Bases de données avancées
Master 1ère année
Université Paris Sud, Faculté des Sciences d'Orsay

Mme Nicole Bidoit-Tollu *

Contact : Nicole Bidoit
LRI UMR 8623 CNRS - Bat 490
Université de Paris Sud,
91405 Orsay Cedex, France
email : bidoit@lri.fr

Avertissement : Ce support de cours est dans une version non stable; il peut donc contenir des erreurs (typos, fautes d'orthographe, ...) et je vous remercie de me les signaler; il n'est pas complet; il contient des sections qui ne seront pas présentées telles qu'elles en cours ou qui ne seront pas prśentées du tout.

*LRI UMR 8623 CNRS- Bat 490, Université de Paris Sud, 91405 Orsay Cedex, France, bidoit@lri.fr

1 Introduction

Objectifs :

- acquérir de nouvelles connaissances
- découvrir de nouvelles techniques
- apprendre à poser les bonnes questions

Un petit historique de la théorie des BDs

• avant 1970 : BD = fichier d'enregistrements
COBOL/CODASYL
modèle réseau

• en 1970 : BD = structure du 1er ordre
modèle relationnel (Codd)
une avancée importante !!!!!!!!!!
20 ans pour l'adopter !!!!!!!!!!

succès : théorie + pratique + commercial

Un petit historique : les années 80 et 90

• Eléments fondamentaux du relationnel
théorie des dépendances fonctionnelles
gestion des transactions (concurrence)
• Etude de nouveaux modèles
modèle non normalisé (imbrication)
modèles à valeurs complexes, à objets
• Etude de nouveaux langages
Datalog, WHILE, Fixpoint ...
complexité, expressivité
• Etude de nouvelles applications
information incomplète
bases de données distribuées
bases de données géographiques

• Point de vue pratique
  un **seul** language : SQL
  une application : OLTP
  une architecture : client – serveur

Aujourd'hui : l'age du Web

• les données du Web : que sont-elles ?
  données semi-structurées, XML
  XML-schéma, DTD
• les langages de requêtes : que sont-ils ?
  X-path, UNQL, XML-query ...
• les architectures : 3-tiers

Le cours de Bases de Données Avancées :

CONCEPTS FONDAMENTAUX : MODELES et LANGAGES

• Du modèle relationnel au modèle semi-structuré
• Langages de requêtes :
  qu'est-ce qu'une requête ?
  qu'est-ce qu'un bon langage de requêtes ?
    – divers paradigmes / divers problèmes
    – complexité / optimisation
    (inclusion, minimisation, ...)
    – expressivité (comparer les langages)

Organisation (classique) : cours, tds, devoir, ..., discussion

Biblio :

[AHV95] S.Abiteboul / R. Hull / V. Vianu : Foundations of Databases. Addison-Wesley Publishing Company, 1995. (Chapitres 3, 4 et 5 : notions élémentaires - bases de données relationnelles et langages de requêtes; Chapitre 16 : contexte général des langages de requêtes ; Chapitre 17 : Résultats d'expressivité et complexité des langages de requêtes.)

2 Modèle Relationnel (rappel) et Logique classique

On commence par faire un rappel rapide de quelques notions familières dans le domaine des bases de données (niveau licence). Le rappel est également fait pour que les notations utilisées dans la suite du cours soient explicitées.

2.1 Préliminaires

On suppose l'existence de trois ensembles :

1. un ensemble d'attributs noté **Attr**,
2. un ensemble de constantes noté **Dom** (on fera l'hypothèse simplificatrice que tout attribut a le même domaine **Dom**),
3. un ensemble de noms de relations noté **Rel** et

A chaque nom de relation R dans **Rel** on associe un ensemble d'attributs noté Attr(R). Donc un schéma de relation est spécifié par son nom. La cardinalité de Attr(R) est l'arité de la relation R, notée |R|. On notera parfois le schéma R par R(Attr(R)). Si Z = X∪Y avec X∩Y=ϕ, on le notera simplement par XY.

Un *schéma de base de données* $\mathcal{R}$ est un ensemble fini de schémas de relation. Les notions de n-uplet sur un schéma de relation R, d'instance I sur R et d'instance de schéma de base de données sont définies de façon classique :

Un n-uplet sur un ensemble d'attributs $\{A_1,A_2, ...,A_n\}$ est une fonction u de $\{A_1,A_2, ...,A_n\}$ dans Dom tel que $u(A_1) \in$ **Dom**. On peut aussi dire plus simplement que u est constitué de n valeurs $(a_1,a_2, ...,a_n)$ telles que pour tout $i = 1..n$, la valeur a_i appartient au domaine de l'attribut A_i. (Cette définition suppose que les attributs soient ordonnés). On note u$[A_i]$ la valeur a_i du n-uplet u que l'on appelle la composante de u sur A_i.

Un n-uplet sur $(A_1,A_2, ...,A_n)$ peut aussi être défini comme un élément du produit cartésien Dom$(A_1)\times$ ... $\times$ Dom(A_n) et une instance de R comme un sous ensemble **finie** de Dom$(A_1)\times$ ... $\times$ Dom(A_n).

On notera Inst(R) (resp. Inst($\mathcal{R}$)) l'ensemble des instances sur le schéma de relation R (resp. sur le schéma de base de données $\mathcal{R}$). Le domaine actif d'une instance $\mathcal{I}$ de $\mathcal{R}$, i.e. l'ensemble de toutes les constantes apparaissant dans $\mathcal{I}$, est noté adom($\mathcal{I}$).

2.2 Calcul relationnel –à variables domaines–

Le calcul relationnel est un langage de requête défini assez directement à partir de la logique classique du premier ordre. Nous allons donc commencé par construire le **pont** entre le modèle relationnel et la logique classique au premier ordre. Rappelons que nous avons supposé que tous les attributs ont le même domaine Dom.

Considérons donc un schéma $\mathcal{R}$=$\{ R_1, R_2, ..., R_m \}$. Associons à celui-ci le langage de la logique classique dont le vocabulaire (en dehors de variables x, y, ...) est constitué de:

- un ensemble Dom de symboles de constante,
- un ensemble vide de symboles de fonction
- un ensemble de prédicats $\{ R_1, R_2, ..., R_m \}$ (auquel on ajoute le prédicat d'égalité) tel que l'arité du prédicat R_i soit égal à l'arité du schéma de relation R_i.

Ce langage étant défini, on peut maintenant écrire des formules de ce langage en utilisant les règles bien connues rappelées maintenant. Une formule atomique est de la forme

- $x = y$ ou $x = a$ où a est une constante,
- $R_i(t_1, ..., t_n)$ où n est l'arité de R_i et t_j est un terme i.e. une variable ou une constante (puisque pas de symboles de fonction).

Une formule bien formée est définie récursivement par :

- toute formule atomique est une formule bien formée,
- soient φ_1 et φ_2 deux formules bien formées alors $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ sont des formules bien formées
- soit φ_1 une formule bien formée alors $\exists x\varphi_1$ est une formule bien formée.

Bien sûr les connecteurs de disjonction $\vee$ et d'implication $\rightarrow$ ainsi que la quantification universelle $\forall$ peuvent être utilisées pour faciliter l'écriture des formules.

La sémantique d'un langage du premier ordre nécessite d'introduire la notion d'interprétation, de valuation des variables, de satisfaction d'une formule par une interprétation. Une interprétation $\mathcal{I}$ est un triplet ($\mathcal{D}$, I_c, I_P) où I_c est une fonction de l'ensemble des constantes dans $\mathcal{D}$ et I_P est une

fonction qui à chaque prédicat R_i d'arité n associe une relation r_i d'arité n sur $\mathcal{D}$.
Remarque : nous ne nous intéresserons qu'à des interprétations qui correspondent à des instances du schéma $\mathcal{R}$ et nous ferons donc les trois restrictions suivantes :

1. $\mathcal{D}$=Dom
2. I_c est l'identité
3. I_P associe à tout prédicat R_i une relation **finie** r_i

Les deux premières restrictions définissent ce qui est appelé classiquement une interprétation d'Herbrand. La dernière restriction correspond au fait, que jusqu'à présent, nous ne considérons que des bases de données finies.
Une **valuation** ν d'un ensemble de variables $\{x_1, ..., x_n\}$ dans $\mathcal{D}$ est une fonction telle que pour tout $i = 1..n$, $\nu(x_i) \in \mathcal{D}$.
Une requête q du calcul à variables domaine (dont les objets manipulés sont des objets apparaissant dans les colonnes d'une table) est spécifiée par une expression $\{(\overrightarrow{x}) \mid \varphi(\overrightarrow{x})\}$ où $\varphi(\overrightarrow{x})$ est une formule de la logique classique et $\overrightarrow{x}$ sont les variables libres de φ.
La réponse à une requête q évaluée sur l'instance $\mathcal{I}$ de $\mathcal{R}$ est une relation (dont l'arité correspond au nombre de variables libres de $\vec{x}$) :
q($\mathcal{I}$)=$\{\nu(\overrightarrow{x}) \mid \mathcal{I}, \nu \models \varphi(\overrightarrow{x})$ où ν valuation de $\overrightarrow{x}\}$.

