Bases de Données : Forme normale de Boyce-Codd (BCNF)

Stéphane Devismes

Université Grenoble Alpes

26 août 2020

Plan

Rappel (1/2)

Pour tout ensemble d'attributs A de R on a

$A^+ = S$, où S est l'ensemble de tous les attributs de la relation R

si et seulement si

A contient une clé, autrement dit A est superclé

Rappel (2/2)

Pour tout ensemble d'attributs A de R on a

$$A^+ = A$$

si et seulement si

les seuls attributs déterminés par A sont ceux qui font partie de A.

Définition

Une relation R est en forme normale de Boyce-Codd (BCNF) si pour tout ensemble d'attributs A de R on a

$A^+ = A$ ou $A^+ = S$, où S est l'ensemble de tous les attributs de la relation R.

Cela signifie que **s'il y a une dépendance $A \to b$ avec $b \notin A$, alors A est une superclé.**

Propriété

Une relation R est en BCNF si et seulement si pour tous ensembles d'attributs A et B de R, on a

$$A \to B$$

si et seulement si :

- ou bien $B \subseteq A$ (dépendance triviale),
- ou bien A contient une clé (*i.e.*, A est une superclé) de R.

Preuve de la propriété

Soit R une relation BCNF. Soient A et B deux ensembles d'attributs de R.

Supposons $A \to B$. D'après la définition, on a $A^+ = A$ ou $A^+ = S$.
Si $A^+ = A$, alors on a nécessairement $B \subseteq A$ (d'après lalgorithme de clôture).
Si $A^+ = S$, alors, par définition, A est une superclé de R, donc A contient une clé de R.

Supposons $B \subseteq A$ ou A contient une clé de R.
Dans le premier cas, on a $A \to B$ par réflexivité.
Dans le second cas, on $A^+ = S$ et donc $A \to S$. Or par réflexivité, $S \to B$ et, par transitivité, $A \to B$.

Preuve de la propriété (suite)

Soit R une relation telle que pour tous ensembles d'attributs A et B de R, on a

$$A \to B$$

si et seulement si :

- ou bien $B \subseteq A$ (dépendance triviale),
- ou bien A contient une clé (*i.e.*, A est une superclé) de R.

Soit C un ensemble d'attributs.
Si $C^+ \neq C$ alors, d'après l'algorithme de clôture, il existe une dépendance fonctionnelle $A \to B$ telle que $A \subseteq C$ et $B \not\subseteq A$.
Donc A **contient une clé**, c'est-à-dire, $A^+ = S$.
De $A \subseteq C$ et $A^+ = S$, on a $C^+ = S$ (la clôture préserve les inclusions)
Ainsi, $C^+ = C$ ou $C^+ = S$: R est BCNF.

Exemple

Soit $R(a,b)$ avec la base de dépendances fonctionnelles $a \to b$, $b \to a$.

Alors R a deux clés (a et b).

Soit $R(a,b,c)$ avec la base de dépendances fonctionnelles $a \to b$, $b \to c$.

Propriété

Si une relation R est en BCNF alors dès qu'on connaît les clés de R, on peut en déduire toutes ses dépendances.

En effet, étant donné un ensemble d'attributs A de R :

- soit A contient une clé et alors $A^+ = S$, où S est l'ensemble de tous les attributs de la relation R.
- soit A ne contient pas de clé et alors $A^+ = A$.

Décomposition en BCNF

Une décomposition $R = R[S_1] * \ldots * R[S_n]$ est en forme normale de Boyce-Codd (BCNF) si chaque $R[S_i]$ est en BCNF.

Algorithme récursif de normalisation BCNF

Entrée. Une relation R, l'ensemble S des attributs de R et une base $\mathcal{D}$ de dépendances fonctionnelles singletons pour R.

Sortie. $Res = \{S_1, \ldots, S_n\}$ où les S_i fournissent une décomposition de R en BCNF qui évite les redondances (mais ne préserve pas toujours les dépendances).

-T- Pour chaque dépendance fonctionnelle $A \to b$ de $\mathcal{D}$ **par ordre croissant de cardinal de partie gauche**

Si A **vérifie** $A^+ \subsetneq S$ alors **passer à (B)**.

- RET0 - Si le test précédent est toujours faux, alors

R est en BCNF, poser $Res = \{S\}$ et retourner Res.

-B- Soit A avec $A^+ \subsetneq S$.

