit de boucler infiniment dans un cycle du graphe).

Requetes recursives en SQL

- requetes recursives dans le standard SQL depuis la quatrieme revision SQL :99 (nom alternatif : SQL3)
- en Postgres depuis version 8.4 (2009)

```
WITH [RECURSIVE] with_query [, ...]
SELECT ...
```

syntaxe pour with_query :

```
query_name [ (column_name [,...]) ]
AS (SELECT ...)
```

- supposez la table *edge*(*origin*, *destination*) pour representer le graphe

```
CREATE OR REPLACE VIEW
shortest_paths(Origin,Destination,Length) AS
WITH RECURSIVE
path(Origin,Destination,Length) AS
    (SELECT e.*,1 FROM edge)
  UNION
    (SELECT
        path.Origin,edge.Destination,path.Length+1
     FROM path,edge
     WHERE path.Destination=edge.Origin
           and path.Length <
            (SELECT COUNT(*) FROM Edge)
     )
SELECT Origin,Destination,MIN(Length)
FROM path
GROUP BY Origin,Destination;

% requete en SQL
SELECT * FROM shortest_paths;
```

Contenu

Introduction a Datalog
- Faits et règles
- Différentes catégories de règles, et leur significations
- Requêtes récursives : chemins dans un graphe

Requêtes relationnelles en Datalog
- Opérations de l'algèbre relationnelle
- Aggregation et groupage en DES

Chemin le plus court : Datalog vs SQL

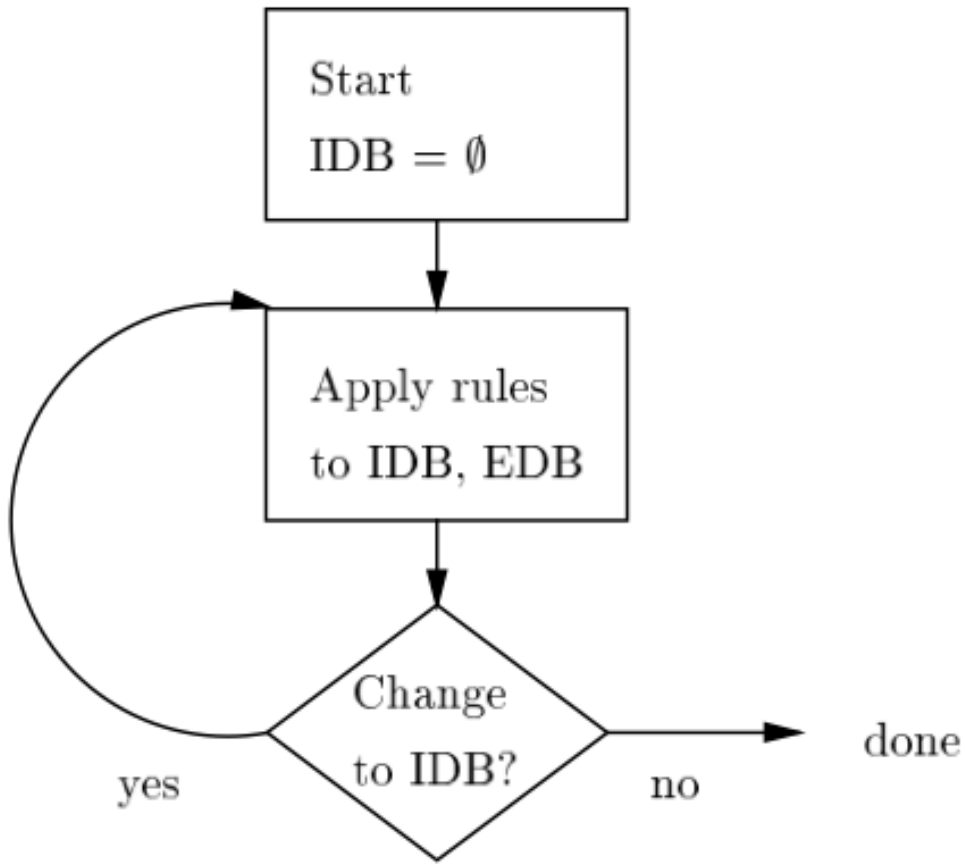
Semantique
- Sémantique des points fixes
- Exemple : graphe
- Memoization
- Sûreté
- Differences entre Datalog et Prolog

Appendix

Sémantique des points fixes

Idée de l'algorithme :