Rappel de la définition de $\models$: Soit $\mathcal{I}$=($\mathcal{D}$, I_c, I_P) une interprétation de notre langage et soit φ une formule de notre langage dont l'ensemble des variables (toutes les variables) est $\{x_1, ..., x_n\}$, soit ν une valuation de ces variables. L'interprétation $\mathcal{I}$ satisfait φ pour la valuation ν (ce qui sera noté $\mathcal{I}, \nu \models \varphi$) si :

- Si φ est la formule atomique $x = y$ (respectivement $x = a$ où a est une constante), alors $\nu(x) = \nu(y)$ (resp. $\nu(x) = a$).
- Si φ est la formule atomique $R_i(t_1, ..., t_m)$ alors $(\nu'(t_1), ..., \nu'(t_m)) \in r_i$ où $\nu'(t_i)$=$\nu(t_i)$ si t_i est une variable et $\nu'(t_i)$=t_i si t_i est une constante. Rappel : $r_i = I_P(R_i)$.
- Si φ est de la forme $\neg\varphi_1$ alors $\mathcal{I}, \nu \not\models \varphi_1$
- Si φ est de la forme $\varphi_1 \wedge \varphi_2$ alors $\mathcal{I}, \nu \models \varphi_1$ et $\mathcal{I}, \nu \models \varphi_2$.

- Si φ est de la forme $\exists x \varphi_1$ alors il existe une valuation ν', (attention : coorection faite ici) telle que pour tout i, $x_i \neq x$, $\nu'(x_i) = \nu(x_i$ et telle que $\mathcal{I}, \nu' \models \varphi_1$.

Exemple : Considérons un schéma de base de données (bien trop connu !!)

Film(Titre, Metteur-en-scène, Acteur)
Cine(Nom-Ciné, Adresse, Téléphone)
Prog(Nom-Ciné, Titre, Horaire)

et les requêtes ... :

1. Donnez les titres de tous les films.

   $\{ x_t \mid \exists x_m \; \exists x_a \; \text{Film}(x_t, x_m, x_a) \}$

2. Quels sont les acteurs des films à l'affiche ?

   $\{ x_a \mid \exists x_c \; \exists x_t \; \exists x_h \; \exists x_m \; (\text{Prog}(x_c, x_t, x_h) \wedge \text{Film}(x_t, x_m, x_a)) \}$

3. Quels sont les films qui ne sont pas à l'affiche ?

   $\{ x_t \mid \exists x_m \; \exists x_a \; (\; \text{Film}(x_t, x_m, x_a) \wedge \forall x_c \; \forall x_h \; \neg \; \text{Prog}(x_c, x_t, x_h) \;)\}$

4. Donnez les cinémas et les horaires de projection après 20h00 du film "Etre et Avoir"

   $\{ (x_c, x_h) \mid \exists x_t \; (\text{Prog}(x_c, x_t, x_h) \wedge x_t = \text{"Etre et Avoir"} \wedge x_h > \text{"20h00"} \;) \}$

   $\{ (x_c, x_h) \mid (\text{Prog}(x_c, \text{"Etre et Avoir"}, x_h) \wedge x_h > \text{"20h00"} \;) \}$

5. Donnez les cinémas qui projettent un film dans lequel joue M.F Pisier ?

   $\{ (x_c) \mid \exists x_t \; \exists x_h \; \exists x_m \; \exists x_a \; (\text{Prog}(x_c, \; x_t, \; x_h) \wedge \text{Film}(x_t, \; x_m, \; x_a) \wedge x_a = \text{"MF Pisier"} \;) \}$

6. Quels sont les (titres) des films dont un des acteurs est metteur en scène de ce film ?

   $\{ x_t \mid \exists x_{ma} \; \exists x_a \; \exists x_m \; (\; \text{Film}(x_t, x_{ma}, x_a) \wedge \text{Film}(x_t, x_m, x_{ma}) \;) \}$

   En supposant que soit la df Titre $\rightarrow$ M-e-S soit la dmv Titre $\rightarrow\!\!\rightarrow$ M-e-S | Acteur est satisfaite par l'instance, la requête s'écrit alors :

   $\{ x_t \mid \exists x_{ma} \; (\; \text{Film}(x_t, x_{ma}, x_{ma}) \;) \}$

7. Donnez les metteurs en scène qui sont aussi acteurs (pas forcément dans le même film).

{ x_{ma} | $\exists x_{t_1}$ $\exists x_a$ $\exists x_m$ $\exists x_{t_2}$ (Film(x_{t_1}, x_{ma}, x_a) $\wedge$ Film(x_{t_2}, x_m, x_{ma})) }

même possibilité de simplification en présence de contraintes d'intégrité.

8. Quels sont les films dirigés par au moins deux metteurs en scènes ?

   { x_t | $\exists x_{m_1}$ $\exists x_{m_2}$ $\exists x_{a_1}$ $\exists x_{a_1}$ (Film(x_t, x_{m_1}, x_{a_1}) $\wedge$ Film(x_t, x_{m_2}, x_{a_2}) $\wedge$ $x_{m_1} \neq x_{m_2}$) }

9. Quels cinémas projettent tous les films de Maurice Pialat ?
   (j'ai simplifié ici)
   { (x_c) | $\exists x_t$ $\exists x_h$ Prog(x_c, x_t, x_h) $\wedge$ $\forall x_{t_P}$ $\forall x_m$ $\forall x_a$ [(Film(x_{t_P}, "Pialat", x_a)) $\rightarrow$ $\exists x_{h_P}$ Prog(x_c, x_{t_P}, x_{h_P})] }

Calcul à variable domaine versus calcul à variable n-uplet

... juste un exemple peut-être pour donner une idée et traiter cela soit en travaux dirigés soit en devoir ....

Dans ce cas, les variables servent à "abstraire" les n-uplets des relations au lieu des valeurs du domaine des attributs. Il faut donc "penser" à x, y et z comme à des n-uplets. Seule la définition des formules atomiques change :

- R(x) est une formule atomique,
- x.A = y.B est une formule atomique, où x.A (resp. y.B) désigne le composant sur A (resp. sur B) de la variable n-uplet x (resp. y).
- x.A = 'a' est une formule atomique.

Exemple : Donnez les cinémas et les horaires de projection après 20h00 du film "Etre et Avoir".

{ (x_{out} | $\exists x_p$ (Prog(x_p) $\wedge$ x_p.titre='Etre et Avoir' $\wedge$ x_p.horaire $\geq$ 20h00 }
$\wedge$ x_{out}.Nom-cine = x_p.Nom-cine
$\wedge$ x_{out}.Horaire = x_p.Horaire)

La particularité du calcul à variable n-uplet par rapport au calcul à variable domaine réside dans la manière dont est déterminé le "type" d'une variable n-uplet. Prenons l'exemple ci-dessus : le "type" de la variable x_p est le type de la relation Prog à cause de l'occurrence de la formule atomique Prog(x_p); le "type" de la variable x_{out} est (Nom-cine, Horaire) à cause de l'occurrence des formules atomiques x_{out}.Nom-cine = x_p.Nom-cine

et x_{out}.Horaire = x_p.Horaire).

Il est important de remarquer le lien entre le calcul à variable n-uplet et le langage commercial SQL. La requête ci-dessus peut s'écrire de la manière suivante en SQL :

SELECT x_p.Nom-cine, x_p.Horaire
FROM PROG AS x_p
WHERE x_p.titre='Etre et Avoir' AND x_p.horaire $\geq$ '20h00'

Expression saine, indépendance du domaine Considérons le schéma constitué des deux prédicats binaires R et S. Considérons l'instance $\mathcal{I}$ dessinée ci-dessous :

R	A	B
	a_1	b_1
	a_2	a_2

S	C
	a_3
	a_2

1. Considérons l'expression q_1 : $\{ x \mid \neg S(x) \}$. En supposant que le domaine de valuation des variables est le domaine actif, la réponse à cette requête est $q(\mathcal{I})$=$\{ a_1, b_1 \}$ (corrigé).

   En supposant maintenant que le domaine de valuation des variables est un ensemble DOM quelconque (mais incluant le domaine actif de $\mathcal{I}$), la réponse à cette requête est $q(\mathcal{I})$=DOM $- \{ a_3, a_2 \}$. Notons en particulier que si DOM est un ensemble infini, la réponse à l'expression ci-dessus n'est pas une relation (au sens où nous l'entendons en base de données même si c'est une relation au sens mathématique) puisque le résultat est infini !!!

   La réponse à cette expression dépend donc du domaine de valuation considéré; ce n'est pas une expression indépendante du domaine.

2. Considérons l'expression q_2 : $\{ (x) \mid \exists y R(x, y) \wedge \neg S(x) \}$. La réponse à cette requête est $q(\mathcal{I})$=$\{ a_1 \}$. On peut observer que quelque soit le domaine de valuation des variables choisi (en supposant qu'il contient les constantes apparaissant dans l'instance $\mathcal{I}$), la réponse à cette requête est identique. On dit que cette expression est indépendante du domaine.