Poser $S_1 = A^+$ et $S_2 = A \cup (S - A^+)$.

Poser $R_1 = R[S_1]$ et $\mathcal{D}_1 =$ le sous-ensemble de $\mathcal{D}$ formé des dépendances fonctionnelles dont tous les attributs sont dans S_1.

Appel récursif avec R_1, S_1 et $\mathcal{D}_1$. Soit Res_1 le résultat.

Poser $R_2 = R[S_2]$ et $\mathcal{D}_2 =$ le sous-ensemble de $\mathcal{D}$ formé des dépendances fonctionnelles dont tous les attributs sont dans S_2.

Appel récursif avec R_2, S_2 et $\mathcal{D}_2$. Soit Res_2 le résultat.

-RET- Poser $Res = Res_1 \cup Res_2$ et retourner Res.

Exemples (1/4)

Soit $R(l, o, s, b)$ avec la base $\{l \to o, l \to s, s \to b\}$.

$l^+ = losb = S$, mais $s^+ = sb \subsetneq S$, donc :

On pose $A = s$, $S_1 = A^+ = sb$ et $S_2 = A \cup (S - A^+) = los$.

1. Appel récursif sur $R_1 = R[sb]$ avec $\{s \to b\}$: $s^+ = sb$, donc R_1 est BCNF de clé s, stop.
2. Appel récursif sur $R_2 = R[los]$ avec $\{l \to o, l \to s\}$: $l^+ = os$, donc R_2 est BCNF de clé l, stop.

Donc $R_1 * R_2$, où $\{s \to b\}$ est la base de $R_1 = R[\underline{s}b]$ et $\{l \to o, l \to s\}$ est la base de $R_2 = R[\underline{l}os]$, est une décomposition BCNF sans perte de dépendances fonctionnelles.

Définition et propriétés | Normalisation BCNF : algorithme récursif | Normalisation BCNF : algorithme de synthèse

Exemples (2/4)

Soit $R(l, o, s, b)$ avec la base $\{l \to o, l \to s, s \to b, o \to b\}$.

$s^+ = sb \subsetneq S$, donc :

On pose $A = s$, $S_1 = A^+ = sb$ et $S_2 = A \cup (S - A^+) = los$.

1. Appel récursif sur $R_1 = R[sb]$ avec $\{s \to b\}$: $s^+ = sb$, donc R_1 est BCNF de clé s, stop.
2. Appel récursif sur $R_2 = R[los]$ avec $\{l \to o, l \to s\}$: $l^+ = los$, donc R_2 est BCNF de clé l, stop.

Donc $R_1 * R_2$, où $\{s \to b\}$ est la base de $R_1 = R[\underline{s}b]$ et $\{l \to o, l \to s\}$ est la base de $R_2 = R[\underline{l}os]$, est une décomposition BCNF, mais elle ne préserve pas les dépendances : on a « perdu » $o \to b$.

Définition et propriétés | Normalisation BCNF : algorithme récursif | Normalisation BCNF : algorithme de synthèse

Exemples (3/4)

Soit $R(l, o, s, b)$ avec la base $\{l \to o, l \to s, s \to b, o \to b\}$.

$o^+ = ob \subsetneq S$, donc :

On pose $A = o$, $S_1 = A^+ = ob$ et $S_2 = A \cup (S - A^+) = los$.

1. Appel récursif sur $R_1 = R[ob]$ avec $\{o \to b\}$: $o^+ = ob$, donc R_1 est BCNF de clé o, stop.
2. Appel récursif sur $R_2 = R[los]$ avec $\{l \to o, l \to s\}$: $l^+ = los$, donc R_2 est BCNF de clé l, stop.

Donc $R_1 * R_2$, où $\{o \to b\}$ est la base de $R_1 = R[\underline{o}b]$ et $\{l \to o, l \to s\}$ est la base de $R_2 = R[\underline{l}os]$, est une décomposition BCNF, mais elle ne préserve pas les dépendances : on a « perdu » $s \to b$.

Définition et propriétés | Normalisation BCNF : algorithme récursif | Normalisation BCNF : algorithme de synthèse

Exemples (4/4)

Soit $R(p, m, e)$ avec la base $\{p \to m, me \to p\}$.

$p^+ = pm \subsetneq S$, donc :

On pose $A = p$, $S_1 = A^+ = pm$ et $S_2 = A \cup (S - A^+) = pe$.