Iterative Fixed-Point Evaluates Recursive Rules



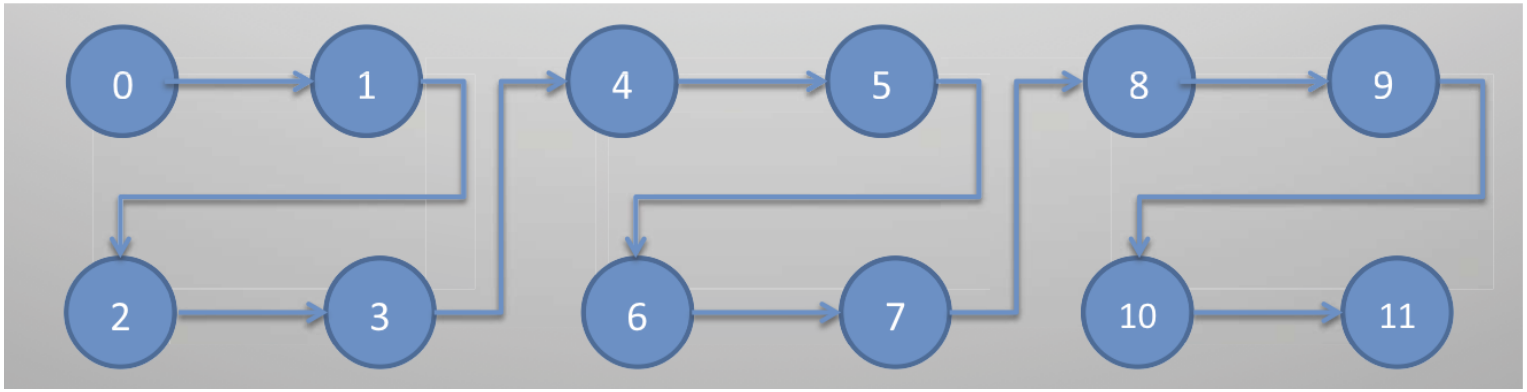
Ullmann

Sémantique des points fixes

Verbalisation de l'algorithme :

1. Suppose que tous les prédicats de la IDB sont vides.
2. Construction de relations IDB de plus en plus grandes :
   - Applique les règles et ajoute des tuples aux relations IDB.
   - Utilise les tuples ajoutés à la IDB dans l'étape précédente pour ajouter encore plus de tuples à la IDB.
3. Continue à appliquer les règles, jusqu'à ce que cela n'ajoute plus de nouveaux tuples. On a atteint un point fixe. Si les règles sont sûres, il n'y aura qu'un nombre fini de tuples satisfaisant les corps des règles, et donc, le point fixe sera atteint avec un nombre borné de répétitions.

Exemple pour la sémantique : graphe



Le prédicat $e/2$ exprime un lien direct entre deux sommets :

```
e(0,1). e(1,2). e(2,3). e(3,4). e(4,5). e(5,6).
e(6,7). e(7,8). e(8,9). e(9,10). e(10,11).
```

Le prédicat $p/2$ exprime un chemin (path) entre deux sommets passant par un nombre arbitraire de liens :

```
p(X,Y) :- e(X,Y).

p(X,Z) :- e(X,Y), p(Y,Z).
```

Exemple : que peut-on déduire du programme ?

```
p(X,Y) :- e(X,Y).

p(X,Z) :- e(X,Y),c(Y,Z).

ok :- p(0,11).
```

EDB :

```
e(10,11).
e(9,10).
e(8,9).
e(7,8).
e(6,7).
e(5,6).
e(4,5).
e(3,4).
e(2,3)
e(1,2).
e(0,1).
```

IDB :

But : montrer qu'il existe un chemin de 0 à 11.

- instantiations des règles, puis déduction de nouveaux tuples dans la IDB, utilisant des tuples existants.
- au tableau ...

Exemple : que peut-on déduire du programme ?

```
p(X,Y) :- e(X,Y).

p(X,Z) :- e(X,Y),p(Y,Z).

ok :- p(0,11).
```

EDB :

```
e(10,11).
e(9,10).
e(8,9).
e(7,8).
e(6,7).
e(5,6).
e(4,5).
e(3,4).
e(2,3)
e(1,2).
e(0,1).
```

IDB :

```
p(10,11).
p(9,11).
p(8,11).
p(7,11).
p(6,11).
p(5,11).
p(4,11).
p(3,11).
p(2,11).
p(1,11).
p(0,11).
ok.
```

- pas 1 : règle 1 avec $e(10, 11)$, ajout de $p(10, 11)$ à la IDB.
- puis : règle 2 avec le prochain lien, et le dernier tuple ajouté à la IDB. injection d'un tuple supplémentaire la IDB. **répète.**
- dernier pas : règle 3.

(pas encore saturée !)

Comment expliquer l'absence de boucle infinie ?

```
a(X) :- b(X).                    % test.dl

b(X) :- a(X).
```

- DES rend le résultat rapidement pour la requête $a(4)$
- pourtant, la recherche de preuve devrait continuer à l'infini (d'après ce que nous avons vu)

Memoization

Technique d'optimisation de code visant à réduire le temps d'exécution. Datalog mémorise les faits qu'il a déjà prouvés, ou tenté de prouver. Évite de répéter le même calcul deux fois. Cette technique est absente en Prolog.

Commande : *list_et*

Motivation pour la sûreté

Sources de problèmes

Mauvaise utilisation des prédicats prédéfinis (comparaisons, négation . . .), et des variables.

- Le résultat d'une requête doit être une relation finie
- Certains types de buts (sous-requetes) génèrent un nombre infinis de lignes, alors qu'il est impossible que la table d'une relation R soit de taille illimitée
- il faut éliminer maximum les warnings **unsafe**

Règles correctes, ou sûres

Une règle est sûre, quand

chaque variable, et notamment,

- dans la tête
- dans une négation
- dans une comparaison

apparaît également sous forme positive dans le corps de la règle, dans un prédicat défini dans le même programme.