Il y a plusieurs manières de résoudre ce problème (dans le but évidemment de pouvoir utiliser la logique classique comme langage de requêtes).

1. N'autoriser à écrire que les requêtes dont le résultat est construit à partir des constantes de l'instance interrogée (plus quelques constantes fixées). Par exemple, $\{ (x) \mid \exists y R(x,y) \wedge \neg S(x) \}$ satisfait cette condition car le résultat de cette requête appartient à la projection de R sur l'attribut A.

   Cette condition n'est toutefois pas tout à fait suffisante. Il faut aussi s'assurer que les "sous-requêtes" n'utilisent pas non plus des constantes en dehors de celles présentes dans l'instance interrogée. Par exemple, $\{ (x) \mid \exists x R(x,y) \wedge \exists z \neg S(z) \}$ satisfait bien la première condition mais pas la deuxième. D'ailleurs on pourrait facilement montrer que la réponse dépend du domaine d'évaluation des variables dans ce cas.

   Dans la littérature, les expressions qui satisfont ces conditions (ici introduites uniquement intuitivement) sont appelées **expressions saines**.

   La définition d'une expression saine n'est absolument pas constructive. Vérifier si une expression est saine est d'ailleurs un problème très difficile ! On sait en particulier qu'il n'existe pas de condition syntaxique nécessaire et suffisante sur les expressions permettant de conclure qu'elles sont saines. Il y a eu dans la littérature beaucoup de propositions de conditions suffisantes. Nous en verrons une : **expression à domaine restreint**.

2. N'autoriser à construire que les requêtes dont le résultat ne varie pas lorsque le domaine de valuation des variables lui varie. Cette restriction appelée **domaine indépendance** que nous définissons en détail par la suite n'est pas plus constructive que la précédente. Elle est tout aussi "impossible" à tester. Même conclusion. Elle assure que le "résultat" d'une requête est fini.

3. Finalement, au lieu d'essayer de restreindre les expressions qui seront utilisées pour écrire des requêtes, il y a une autre voie qui consiste à modifier le sémantique des expressions (ie modifier la définition de la réponse à une requête) en restreignant le domaine de valuation des variables aux constantes apparaissant dans l'instance (plus quelques autres). Le calcul ainsi obtenu est appelé **calcul à domaine actif** et sera présenté en détails dans la suite.

Des résultats très intéressants lient ces trois "solutions". Nous avons besoin pour poursuivre de définir la notion de **domaine actif** et aussi de **satisfaction relative à un domaine**.

Définition : Etant donné une instance $\mathcal{I}$ et une expression $\varphi(\overrightarrow{x})$ du calcul, le domaine actif de $\mathcal{I}$ noté adom($\mathcal{I}$) est l'ensemble des constantes apparaissant dans $\mathcal{I}$. Le domaine actif de $\mathcal{I}$ et φ est l'union de adom($\mathcal{I}$) et de l'ens. des constantes apparaissant dans la requête. On utilisera la même notation adom.

Exemple : Pour l'instance de l'exemple ci-dessus, adom($\mathcal{I}$) =$\{$ a_1, a_2, a_3, b_1 $\}$. Pour cette instance, adom($\mathcal{I}$, q_1)= $\{$ a_1, a_2, a_3, b_1 $\}$=adom($\mathcal{I}$, q_2). Considérons la requête q_3=$\{$ $x \mid \exists y \neg S(x) \wedge y = a\}$. Alors adom($\mathcal{I}$, q_3)= $\{$ a_1, a_2, a_3, b_1, a $\}$.

La notion de **satisfaction relative à un domaine D** est très intuitive : elle consiste à utiliser les éléments de **D** pour valuer les variables des formules au lieu d'utiliser les éléments de Dom. Dans ce cas le domaine "relatif" est indiqué en indice de la requête. Attention, cette évaluation n'a de sens que si **D** contient au moins le domaine actif de l'instance et de l'expression évaluée.

Définition : Pour une expression du calcul relationnel q=$\{$ $(\overrightarrow{x})$ $\mid$ ... $\varphi(\overrightarrow{x})$ $\}$ et une instance $\mathcal{I}$, l'évaluation relative au domaine actif de q est l'évaluation relative au domaine actif de $\mathcal{I}$ et φ.

Exemple : L'évaluation de l'expression q_1 relative au domaine actif produit l'ensemble fini des n-uplets suivants : ${q_1}_{adom}(\mathcal{I})$=$\{$ (a_1), $(b_1)\}$.

L'évaluation de l'expression q_2 relative au domaine actif produit l'ensemble fini des n-uplets suivants : ${q_2}_{adom}(\mathcal{I})$=$\{$ (a_1) $\}$.

L'évaluation de l'expression q_3 relative au domaine actif produit l'ensemble fini des n-uplets suivants : ${q_3}_{adom}(\mathcal{I})$=$\{$ (a_1), (b_1), (a) $\}$.

Définition : Une expression $\varphi(\overrightarrow{x})$ du calcul relationnel est **indépendante du domaine** ssi pour toute instance $\mathcal{I}$, pour tout couple de domaines D_1 et D_2 tel que $D_1 \supseteq adom(\mathcal{I})$ et $D_2 \supseteq adom(\mathcal{I})$, on a : ${q_{D_1}}(\mathcal{I}) = {q_{D_2}}(\mathcal{I})$.

Exemple : Il est assez facile de voir que les requêtes q_1 et q_3 ne sont pas indépendantes du domaine et que la requête q_2, elle, est indépendante du domaine.

Exemple (plus élaboré): Considérons un schéma de relation (prédicat) ternaire R et l'expression q_4 du calcul relationnel suivante :
$\{$ $x \mid \forall z(\exists y R(x, y, z) \rightarrow \exists v R(v,' a', z)$ $\}$.

Cette requête n'est pas indépendante du domaine car elle rend les "x" satisfaisant la propriété exprimée par la formule et appartenant à la projection de R sur la première composante **et** les 'x' qui n'appartiennent pas à cette projection ! Choisissons l'instance $\mathcal{I}$ de R.

R			
	a_1	a	c_1
	a_1	a	c_2
	a_2	a_2	c_1

adom($\mathcal{I}$, q_4)={ a , a_1, a_2 c_1 , c_2 }.
$q_{4_{adom}}(\mathcal{I})$= { a_1 } ∪ { a, c_1 , c_2 }.

Soit D=adom($\mathcal{I}$) ∪ { c }.
$q_{4_D}(\mathcal{I})$= { a_1 } ∪ { a, c_1 , c_2 , c }.

Autre exemple :
$q_5 = \{(xy) \mid \exists z(R(x,z) \vee y = a)\}$

Lemme : Toutes les requêtes indépendantes du domaine sont des requêtes relatives au domaine actif dans le sens suivant : Si q est un expression du calcul relationnel indépendante du domaine alors pour toute instance, son évaluation sur un domaine quelconque (incluant le domaine actif de $\mathcal{I}$) est égale à l'évaluation relative au domaine actif.

Théorème : Tester si une requête est indépendante du domaine est indécidable.

Ce résultat est démontré à partir d'un autre résultat important établissant que la satisfaisabilité (finie) des expressions du calcul relationnel est récursivement énumérable mais pas récursif. Il faut utiliser le fait que si φ est une expression dépendante du domaine alors $\varphi \wedge \psi$ est indépendante du domaine ssi ψ est insatisfiable.

On sait donc qu'il n'existe pas de condition syntaxique nécessaire et suffisante sur les expressions permettant de conclure qu'elles sont indépendantes du domaine. Il y a eu dans la littérature beaucoup de propositions de conditions suffisantes. Nous allons nous intéresser à l'une d'entre elles.

Considérons les règles de réécritures suivantes :

- Renomage des variables afin d'obtenir une formule sans réutilisation de variables. Par exemple la formule $\exists x \exists y(R(x,y) \rightarrow \forall x S(x)$ sera réécrite en $\exists x \exists y(R(x,y) \rightarrow \forall z S(z)$
- Suppression des $\forall$. Par exemple $\forall x S(x)$ est réécrite en $\neg\exists x \neg S(x)$.
- Suppression de $\rightarrow$. Par exemple $R(x,y) \rightarrow S(x)$ est réécrit en $\neg R(x,y) \vee S(x)$.

- Pousser les $\neg$ aux feuilles ie remplacer
  - $\neg\neg\varphi$ par φ
  - $\neg(\varphi_1 \vee ... \vee \varphi_m)$ par $\neg\varphi_1 \wedge ... \wedge \neg\varphi_m$
  - $\neg(\varphi_1 \wedge ... \wedge \varphi_m)$ par $\neg\varphi_1 \vee ... \vee \neg\varphi_m$ Dans l'arbre syntaxique de la formule "traitée" par ses règles, le fils d'un $\neg$ est soit un atome soit une formule existentiellement quantifiée.
- désimbrication (il faut mieux dire aplatissement : en fait ça consiste à rendre les connecteurs binaires naires) des $\wedge$, $\vee$ et $\exists$ pour que dans l'arbre syntaxique de la formule, le fils d'un $\wedge$ ne soit pas un $\wedge$.

L'application itérée sur la formule φ de ces règles de réécriture (jusqu'à impossibilité d'appliquer une règle) produit une formule notée DR(φ).