1. Appel récursif sur $R_1 = R[pm]$ avec $\{p \to m\}$: $p^+ = pm$, donc R_1 est BCNF de clé p, stop.
2. Appel récursif sur $R_2 = R[pe]$ avec $\emptyset$: R_2 est BCNF de clé pe, stop.

Donc $R_1 * R_2$, où $\{p \to m\}$ est la base de $R_1 = R[\underline{p}m]$ et la base de $R_2 = R[\underline{pe}]$ est vide, est une décomposition BCNF, mais elle ne préserve pas les dépendances : on a « perdu » $me \to p$.

Preuve de l'algorithme : rappels

La clôture préserve les inclusions : si $X \subseteq Y$ alors $X^+ \subseteq Y^+$.

Théorème de décomposition : soit A un ensemble d'attributs.

Soient $R_1 = R[A^+]$ et $R_2 = R[A \cup (S - A^+)]$, alors :

- Les attributs communs à R_1 et R_2 sont exactement les attributs de A.
- $R[A] = R_1[A] = R_2[A]$.
- A est une superclé pour R_1.
- $R_1 = R$ si seulement si $A^+ = S$.
- $R_2 = R$ si seulement si $A^+ = A$.

De plus, $R = R_1 * R_2$.

Preuve de l'algorithme : terminaison

En cas de récursion (-B-), les appels se font avec S_1 **et** S_2 qui **sont strictement inclus dans** S

En effet dans ce cas, on a $A^+ \subsetneq S$ (par hypothèse) et $A \subsetneq A^+$ (car $A \to b$ est une dépendance singleton).

Et donc S_1 et S_2 sont strictement inclus dans S, d'après le théorème de décomposition.

Preuve de l'algorithme : correction (1/6)

But : On doit prouver que l'algorithme calcule une décomposition de R qui est BCNF et qui évite les redondances.

Nous démontrons ce résultat par une récurrence structurelle.

Preuve de l'algorithme : correction (2/6)

Supposons que **le test en T est faux** : $\forall A \to b \in \mathcal{D}, A^+ = S$. Soit C un ensemble d'attributs de R.

- S'il existe $A \to b$ dans $\mathcal{D}$ avec $A \subseteq C$, alors $A^+ \subseteq C^+$, puisque la clôture préserve les inclusions, donc $C^+ = S$.
- Sinon, $C^+ = C$, d'après l'algorithme de clôture.

Ainsi R est BCNF. Or, $Res = \{S\}$ et $R[S] = R$.

Donc, on obtient une décomposition BCNF de R.

Bien sûr cette décomposition évite les redondances.

Preuve de l'algorithme : correction (3/6)

Supposons maintenant que **le test en T est vrai pour une dépendance** $A \to b$.

On a alors en entrée de (B) un ensemble d'attributs A tel que $A^+ \not\subseteq S$, et en sortie $Res = Res_1 \cup Res_2$.

Posons $Res_1 = \{S_1^1 \dots S_1^n\}$ et $Res_2 = \{S_2^1 \dots S_2^m\}$.

Par hypothèse de récurrence :

- Les relations $R[S_1^1], \dots, R[S_1^n]$ sont toutes BCNF, $R_1 = R[S_1^1] * \dots * R[S_1^n]$, et $\forall i,j \in \{1 \dots n\}, (i \neq j) \Rightarrow (S_1^i \not\subseteq S_1^j)$.
- Les relations $R[S_2^1], \dots, R[S_2^m]$ sont toutes BCNF, $R_2 = R[S_2^1] * \dots * R[S_2^m]$ et $\forall i,j \in \{1 \dots m\}, (i \neq j) \Rightarrow (S_2^i \not\subseteq S_2^j)$.

Montrons d'abord qu'on obtient une décomposition BCNF de R.

- On a $Res = Res_1 \cup Res_2$.
  Donc, par l'hypothèse de récurrence, $\forall X \in Res$, $R[X]$ est BCNF.
- De plus, d'après le théorème de décomposition, on a $R = R_1 * R_2 = R[S_1^1] * \dots * R[S_1^n] * R[S_2^1] * \dots * R[S_2^m]$.

Par conséquent, on obtient bien une décomposition BCNF de R.

Preuve de l'algorithme : correction (4/6)

Montrons maintenant que cette décomposition évite les redondances.

D'après l'hypothèse de récurrence, il reste à montrer que quels que soient $S_1^i \in Res_1$ et $S_2^j \in Res_2$, on a $S_1^i \not\subseteq S_2^j$ et $S_2^j \not\subseteq S_1^i$, ce qui équivaut à $S_1^i \cap S_2^j \neq S_1^i$ et $S_1^i \cap S_2^j \neq S_2^j$.