Le résultat d'une requête est toujours de taille finie.

Exemples de violations :

```
p(X) :- q(Y).
celibataire(X) :- not(marie(X,Y)).
celibataire(X) :- personne(X), not(marie(X,Y)).
homme(X) :- not(femme(X)).
```

Mauvais exemple avec variable

```
c(X) :- d.
d.
```

Permet de prouver un nombre infini de faits.

Mauvais exemples avec comparaison et arithmétique

```
% insecure1.dl
p(X) :- X>10.
```

```
% insecure2.dl

 c(Y) :- c(X), Y+1=X.
```

Requêtes closes vs requêtes avec variables.

Differences entre Datalog et Prolog

Prolog est vu au S6 dans l'UE Logique.

- Tout ce qui peut etre ecrit en Datalog (a l'eception des constructions specifiques de DES pour le groupage et l'aggregation), peut etre ecrit en Datalog.
- Prolog permet d'ecrire des faits plus complexes que Datalog (distinction en symboles de fonction et predicats). Prolog permet du filtrage de motif sur les termes complexes.
- En Prolog, on dessine manuellement des arbres pour expliquer le comportement d'un programme. Ces arbres s'appellent des arbres de resolution.

Differences entre Datalog et Prolog

Prolog est vu au S6 dans l'UE Logique.

- Prolog, contrairement a Datalog, n'a pas de semantique formelle. Nous avons vu en survol une des trois semantiques de Datalog, aujourd'hui.
- En Prolog, il n'y a pas de stratification. En Datalog, la stratification rend l'utilisation de la negation sure. En Prolog, il faut connaitre utiliser des heuristiques pour la syntaxe pour eviter des bugs. Seulement certaines des variantes syntaxiques admises d'un programme se comportent correctement. Il n'y a pas de message d'erreurs permettant de dectecter des mauvais usages du langage (qui menent a la non-termination de programmes, dont la visualisation se fait par des branches infinies dans l'arbre de resolution).
- Pour comprendre la negation en Prolog, il faut s'imaginer de couper des branches dans des arbres de preuve.

Contenu

Introduction a Datalog
- Faits et règles
- Différentes catégories de règles, et leur significations
- Requêtes récursives : chemins dans un graphe

Requêtes relationnelles en Datalog
- Opérations de l'algèbre relationnelle
- Aggregation et groupage en DES

Chemin le plus court : Datalog vs SQL

Semantique
- Sémantique des points fixes
- Exemple : graphe
- Memoization
- Sûreté
- Differences entre Datalog et Prolog

Appendix

Appendix : Fonctionalites supplementaires en DES

Documentation DES

```
DES> /help count
* Aggregate Functions:
count/1
  Count but nulls wrt. its argument.
  Returns an integer
* Predicates:
count/3
  Aggregate returning the number of the tuples
  in a relation wrt. an argument, ignoring nulls
count/2
  Aggregate returning the number of the tuples
  in a relation (cf. SQL's COUNT(*))
```

Exemple : count/3

Paramètres :

1. requête
2. compter cette variable
3. associer résultat à cette variable

Combien d'articles rouges ?

```
combien_rouge(R):-
count(articles(Aid,_,'rouge'),Aid,R).
```

Requête et résultat :

```
DES> combien_rouge(R)
{   combien_rouge(5) }
Info: 1 tuple computed.
```

SQL :

```
SELECT count(*) as R
FROM articles
WHERE acoul='rouge'
```

Prédicat de groupage de DES : syntaxe

group_by/3 :

```
group_by(
   Relation_A ,              % FROM
   [Var_1 ,... , Var_n] ,    % attributs de groupage
   C                         % HAVING /projection
   )
```

Prix moyen par fournisseur (group by + avg/1)

```
prixmoyen(Fid,Pmo) :-
        group_by(catalogue(Fid,Aid,P),
        [Fid],
        Pmo=avg(P)).
```

```
DES> prixmoyen(F,P)
{ prixmoyen(1,19585.389166666668),
  prixmoyen(2,8.333333333333334),
  prixmoyen(3,6.75),
  prixmoyen(4,42.699999999999996),
  prixmoyen(5,234555.67)}
Info: 5 tuples computed.
```

```
SELECT fid,avg(prix)
FROM   catalogue
GROUP BY fid
```

Nombre d'articles par couleur (group by + count /1)

```
coul(C,R) :-
        group_by(articles(A,N,C),
        [C],
        R=count).
```

Requête et résultat :

```
DES> coul(C,R)
{   coul(argente,1),...   coul(vert,2)}
Info: 8 tuples computed.
```

Couleur avec plus de 2 articles (group by-count-TEST)

```
coul_2(C) :-
   group_by(
       articles(A,N,C),
       [C],
       (R=count,R>2)
    ).
```

Résultat :

```
DES> coul_2(X)
{   coul_2(rouge) }
Info: 1 tuple computed.
```

```
SELECT acoul
FROM articles
GROUP BY acoul
HAVING count(*)>2
```