Définition : Une formule φ est sous forme normale DR si φ=DR(φ).

Proposition : On a $\varphi \equiv$ DR(φ) ie (par définition de $\equiv$), pour toute instance $\mathcal{I}$ et pour toute valuation ν des variables libres de φ, $\mathcal{I}, \nu \models \varphi$ ssi φ, $\mathcal{I}, \nu \models$ DR(φ).

L'algorithme qui suit sert à tester si une formule φ est à domaine restreint est maintenant présenté ci-dessous. Cet algorithme calcule l'ensemble des variables "restreintes" par le domaine actif par la syntaxe de la formule. L'algorithme rend l'ensemble des variables libres restreintes ou un symbole spécial $\perp$ indiquant qu'une variable quantifiée existentiellement dans une sous-formule n'est pas restreinte.

Algo DomRest
`Entrée :` une formule φ sous forme normale DR
`Sortie :` un sous-ensemble des variables libres de φ ou $\perp$
Case φ of

R(t_1, ...,t_m) :	vardr(φ) = var(t_1, ...,t_m);
$x = a$ ou $a = x$:	vardr(φ)={ x }
$\varphi_1 \wedge \varphi_2$:	vardr(φ)=vardr(φ_1) $\cup$ vardr(φ_2).
$\varphi_1 \wedge x = y$:	vardr(φ)= vardr(φ_1) si $\{x, y\} \cap$ vardr(φ_1)=$\emptyset$ vardr(φ_1) $\cup \{x, y\}$ sinon
$\varphi_1 \vee \varphi_2$:	vardr(φ)=vardr(φ_1) $\cap$ vardr(φ_2).
$\neg\varphi_1$:	vardr(φ)=$\emptyset$
$\exists x\varphi_1$:	vardr(φ)= vardr(φ_1)$-\{x\}$ si vardr(φ_1) $\sqsupseteq \{x\}$ vardr(φ) = $\perp$ et exit sinon

End Case

Intuitivement, l'occurrence d'une variable x dans une relation de base (un prédicat) ou dans un atome de la forme $x = a$ "restreint" la variable par le domaine actif. Cette restriction est propagée par la conjonction, elle peut être perdue par la disjonction, elle est toujours perdue sous une négation. En plus, chaque variable quantifiée doit être restreinte dans la sous formule où elle a une occurrence libre.

Définition Une formule φ sous forme normale DR est à domaine restreint si vardr(φ)=libre(φ).

Exemple : Si on considère la formule φ_1 de l'expression q_1, il est immédiat de voir que vardr(φ)=$\perp$.

Considérons la formule φ_4 de l'expression q_4. Il faut commencer par mettre φ_4 sous forme normale DR, ce qui donne : $\neg\exists z(\exists y R(x, y, z) \wedge \neg\exists v R(v,' a', z))$. On notera ψ la sous formule $\exists y R(x, y, z) \wedge \neg\exists v R(v,' a', z)$. Le calcul de vardr($\varphi_4$) se décompose comme suit :

vardr($R(x, y, z)$)	= { x, y, z }
vardr($\exists y R(x, y, z)$	= { x, z }
vardr($R(v,' a', z)$)	{ v, z }
vardr($\exists v R(v,' a', z)$	{ z }
vardr($\neg\exists v R(v,' a', z)$	$\emptyset$
vardr(ψ)	{ x, z } $\cup \emptyset$
vardr($\exists z\psi$)	{ x }
vardr($\neg\exists z\psi$)	$\emptyset$

Donc vardr(φ_4)=$\emptyset$ et puisque libre(φ_4)={ x }, φ_4 n'est pas à domaine

restreint.

Exemple : Reprenons la requête `"Quels cinémas projettent tous les films de Maurice Pialat ?"` exprimée par l'expression suivante du calcul.

$\{ (x_c) \mid \varphi(x_c) \}$ où

$$\varphi(x_c) :: \exists x_t \exists x_h (Prog(x_c, x_t, x_h) \wedge \forall x_{t_P} \forall x_m \forall x_a \, [(Film(x_{t_P}, x_m, x_a) \wedge x_m = "Pialat") \rightarrow \exists x_{h_P} Prog(x_c, x_{t_P}, x_{h_P})] \}$$

La formule $\varphi(x_c)$ doit d'abord être mise sous forme normale DR. DR(φ) est la formule ci-dessous :

$\exists x_t \exists x_h (Prog(x_c, x_t, x_h) \wedge$

.....

Le calcul de vardr(DR(φ)) se décompose comme suit :

Théorème : Les langages suivants sont équivalents :

- le calcul à domaine indépendant (Calc-di)
- le calcul relatif au domaine actif (Calc-da)
- le calcul à domaine restreint (Calc-dr)
- l'algèbre relationnelle (Alg)

Définition : Soit $\mathcal{L}_1$ et $\mathcal{L}_2$ deux langages de requêtes. Dire que $\mathcal{L}_1 \equiv \mathcal{L}_2$ c'est dire que pour toute requête $q_{\mathcal{L}_1}$ de $\mathcal{L}_1$ il existe une requête $q_{\mathcal{L}_2}$ de $\mathcal{L}_2$ telle que pour toute instance $\mathcal{I}$, on ait : $q_{\mathcal{L}_1}(\mathcal{I})=q_{\mathcal{L}_2}(\mathcal{I})$, et inversement pour toute requête $q_{\mathcal{L}_2}$ de $\mathcal{L}_2$, il existe une requête $q_{\mathcal{L}_1}$ de $\mathcal{L}_1$ telle que ...

Preuve :
• Nous montrerons en TD que Alg $\ll$ Calc-dr et que Alg $\ll$ Calc-da.
fait en dtails en cours)
• Nous allons montrer ici que Calc-da $\ll$ Alg. Considérons un schéma de base de données quelconque $\mathcal{R}$ et une expression $\varphi(\vec{x})$ (posons $\vec{x} = x_1, ..., x_k$) et la requête q=$\{ \vec{x} \mid \varphi(\vec{x}) \}$ de Calc-da. Le but est de construire (par induction sur la structure de $\varphi(\vec{x})$) une requête de l'algèbre relationnelle E telle que : $\forall \mathcal{I}$ instance de $\mathcal{R}$, $E(\mathcal{I})=q_{adom}(\mathcal{I})$.

On procède donc par induction sur la structure de la formule φ. Tout d'abord nous allons construire une requête algébrique E_{adom} permettant

d'extraire les constantes du domaine actif d'une instance. Cette requête algébrique servira à simuler le mode d'évaluation "domaine actif" de la requête calcul dans la suite.

E_{adom}=$\cup_{i=1..n}(\cup_{j=1..n_i}\pi_{A^i_j}[R_i])$

On vérifie aisémment que $\forall\ \mathcal{I}$ instance de $\mathcal{R}$, $E_{adom}(\mathcal{I})$=adom($\mathcal{I}$).

En fait, nous avons besoin d'étendre cette requête afin d'inclure dans le résultat de celle-ci les constantes apparaissant dans l'expression φ. Soit 'a' une constante apparaissant dans φ, considérons la requête constante $(< a >)$ (définie de façon triviale par $\forall\ \mathcal{I}\ (< a >)(\mathcal{I})=\{ < a > \}$). L'extension de E_{adom} est alors immédiate.

Dans la suite, les attributs des schémas de $\mathcal{R}$ tout comme les attributs des schémas des résultats intermédiaires sont numérotés. Si R est un schéma binaire alors son "premier" attribut est nommé 1 et son deuxième attribut est nommé 2.

- $\varphi := R(t_1, ..., t_l)$ (on va supposer sans perte de généralité que tous les t_i sont des variables) : $E_\varphi := R$
- $\varphi := (x_1 = x_2)$ alors : $E_\varphi := \sigma_{1=2}(E_{adom} \times E_{adom})$.

  $E_{adom} \times E_{adom}$ est une abbréviation pour $(E_{adom} \bowtie ren_{1\to 2}E_{adom})$
- $\varphi := (x_1 = a)$ où x_1 est une variable et a une constante : $E_\varphi := (< a >)$.
- $\varphi := \varphi_1(x_1, x_2) \wedge \varphi_2(x_2, x_3)$ où x_1, x_2 sont les variables libres de φ_1 et x_2, x_3 sont les variables libres de φ_2 (pour simplifier) : $E_\varphi := E_{\varphi_1} \bowtie ren_{1\to 2,2\to 3}(E_{\varphi_1})$
- $\varphi := \neg\varphi_1(x_1, x_2)$ alors : $E_\varphi := (E_{adom} \times E_{adom}) - E_{\varphi_1}$
- $\varphi := \exists x_2\varphi_1(x_1, x_2, x_3)$ alors : $E_\varphi := \Pi_{1,3}(E_{\varphi_1})$

Donc Calc-da $\equiv$ Alg !

• Il est (relativement) facile de montrer que toute requête de Calc-dr est indépendante du domaine et donc peut s'évaluer relativement à son domaine actif. Donc :

Alg $\ll$ Calc-dr $\ll$ Calc-di $\ll$ Calc-da $\ll$ Alg et les 4 langages sont équivalents !