Pour cela, on va montrer

- d'une part que $S_1^i \cap S_2^j \subseteq A$ et
- d'autre part que $S_1^i \not\subseteq A$ et $S_2^j \not\subseteq A$.

Preuve de l'algorithme : correction (5/6)

Montrons que $S_1^i \cap S_2^j \subseteq A$

Puisque $S_1^i \subseteq S_1$ et $S_2^j \subseteq S_2$, on a $S_1^i \cap S_2^j \subseteq S_1 \cap S_2$.

Or, d'après le théorème de décomposition $S_1 \cap S_2 = A$.

D'où $S_1^i \cap S_2^j \subseteq A$.

Preuve de l'algorithme : correction (6/6)

Montrons que $S_1^i \not\subseteq A$
(la preuve de $S_2^j \not\subseteq A$ est analogue)

L'ensemble d'attributs S_1^i est obtenu lors d'un appel récursif avec R', S' et $\mathcal{D}'$, suite à la découverte d'une dépendance singleton $A' \to b'$ dans $\mathcal{D}'$, telle que $A'^+ \subsetneq S'$ (on peut avoir $A' = A$).

D'après le théorème de décomposition, on a $A' \subseteq S_1^i$.

De plus, d'une part $A' \subsetneq A'^+$ car $b \notin A'$ ($A' \to b'$ est singleton) et d'autre part, $S' - A'^+ \neq \emptyset$ ($A'^+ \subsetneq S'$)

D'où, $A' \subsetneq S_1^i$.

Puisque les dépendances fonctionnelles sont choisies en respectant l'ordre croissant de cardinal du membre gauche, on a $|A'| \geq |A|$, et donc, soit $A' = A$ soit $A' \not\subseteq A$.

- Si $A' = A$, alors $A' \subsetneq S_1^i$ signifie $A \subsetneq S_1^i$, donc $S_1^i \not\subseteq A$
- Si $A' \not\subseteq A$, alors $S_1^i \not\subseteq A$ puisque $A' \subsetneq S_1^i$.

Donc, dans tous les cas, on a $S_1^i \not\subseteq A$.

Principe

L'algorithme de synthèse calcule une **décomposition BCNF avec préservation des dépendances.**

Mais en général il retourne **un résultat qui comporte des redondances**.

Il est fondé sur la notion de base minimale.

Base minimale

Une base de dépendances fonctionnelles $\mathcal{D}$ est minimale si elle est formée de **dépendances fonctionnelles singletons** et si :

- Aucune dépendance fonctionnelle de $\mathcal{D}$ n'est conséquence des autres dépendances fonctionnelles de $\mathcal{D}$.
- Pour chaque dépendance fonctionnelle $a_1 \ldots a_n \to b$ de $\mathcal{D}$ (avec $n \geq 2$) et chaque $i \in \{1, \ldots, n\}$, la dépendance fonctionnelle $a_1 \ldots \cancel{a_i} \ldots a_n \to b$ n'est pas conséquence de $\mathcal{D}$

  (on dit alors que la dépendance fonctionnelle $a_1 \ldots a_n \to b$ de $\mathcal{D}$ est élémentaire).

Algorithme de détermination d'une base minimale

Entrée. Une base $\mathcal{D}$ de dépendances fonctionnelles singletons.

Sortie. Une base $\mathcal{D}_{min}$ minimale de dépendances fonctionnelles.

Principe. « Nettoyer » $\mathcal{D}$.

-I- Initialiser $\mathcal{D}_{min}$ à $\mathcal{D}$.

-B- Exécuter B1 et B2, dans n'importe quel ordre, jusqu'à ce que $\mathcal{D}_{min}$ ne puisse plus être modifiée :

-B1- **Si** $a_1 \ldots a_n \to b$ dans $\mathcal{D}_{min}$ est conséquence des autres dépendances fonctionnelles de $\mathcal{D}_{min}$,
alors enlever $a_1 \ldots a_n \to b$ de $\mathcal{D}_{min}$.

-B2- **Si** $a_1 \ldots \cancel{a_i} \ldots a_n \to b$ est conséquence de $\mathcal{D}_{min}$,
pour un $a_1 \ldots a_n \to b$ dans $\mathcal{D}_{min}$ (avec $n \geq 2$) et un i entre 1 et n,
alors remplacer $a_1 \ldots a_n \to b$ par $a_1 \ldots \cancel{a_i} \ldots a_n \to b$ dans $\mathcal{D}_{min}$.