SQL et QUEL Le langage SQL est un langage dont la syntaxe utilise certains aspects du calcul à variable n-uplet. Un autre langage "commercial" a été développé en s'inspirant du calcul : Quel (système INGRES)

3 Modèle relationnel et Langages de Requêtes

3.1 Fondements des langages de requêtes

... juste quelques mots pour faire la transition

Cette partie du cours a pour but d'introduire de façon fondamentale la notion de requête en faisant abstraction de tout langage particulier et les outils permettant de "mesurer" un langage de requêtes pour base de données.

En effet, jusqu'à présent, les langages (algèbre relationnelle, calcul relationnel) ont été introduits de façon adhoc ou sur la base de motivation intuitive :

- l'algèbre est LE langage du modèle relationnel
- l'algèbre ne permet pas de poser des requêtes récursives

Expressibité: fermeture transitive Nous avons vu pendant la révision de l'algèbre que certaines requêtes ne peuvent être exprimées à l'aide de ce langage (ni donc à l'aide du calcul). Il s'agissait de la requête demandant si il existe un moyen d'aller de Paris à LA en avion.

Nous avons vu qu'il n'est possible de "compter" que de façon très limitée (donnez les aviateurs qualifiés pour piloter au moins 2 appareils, ou 3 ..). Nous avons vu qu'il n'est pas possible de retrouver le pilote qualifié pour piloter le plus d'appareils.

Toutes ces requêtes "infaisables" en algèbre nécessite d'effectuer un nombre de jointures qui n'est pas prédéterminé et qui dépend en fait du contenu de l'instance considérée. On peut très bien imaginer réussir à écrire une expression de l'algèbre relationnelle qui permette d'extraire les possibilités d' "aller de Paris à LA" en sachant que dans l'instance aucune connection (chemin aérien) n'a une longueur supérieure à 4 (par exemple !). Mais si l'instance est modifiée ... l'expression peut ne plus rendre le résultat escompté. Donc l'expression n'exprime pas la requête "aller de Paris à LA".

Soit G un schéma de relation binaire servant à stocker les arcs d'un graphe. La **fermeture transitive** est la requête qui a toute instance g de G associe la plus petite relation binaire ft satisfaisant : $g \subset ft$, si $(s_1, s_2) \in ft$ et $(s_2, s_3) \in ft$ alors $(s_1, s_3) \in ft$.

Intuitivement, la fermeture transitive d'une instance g de G (donc d'un graphe) est l'ensemble des couples de sommets connectés par un chemin.
Résultat : La fermeture transitive d'un graphe est une requête qui ne peut être exprimée ni à l'aide de l'algèbre, ni à l'aide du calcul.
La preuve de ce résultat est très complexe et ... sera faite en Master Recherche ...

Expressibité: parité **Résultat :** La requête de parité (qui consiste à vérifier si une relation contient un nombre pair de n-uplets) ne peut pas être exprimée ni à l'aide de l'algèbre, ni à l'aide du calcul.

Pour montrer cela, on montre que le calcul admet une loi 0-1. La preuve de ce résultat est très complexe et ... sera faite en Master Recherche ...

Les différents paradigmes de langages de requêtes

- **Paradigme impératif :** les opérateurs sont ensemblistes, certains opérateurs spécialisés sont introduits pour le filtrage de données. Exemple : l'algèbre relationnelle.
- **Paradigme logique :** les requêtes sont des formules de la logique classique (ou d'une logique xxx). Exemple : le calcul relationnel.
- **Paradigme de règles de production (programmation logique) :** les requêtes sont des programmes logiques. Exemple : Datalog.

Généralités : Les langages que nous allons présentés partagent un certain nombre de caractéristiques communes:

1. Les langages pourront utilisés (en plus du schéma source) un schéma intermédiaire **fixe**. En d'autres termes, une requête dans un langage peut référencer des relations autres que celles du schéma source mais ne peut dynamiquement créer des relations.

2. Tout résultat intermédiaire est constitué uniquement de constantes apparaissant dans l'instance source ou éventuellement dans la requête. En d'autres termes et intuitivement, il n'y a pas d'*invention de constantes* lors de l'exécution des requêtes.

   En conséquence, le nombre de n-uplets intermédiaires pouvant être calculés à chaque étape d'évaluation d'une requête est majoré par $\sum_{i=1}^{k}(n+c)^{r_i}$ où

   - r_i est l'arité des relations intermédiaires,

- k est le nombre de relations intermédiaires,
- n est le nombre de constantes de l'instance source,
- c est le nombre de constantes apparaissant dans la requête.

Les langages "inflationistes" utilisent des itérations cumulatives (à chaque itération de nouveaux n-uplets sont calculés, aucun n'est retiré). Ce type de mécanisme assure la terminaison (la définition de langage de requêtes total) et une complexité en temps polynomial. Les langages "non inflationistes" utilisent des itérations destructives (à chaque itération des n-uplets peuvent être ajoutés, d'autres supprimés). Ce type de mécanisme n'assure pas la terminaison. La complexité des calculs est en espace polynomial.

3.2 Le langage WHILE

Le langage WHILE a pour noyau l'algèbre relationnelle (mais ce n'est pas le point important car on peut tout aussi bien utiliser le calcul). Il utilise le paradigme de la programmation impérative pour introduire des itérations dans le calcul d'une requête. Ce langage est donc de type procédural.

Syntaxe : Soit $\mathcal{S}=\{S_1, ..., S_n\}$ un schéma relationnel

Une instruction élémentaire de WHILE sur S est de la forme $S_i := E$ où $S_i \in \mathcal{S}$ et E est une expression algébrique sur $\mathcal{S}$.

Une instruction de WHILE sur $\mathcal{S}$ est soit une expression élémentaire, soit la composition de n instructions WHILE notée $\alpha_1; ...; \alpha_n$ soit une expression de la forme *while change do*
begin α *end*

où α est une instruction (programme) de WHILE sur $\mathcal{S}$.

Intuitivement l'instruction de type "while .." est une instruction permettant d'itérer l'exécution du corps de la boucle, de façon classique. Toutefois, il faut remarquer ici que la condition n'est pas explicite. Intuitivement, le "while change..." signifie "tant que le corps de la boucle effectue des changements...". L'imbrication de boucles "while" est bien entendu autorisée.

Nous pouvons maintenant définir ce qu'est une requête du langage WHILE. Etant donné un schéma de base de données $\mathcal{R}$, une requête q de WHILE sur $\mathcal{R}$ est specifiée par un triplet ($\mathcal{S}$, $\mathcal{P}$, Res) tel que :
(1) $\mathcal{S}$ est un schéma de base de données vérifiant $\mathcal{R} \subset \mathcal{S}$ (tout simplement $\mathcal{S}$ est le schéma $\mathcal{R}$ enrichi par quelques schémas de relation auxiliaires

nécessaires à l'écriture du programme),
(2) $\mathcal{P}$ est un programme WHILE sur $\mathcal{S}$ vérifiant que tout schéma apparaissant en partie gauche d'une affectation est dans $\mathcal{S} - \mathcal{R}$ (cette condition est là pour bien distinguer les schémas auxiliaires des schémas de relations de la base; les relations de la base de données ne peuvent pas être modifiés par une requête).
(3) $Res \in \mathcal{S} - \mathcal{R}$ (Res est un schéma de relation distingué dans $\mathcal{S}$ et qui est destiné à recevoir le résultat de la requête.
Le langage WHILE est un langage extrèmement simple. Sa sémantique est copiée sur la sémantique des langages impératifs. C'est la **sémantique de l'affectation** $S_i := E$ qui va distinguer la variante **non inflationiste** de la variante **inflationiste**. Dans les deux cas, il faut retenir que les instances des relations intermédiaires (les relations de $\mathcal{S} - \mathcal{R}$) sont initialisées à vide au début de l'évaluation de la requête q.

Sémantique Non Inflationiste : Soit $\mathcal{J}$ une instance de $\mathcal{S}$. L'évaluation non inflationiste de l'affectation $S_i := E$ s'effectue relativement à $\mathcal{J}$ en deux étapes comme suit :

1. L'expression E est évaluée sur $\mathcal{J}$ ie on effectue le calcul de $E(\mathcal{J})$.
2. L'instance de S est **remplacée** par le résultat de l'évaluation de E. ie $s \leftarrow E(\mathcal{J})$.

Donc l'évaluation de l'affectation *détruit* le contenu de S.

Définition : Etant donné une requête WHILE q=($\mathcal{S}$, $\mathcal{P}$, Res) sur $\mathcal{R}$ utilisant le schéma $\mathcal{S}$ et ayant pour schéma cible Res. Soit une instance $\mathcal{J}$ sur $\mathcal{S}$.
Alors l'effet de $\mathcal{P}$ sur $\mathcal{J}$ noté trans($\mathcal{J}$, $\mathcal{P}$) est une instance $\mathcal{K}$ sur $\mathcal{S}$ définie par :

- si $\mathcal{P}$ est une affectation $S_i := E$ alors $\mathcal{K}(S_j)$=$\mathcal{J}(S_j)$ pour tout $j \neq i$ et $\mathcal{K}(S_i)$ est égal au résultat de l'affectation comme définie précédemment.
- si $\mathcal{P}$ est $\alpha_1; ...; \alpha_p$ alors $\mathcal{K}$=trans($\mathcal{J}$, $\alpha_1; ...; \alpha_p$)= trans($\mathcal{K}'$, α_p) où $\mathcal{K}'$=trans($\mathcal{J}$, $\alpha_1; ...; \alpha_{p-1}$)
- si $\mathcal{P}$ est `While change do begin` α `end` alors $\mathcal{K}$=trans($\mathcal{J}$, $\mathcal{P}$) est définie comme la limite de la suite ci-dessous **quand cette limite existe**:

  $\mathcal{K}_0$=$\mathcal{J}$
  $\mathcal{K}_{i+1}$=trans($\mathcal{K}_i$,α)

La réponse à la requête q évaluée pour une instance $\mathcal{I}$ de $\mathcal{R}$ est définie par RES($\mathcal{K}$) où $\mathcal{K}$=trans($\mathcal{J}$, $\mathcal{P}$) avec $\mathcal{J}(R)$=$\mathcal{I}(R)$ pour tout $R \in \mathcal{R}$ et $\mathcal{J}(S)$=$\emptyset$ pour tout $S \in \mathcal{S} - \mathcal{R}$.

Exemple : Fermeture Transitive Considérons une base de données stockant les informations concernant un graphe orienté G. G(Orig,Dest) est ici un schéma de relation. La requête ci-dessous calcule tous les couples de sommets qui sont connectés par un chemin dans le graphe. Cette requête While est spécifiée par

1. le schéma $\mathcal{S}$={ G, T }. où T(Deb,Fin) est un schéma binaire auxiliaire
2. le schéma cible T
3. le "programme" While ci-dessous :

$T := ren_{Orig \to Deb, Dest \to Fin} G$;
while change do
begin
$\quad T := T \cup \pi_{Deb,Fin}[(ren_{Fin \to Orig}(T)) \bowtie (ren_{Dest \to Fin} G)]$;
end

Développer l'exemple avec le calcul de la réponse pour une instance particulière.

Le langage WHILE non inflationiste (dans la suite appelé WHILE tout simplement) définit un langage de requête partiel. En effet (voir exemple ci-dessous), l'évaluation inflationiste de l'affectation peut impliquer la non terminaison de l'évaluation d'une boucle *while*.

Exemple : Ajout-Retrait Considérons le même schéma. Le programme WHILE ci-dessous a pour objectif de supprimer les arcs (a, b) si il existe un chemin de a à b de longueur 2 et d'insérer un arc (a, b) si il existe un sommet non directement connecté à a et b.

$T := G$;
while change do
$\quad$ *begin*
$\quad$ Retrait := $\{(x, y) \mid \exists z T(x, z) \wedge T(z, y)\}$
$\quad$ Ajout := $\{(x, y) \mid \exists z(\neg T(x, z) \wedge \neg T(z, x) \wedge \neg T(y, z) \wedge T(z, y)\}$;

$T := (T \cup$ Ajout$) -$ Retrait ;
end

Sur l'instance { (a,a) , (a,b) (b,a) , (b,b) }, la suite développée pour calculer la sémantique de l'itération n'a pas de limite.

Résultat :

- Le problème consistant à déterminer si une requête WHILE termine pour toute instance source I est indécidable. (La preuve utilise le résultat d'indécidabilité de l'inclusion des requêtes algébriques.)
- Le problème consistant à déterminer si un requête WHILE termine pour une instance source $\mathcal{I}$ donnée est décidable.

Sémantique Inflationiste :

L'évaluation inflationiste de l'affectation $S := E$ s'effectue comme suit :

1. L'expression E est évaluée.
2. Le résultat de l'évaluation de E est ajouté au contenu de la relation S.

Donc l'évaluation de l'affectation est *cumulative*. Pour distinguer la version inflationiste de la version non inflationiste l'affectation inflationiste sera notée $S+ = E$ et le langage WHILE inflationiste sera appelé WHILE$^+$.

Exemple : Fermeture Transitive Le programme WHILE$^+$ ci-dessous calcule la fermeture transitive de G dans la relation cible T.

$T+ = ren_{Orig \to Deb, Dest \to Fin} G$;
while change do
begin
$T+ = \pi_{Deb,Fin}(ren_{Fin \to Orig}(T) \bowtie ren_{Dest \to Fin}(G))$;
end

Exemple : Ajout-Retrait Reprendre cet exemple avec la sémantique inflationiste.

3.3 Datalog

Voir photocopie

4 Du modèle relationnel au modèle à valeur complexe

La structure de données du modèle relationnel est extrèmement simple. Elle permet de modéliser un large spectre d'applications (en particulier de gestions) avec facilité. Par contre, cette structure s'avère "rigide" dès que l'on s'intéresse à des applications un peu plus sophistiquées.

4.1 Le modèle non normalisé

Ce modèle est le résultat de la levée de la contrainte imposant aux valeurs associées à un attribut (ie aux valeurs du domaine d'un attribut) d'une relation d'être des valeurs atomiques. Ecarter cette contrainte permet d'envisager, par exemple, d'autoriser les valeurs d'un attribut à être des instances de relations ... (**les poupées russes**) ... Nous allons donc revoir les notions de schémas et d'instances de schéma dans le cadre du modèle non normalisé.
On suppose dans la suite l'existence de

- un ensemble d'attributs (simples) A_1, A_2, A_3, ...
- pour chaque attribut simple A_i, un ensemble de valeurs atomiques appelé domaine (pour des raisons de simplicité on suppose que tous les attributs simples ont le même domaine `Dom`),
- un ensemble de noms de relation (appelés aussi attributs complexes) $R_1, R_2, ..., , S_1, ...$

Définition Un schéma $\mathcal{R}$ de relation non normalisée est défini récursivement par :

1. $R(A_1, A_2, ..., A_n)$ et pour i=1 à n, A_i est un attribut simple tels que ces attributs sont distincts deux à deux. L'ensemble des attributs simples de $\mathcal{R}$, noté Attr($\mathcal{R}$), est $\{A_1, A_2, ..., A_n\}$.
2. $R(R_1, R_2, ..., R_n)$ et pour i=1 à n, R_i est soit un attribut simple soit un schéma non normalisé tels que i≠j implique $R_i \neq R_j$, Attr(R_i) et Attr(R_j) sont disjoints (en posant pour convention que si $R_i = A$ alors Attr(R_i)={ A }).
   L'ensemble des attributs de $\mathcal{R}$, noté Attr($\mathcal{R}$), est l'union des attributs simples des R_i.

Exemple 1: (Dessinez le graphe associé au schéma)

- Personne (Nom , Age , Tels)
- Tels (NumTel)
- Emp(Personne, Salaire, NomProj)
- Entreprise (NomEnt , Proprio , Tels , Emp)

Exemple 2:

- Tels (NumTel)
- Projet(NomProj , Empl)
- Empl (NomEmp , Salaire)
- Entreprise(NomEnt , Proprio, Téléphones , Projet)

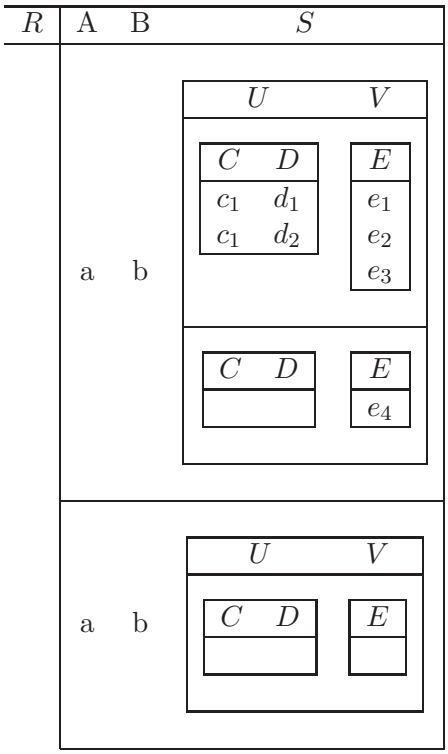
Exemple 3: R(A S(U(CD)V(E)) B) est une notation du schéma : R(ASB) où A et B sont des attributs simples, où S est l'attribut complexe défini par S(UV) où U et V sont deux attributs complexes définis par U(CD) et V(E) où CD et E sont des attributs simples.

Notation : pour simplifier l'écriture des schémas non normalisés, on choisit de regrouper "en tête" les attributs simples d'un schéma complexe. Pour l'exemple précédent cela donne R(ABS(U(CD)V(E))).