-RET- Retourner $\mathcal{D}_{min}$.

B1 : détail

Pour tester si $A \to b$ dans $\mathcal{D}_{min}$ est **conséquence** des autres dépendances fonctionnelles de $\mathcal{D}_{min}$:

On note $\mathcal{D}'$ l'ensemble de dépendances fonctionnelles obtenu **en retirant** $A \to b$ **de** $\mathcal{D}_{min}$.

On calcule A^+ **(la clôture de** A**) relativement à** $\mathcal{D}'$**.**

La réponse est oui si et seulement si $b \in A^+$.

Si la réponse est oui, alors **on retire** $A \to b$ **de** $\mathcal{D}_{min}$.

B2 : détail

Pour tester si $A \to b$, où $A = a_1 \ldots \cancel{a_i} \ldots a_n$, est conséquence de $\mathcal{D}_{min}$:

On calcule A^+ **(la clôture de A) relativement à** $\mathcal{D}_{min}$.

La réponse est oui si et seulement si $b \in A^+$.

Si la réponse est oui, alors **on remplace** $a_1 \ldots a_n \to b$ **par** $a_1 \ldots \cancel{a_i} \ldots a_n \to b$ **dans** $\mathcal{D}_{min}$.

Exemples (1/3)

Soit $R(a,b,c)$ avec la base de dépendances fonctionnelles $\{ab \to c,\ a \to b\}$.

On ne peut pas appliquer B1. Par exemple, $a^+ = a$ relativement à $\{ab \to c\}$.

On peut appliquer B2 pour réduire $ab \to c$ en $a \to c$ car $a^+ = abc$ relativement à $\{ab \to c,\ a \to b\}$.

Ainsi, on obtient **la base minimale** $\{a \to c,\ a \to b\}$.

Exemples (2/3)

Soit $R(a,b,c,d,e)$ avec la base de dépendances fonctionnelles $\{a \to b,\ a \to d,\ b \to d,\ b \to e,\ de \to c,\ ae \to c\}$.

On peut appliquer B1 : $a \to d$ est conséquence de $\{a \to b,\ b \to d,\ b \to e,\ de \to c,\ ae \to c\}$. En effet, $a^+ = abdec$ relativement à $\{a \to b,\ b \to d,\ b \to e,\ de \to c,\ ae \to c\}$. (d'ailleurs, a est une clé !)

On réduit donc la base à $\{a \to b,\ b \to d,\ b \to e,\ de \to c,\ ae \to c\}$.

On peut encore appliquer B1 : $ae \to c$ est conséquence de $\{a \to b,\ b \to d,\ b \to e,\ de \to c\}$. En effet, $ae^+ = abdec$ relativement à $\{a \to b,\ b \to d,\ b \to e,\ de \to c\}$ (d'ailleurs, ae est une clé !)

On réduit donc la base à $\{a \to b,\ b \to d,\ b \to e,\ de \to c\}$.

On ne peut plus appliquer B1

On ne peut pas appliquer B2

Ainsi, on obtient **la base minimale** $\{a \to b,\ b \to d,\ b \to e,\ de \to c\}$.

Exemples : ADE (3/3)

Soit la relation $R(c,p,h,s,e,n)$ où les attributs signifient :

c	Cours
p	Professeur
h	Heure
s	Salle
e	Etudiant
n	Note

avec une base de dépendances fonctionnelles :

$c \to p$	Chaque cours a un professeur
$hs \to c$	Il y a (au plus) un cours pour une heure et une salle donnés
$hp \to cs$	Il y a (au plus) un cours et (au plus) une salle pour une heure et un professeur donnés
$he \to cs$	Il y a (au plus) un cours et (au plus) une salle pour une heure et un étudiant donnés
$ce \to n$	Un étudiant n'a qu'une note par cours.