Définition Soit $R(R_1, R_2, ..., R_n)$ un schéma de relation non normalisée. Inst(R) désigne l'ensemble de toutes les instances possibles de R. Une instance $\mathcal{I}$ de R est un ensemble fini de n-uplets sur R. Un n-uplet u sur R est défini récursivement par :

1. Si $R(A_1, ..., A_n)$ et pour i=1 à n, A_i est un attribut simple (donc R est un schéma de relation "classique"). Alors u est une fonction de $\{A_1, ..., A_n\}$ dans `Dom` tel que $u(A_i) \in$ `Dom`
2. Si $R(A_1, ..., A_k, R_1..., R_n)$ et pour i=1 à k, A_i est un attribut simple et pour i=1..n, R_i est un attribut complexe. Alors u est une fonction $\{A_1, ..., A_k, R_1..., R_n\}$ dans `Dom`$\cup_{i=1..n} Inst(R_i)$ telle que pour i=1..k, $u(A_i) \in$ `Dom` et pour i=1..n, $u(A_i) \in Inst(R_i)$.

Exemples : Considérons le schéma de l'exemple 3 : R(ABS(U(CD)V(E))).



Commentaires:

- La définition n'interdit pas 2 n-uplets avec les mêmes valeurs d'attributs simples. Dans la pratique il est vrai que cette contrainte est pertinente. Elle correspond à un type particulier de relations non normalisées appelées V-relations ou encore à une forme normale de relations non normalisées appelé forme partitionnée.
- La définition permet que la valeur d'un n-uplet sur un attribut complexe soit un ensemble vide de n-uplets complexes. C'est particulièrement important car cela permet de traiter les valeurs nulles sans "sortir" du modèle du moins dans une certaine mesure.

Exemple : Donner une instance de R(A) et une instance de R(S(A)). Cet exemple sert à la transition avec la sous-section suivante.

4.2 Le modèle à valeur complexe

Nous allons maintenant voir comment cette notion de relation non normalisée peut être généralisée afin d'introduire encore davantage de souplesse (??) dans la définition des valeurs associées à un attribut.

Revenons à la définition d'une instance non normalisée et plus particulièrement à la définition d'un n-uplet ... et encore plus précicément à la valeur d'un n-uplet sur un attribut :

- si l'attribut est simple (par exemple A) alors u(A) est une valeur atomique
- si l'attribut est complexe (par exemple R) alors u(R) est un ensemble de n-uplets.

Donc la définition d'une instance d'un schéma non normalisé utilise deux constructeurs de valeurs : un constructeur de n-uplets (noté <> dans la suite) et un constructeur d'ensembles (noté {}). Reprenons nos exemples avec cette nouvelle vision des choses.

Exemple 1 : (suite) (Dessinez les graphes associés aux types)

- Une instance de Tels (NumTel) est un ensemble de n-uplets ie une instance est une valeur de type $\{<$ Dom $>\}$.
- Une instance de Emp(NomEmp, SalEmp, NomProjet) est un ensemble de n-uplets ... ie une valeur de type $\{< Dom, Dom, Dom >\}$.
- Une instance de Entreprise (NomEnt , Proprio , Tels , Emp) est une valeur de type $\{< Dom, Dom, \{< Dom >\}, \{< Dom, Dom, Dom > \} >\}$.

Exemple 2: (Dessinez les graphes associés aux types)

- Tels (NumTel) correspond au type $\{< Dom >\}$.
- Empl (NomEmp , SalEmp) correspond au type $\{< Dom, Dom >\}$.
- Projet(NomProj , Empl) correspond au type $\{< Dom, \{< Dom, Dom > \} >\}$.
- Entreprise(NomEnt , Proprio, Téléphones , Projet) correspond au type $\{< Dom, Dom, \{< Dom >\}, \{< Dom, \{< Dom, Dom >\} >\} > \}$.

Exemple 3: R(A B S(U(CD) V(E))) correspond au type
$\{< Dom, Dom, \{< \{< Dom, Dom >\}, \{< Dom >\} >\} >\}$.

Il est facile de remarquer que les "seuls" types de valeurs manipulées dans le cadre du modèle relationnel classique sont de la forme $\{<>\}$ ie les types "ensemble de n-uplets". Les seuls types de valeurs manipulées dans le cadre du modèle non normalisé sont $\{<>\}, , \{<, ..., \{<>\}, ..., \{<, ..., \{< \{< \{<>\}, , > \} >\}, ..., >\} >\}$ ie des types "ensemble de n-uplets" dont les composantes sont parfois des "ensembles de n-uplets". Pour le modèle non normalisé il faut remarquer donc cette alternance entre le constructeur d'ensemble et le constructeur de n-uplet. C'est cette restriction sur les types de valeurs manipulées qui va être levée dans le cadre du modèle à valeurs complexes.

Définition : Classiquement, on dispose dans la suite d'un ensemble de noms R, S, A, B, C et d'un type de base Dom. Un type τ est défini par :

$\tau = \text{Dom} \mid < B_1 : \tau_1, ..., B_n : \tau_n > \mid \{\tau\}$ où les B_i sont des noms (d'attributs) distincts deux à deux.
L'ensemble $[\![\tau]\!]$ des valeurs de type τ (l'interprétation de τ) est défini par (une valeur de type τ est appelée une valeur complexe):

1. $[\![\text{Dom}]\!] = \text{Dom}$,
2. $[\![< B_1 : \tau_1, ..., B_n : \tau_n >]\!] = \{< B_1 : c_1, ..., B_n : c_n > \mid c_j \in [\![\tau_j]\!], j \in [1, k]\}$
3. $[\![\{\tau\}]\!] = \mathcal{P}_{\text{finie}}^{[\![\tau]\!]}$.

Dans la suite, on considèrera un schéma de relation complexe comme un nom R auquel on associe un type τ. Une instance de R est alors un ensemble fini de valeurs complexes de type τ (donc une instance de R est en fait une valeur de type $\{\tau\}$.

Exemple :

type	valeur
Dom	a
$\{Dom\}$	$\{a, b, c\}$
$< A : Dom, B : Dom >$	$< A : a, B : b >$
$\{< A : Dom, B : Dom >\}$	$\{< A : a, B : b >, < A : a, B : c >\}$
$\{\{Dom\}\}$	$\{\{a, b\}, \{a\}, \{\}\}$

Remarques :
Une valeur complexe donnée peut avoir plusieurs types distincts. Il suffit de considérer la valeur { } (ensemble vide) qui est de type $\{\tau\}$ pour tout type τ.

Dans le cadre du modèle à valeurs complexes, il n'y a pas de restriction d'unicité de noms des attributs "imbriqués".
Exemple :
$\tau_1 = < A : Dom, B : Dom, C :< D : Dom, E : \{< F : Dom, G : Dom >\} >>$
$\tau_2 = < A : Dom, B : Dom, C :< D : Dom, E : \{\{F : Dom, G : Dom\}\} >$

Définition : Un schéma de relation à valeurs complexes est un couple (R,τ) où R est un nom (de relation) et où τ est un type. Un schéma de base de données est un ensemble fini de schémas de relations complexes.
Une instance d'un schéma de relation complexe est une valeur de type $\{\tau\}$.
Une instance d'un schéma de base de données est défini comme d'habitude.
Exemple : Reprendre les exemples donnés dans la présentation du modèle non normalisé et définir les types correspondant.

4.3 Algèbre pour le modèle à valeurs complexes

Le guide principal pour construire cette algèbre consiste à voir une valeur complexe comme un arbre ayant 2 types de noeud interne correspondant aux constructeurs d'ensemble * et de n-uplet <>. L'algèbre va proposer un ensemble d'opérations pour chaque type de noeud.

Dans la suite, I, I_1, ..., I_n sont des instances des relations $R : \tau$, $R_1 : \tau_1$, ... $R_n : \tau_n$. Rappelons qu'une instance I de $R : \tau$ est une valeur complexe de type τ.

• Les **opérations ensemblistes (union, intersection, différence)** sont définies de manière immédiate lorsque I_1 et I_2 sont des instances de $R_1 : \tau_1$ et $R_2 : \tau_2$ tel que $\tau_1 = \tau_2$.

• Les **opérations sur n-uplet** sont la **sélection** et la **projection**.

- Posons $\tau =< A_1 : \tau_1, ...A_n : \tau_n >$. Une condition de sélection de type τ est (modulo un bon typage) de la forme $A_i = v$ (v est une constante) , ou $A_i = A_j$, ou $A_i \in A_j$ ou $A_i = A_j.C$ (τ_j est un type n-uplet ayant un attribut C).

  $\sigma_\gamma(I) = \{u \mid u \in I$ et u satisfait $\gamma\}$

  La notion "u satisfait γ" est suffisamment usuelle pour ne pas être détaillée. Notons que $\sigma_\gamma(I)$ est une valeur complexe de type $\{\tau\}$.

- Considérons $p \leq n$.

  $\pi_{A_1 \ldots A_p}(I) = \{< A_1 : v_1, ..., A_p : v_p >|$ il existe v_{p+1}, ... v_n tels que $< A_1 : v_1, ..., A_n : v_n > \in I\}$

  Notons que $\pi_{A_1 \ldots A_p}(I)$ est une valeur complexe de type $\{< A_1 : \tau_1, ...A_p : \tau_p >\}$.

• Les opérateurs ci-dessous permettent la **construction d'ensembles** et la **construction de n-uplets**.

- part(I)=$\{v \mid v \subseteq I\}$. Notons que part(I) est de type $\{\{\tau\}\}$.
- set(I)=$\{I\}$. Notons que set(I) est de type $\{\{\tau\}\}$.
- Si $A_1, ..., A_n$ sont des attributs distincts alors nuplet$_{A_1,...,A_n}(I_1, ..., I_n)$= $\{< A_1 : v_1, ..., A_n : v_n >|$ pour tout i, $v_i \in I_i\}$.

  Notons que nuplet$_{A_1,...,A_n}(I_1, ..., I_n)$ est de type $\{< A_1 : \tau_1, ...A_n : \tau_n >$ $\}$

• Les opérateurs ci-dessous permettent la **destruction d'ensembles** et la **destruction de n-uplets**.

- Posons $\tau = \{\tau\prime\}$. unset(I) = $\{v \mid$ il existe $w \in I$ tel que $v \in w\}$.

  Notons que unset(I) est une valeur complexe de type $\{\tau\prime\}$.
- Posons $\tau =< A : \tau\prime >$. unuplet(I) =$\{v \mid < A : v > \in I\}$.

  Notons que unuplet(I) est une valeur complexe de type $\{\tau\prime\}$.

Définition d'une requête :

1. si R est un schéma de relation complexe alors $[R]$ est une requête dont le type est τ (le type de R). On a alors $[R](\mathcal{I})=\mathcal{I}(R)$.

2. si $a \in Dom$ alors $\{a\}$ est une requête dont le type est Dom. On a alors $\{a\}(\mathcal{I})=\{a\}$

3. soit q_1, ..., q_n des requêtes de type τ_1, ..., τ_n. Alors les opérateurs précédents permettent de définir une nouvelle requête dont le type est donné par la définition de chaque opérateur.

De nombreux exemples seront proposés en cours et en Tds.
Pour faciliter l'écriture des requêtes, on peut ajouter un certain nombre d'opérateurs à ceux introduits précédemment. Nous étudierons certains des opérateurs ci-dessous en TDs.

- valeurs complexes
- renomage
- produit cartésien
- jointure
- singleton
- Nest et Unnest

4.4 Calcul pour valeurs complexes

Le calcul pour le modèle à valeurs complexes est défini sur la base de la logique du premier ordre multi-sort. La notion de "sort" doit être rapprochée ici à celle de type introduit lors de la définition de ce modèle.

Dans la suite, pour chaque type (sort) τ, on supposera l'existence d'un ensemble enumerable de variables x de ce type. Une variable est dite **atomique** si elle est de type Dom.

- Un **terme** est soit un élément atomique (une constante), soit une variable, soit une expression de la forme $x.A$ si le type de x est $< ...A : \tau\prime ... >$.

- Une formule atomique est de l'une des formes suivantes :

  $R(t)$ si R est de type τ et t est un terme de type τ

  $t = t\prime$ si t et $t\prime$ sont de même type τ

  $t \in t\prime$ si t est de type τ et $t\prime$ de type $\{\tau\}$.

  $t \subseteq t\prime$ si t et $t\prime$ sont de même type $\{\tau\}$.

- Une formule est alors définie de façon classique en utilisant les connecteurs logiques $\wedge \vee \neg \rightarrow$ et les quantificateurs $\exists$ et $\forall$.

Une requête est une expression de la forme $\{x : \tau \mid \varphi\}$.

Le reste de la présentation du calcul sera faite en tds.

Exemple : `Le Jardin d'enfants`

Jardin : { < Enfants : { Enfant: Dom }
Age : Dom
Activités : { Activité : Dom }
>
}

Requête 1: La liste des activités de Noémie.

{ x | ∃ y (Jardin(y) ∧ x ∈ y.Activités ∧ "Noémie" ∈ y.Enfants) }

Requête 2: La liste des enfants pour chaque activité.

{ w | ∃ y (Jardin(y) ∧
w.Activité ∈ y.Activités ∧
∀ z (z ∈ w.Enfants ⇔ ∃ y' (Jardin(y') ∧
w.Activité ∈ y'.Activités ∧
z in y'.Enfants
)
)
)
}

Le type de w est < Activité : Dom, Enfants : { Enfant : Dom } >

Exemple : `La Famille`

Familles : { < Père : < Nom : Dom, Prof : Dom, Age: Dom >
Mère : < Nom : Dom, Prof : Dom, Age: Dom >
Enfants : { < Prénom : Dom, Age : Dom > }
>
}

Requête : Les enfants d'une personne

Type des éléments de la réponse :
< Nom : Dom , Enfants : { < Prénom : Dom, Age : Dom > }

Traduction 1:

$$\{ w \mid \exists f \exists p (\text{Familles}(f) \wedge ((f.\text{Mère} = p \vee f.\text{Père} = p) \wedge (w.\text{Nom} = p.\text{Nom}) \wedge (w.\text{Enfants} = f.\text{Enfants}))) \}$$

Traduction 2:

$$\{ w \mid \forall z (z \in w.\text{Enfants} \Leftrightarrow (\exists f (\text{Familles}(f) \wedge (f.\text{Mère} = p \vee f.\text{Père} = p) \wedge (w.\text{Nom}=p.\text{Nom}) \wedge (z \in f.\text{Enfants})))) \}$$

Récursivité

Considérons le schéma (ou le type)

$R : \star$ [AB : $\times$ [A : D , B : D]]

Calcul de la fermeture transitive de R :

1. La formule F_1 calcule l'ens. R_1 de toutes les paires construites avec des valeurs de R :

   $F_1 = \{ v \mid \exists y , z (R(y) \wedge R(z) \wedge$
   $(v.A = y.A \vee v.A = y.B) \wedge$
   $(v.B = z.A \vee v.B = z.B)) \}$

   Le type de v est $\times$ [A: D, B: D].

2. La formule F_2 construit l'ensemble R_2 des parties de R_1:

   $F_2 = \{ x \mid \forall z (z \in x \Rightarrow z \in F_1) \}$.

   Le type de x est $\{ \times$ [A: D, B: D] $\}$.
   R_2 est un ensemble de relations.

3. La formule F_3 calcule l'ensemble R_3 des élts de R_2 contenant R :

   $F_3 = \{ x \mid x \in F_2 \wedge (\forall z) (R(z) \Rightarrow z \in x) \}$

   Le type de x est $\{ \times$ [A: D, B: D] $\}$.

4. La formule F_4 calcule l'ensemble R_4 des élts de R_3 fermés transitivement :

$$\begin{aligned} F_4 = \{\ x \mid x \in F_3 \wedge (\forall\ u, v\ (\ (\ u \in x \wedge v \in x \wedge \\ & u.B = v.A\ ) \Rightarrow \\ & \exists\ z\ (\ z \in x \wedge \\ & \qquad z.A{=}u.A \wedge \\ & \qquad z.B{=}v.B\ )\ )\ )\ \} \end{aligned}$$

La variable x est de type $\{\ \times\ [\ A{:}\ D,\ B{:}\ D\ ]\ \}$.

5. La formule F calcule la fermeture transitive de R en faisant l'intersection des éléments de R_4 :

$$F = \{\ v \mid \forall\ x\ (\ x \in F_4 \Rightarrow v \in x\ )\ \}$$

La variable v est de type $\times\ [\ A{:}\ D,\ B{:}\ D\ ]$.

5 Données semi-structurées

Dans cette partie nous allons présenter et discuter un certain nombre de problèmes relatifs à l'usage du Web et à l'interaction entre bases de données et Web.

Qu'est-ce fait que le Web est différent d'une base de données classique ? A peu près tout : une bd classique est un système conçu de manière cohérente. Le systme impose des structures "rigides" et fournit, dans un environnement maîtrisé des processus d'interrogation (requêtes), de mise à jour, de transactions, concurrence, reprise ...

Le Web échappe à cette caractérisation. C'est un ensemble de sources de données (informations) en constante évolution, de formes et de natures très diverses, et avec lesquelles l'utilisateur interagit. Il est impossible d'acquerir / maintenir une image consistante de l'intégralité du web. En l'absence de sources de données bien définies, il reste à trouver un sens à `interroger le web`.

La voix consistant à voir le web comme une grande base de données n'est pas très prometeuse (même si d'intéressants problèmes se profilent). Les échanges sur le web sont souvent effectués au sein d'environnement contrôllés. Beaucoup de sites Web sont ni plus ni moins que des bases de données classiques utilisant des wrappers XML. Certains sites intègrent des vues d'un ensemble (statiquement ou dynamiquement spécifié) de sources de données distribuées. Certains sites du web hebergent des serveurs dont l'obejctif est d'étendre les moteurs de recherche classique afin de répondre le mieux possible à des requêtes sur le web ou sur une partie XML-isé du web, en utilisant une approche centralisé (entrepôt) ou non.

Données semi-structurées Données semi-structurées = graphe étiqueté

pas de distinction entre schéma et données
Quelques modèles :

- Object Exchange Model (OEM) introduit par le projet Tsimmis pour l'integration de sources de données
- LORE
- UnQL (Un. of Pennsylvania) inspiré par OEM et par ACeDB (pour la biologie)
- XML (Extended Markup language) - communauté "document" - un sous-ensemble de SGML, une extension de HTML