Exemples : ADE (3/3)

On considère la base de dépendances fonctionnelles singletons équivalentes :
$\mathcal{D}_{min} = \{c \to p, hs \to c, hp \to c, he \to c, hp \to s, he \to s, ce \to n\}$

On peut appliquer B1 :

- On peut enlever $hp \to c$ car $hp^+ = hpsc$ relativement à $\mathcal{D}_{min} - \{hp \to c\}$. Donc, $\mathcal{D}_{min}$ devient $\{c \to p, hs \to c, he \to c, hp \to s, he \to s, ce \to n\}$.
- On peut ensuite enlever $he \to c$ car $he^+ = hescpn$ relativement à $\mathcal{D}_{min} - \{he \to c\}$. Donc, $\mathcal{D}_{min}$ devient $\{c \to p, hs \to c, hp \to s, he \to s, ce \to n\}$.

On ne peut plus appliquer B1

On ne peut pas appliquer B2

Ainsi, on obtient **la base minimale**
$\{c \to p, hs \to c, hp \to s, he \to s, ce \to n\}$.

Preuve de l'algorithme : terminaison

La somme du nombre d'attributs dans les membres gauches des dépendances fonctionnelles décroît strictement à chaque pas.

Preuve de l'algorithme : correction B1

Considérons un pas de B1 qui modifie $\mathcal{D}_{min}$.

Notons $\mathcal{D}_A$ et $\mathcal{D}_B$ **les deux valeurs successives de l'ensemble courant** $\mathcal{D}_{min}$: on passe de $\mathcal{D}_A$ à $\mathcal{D}_B$

Soit $A \to b$ la dépendance fonctionnelle supprimée lors du pas.

Par définition, $\mathcal{D}_B \subsetneq \mathcal{D}_A$. Donc, $\mathcal{D}_A \implies \mathcal{D}_B$.

D'après l'algorithme, $\mathcal{D}_B \implies \{A \to b\}$. Donc $\mathcal{D}_B \implies \mathcal{D}_B \cup \{A \to b\} = \mathcal{D}_A$. D'où, $\mathcal{D}_B \implies \mathcal{D}_A$.

Ainsi, $\mathcal{D}_A \iff \mathcal{D}_B$.

Preuve de l'algorithme : correction B2

Considérons un pas de B2 qui modifie $\mathcal{D}_{min}$.

Notons $A = a_1 \ldots a_n$ et $B = a_1 \ldots \cancel{a_i} \ldots a_n$, notons $\mathcal{D}_A$ et $\mathcal{D}_B$ **les deux valeurs successives de l'ensemble courant** $\mathcal{D}_{min}$: on passe de $\mathcal{D}_A$ à $\mathcal{D}_B$ en remplaçant $A \to b$ par $B \to b$.

Autrement dit, en notant $\mathcal{D}'$ **l'ensemble** $\mathcal{D}_A$ **privé de** $A \to b$, on a $\mathcal{D}_A = \mathcal{D}' \cup \{A \to b\}$ et $\mathcal{D}_B = \mathcal{D}' \cup \{B \to b\}$.

Donc, $\mathcal{D}_A \implies \mathcal{D}'$ et $\mathcal{D}_B \implies \mathcal{D}'$.

- $B \to b$ est conséquence de $\mathcal{D}_A$ **par hypothèse** et $\mathcal{D}_A \implies \mathcal{D}'$, donc $\mathcal{D}_A \implies \mathcal{D}_B$.
- $A \to b$ est conséquence de $B \to b$ car $B \subseteq A$, donc, par réfléxivité $A \to B$ et par transitivité, on a $A \to B$ et $B \to b$ qui donne $A \to b$.
  Donc $A \to b$ est conséquence de $\mathcal{D}_B$ et $\mathcal{D}_B \implies \mathcal{D}'$ donnent $\mathcal{D}_B \implies \mathcal{D}_A$.

Ainsi, $\mathcal{D}_A \iff \mathcal{D}_B$.

Algorithme de synthèse BCNF

Entrée. Une base **minimale** $\mathcal{D}_{min}$ de R.

($\mathcal{D}_{min}$ peut être obtenue par l'algorithme de détermination d'un ensemble de dépendances fonctionnelles minimal).

Sortie. $Res = \{S_1, ..., S_n\}$ où les S_i fournissent des relations formant une décomposition de R en BCNF qui **préserve les dépendances** (mais **n'évite pas toujours les redondances**).

-I- Poser $Res = \emptyset$

-U- (UNION) Pour chaque partie gauche A d'une dépendance fonctionelle de $\mathcal{D}_{min}$:
- Soit B **l'ensemble de tous les attributs** b **tels que** $A \rightarrow b$ est dans $\mathcal{D}_{min}$.
- Ajouter $A \cup B$ dans Res.
- Cela donne une relation $R[A \cup B]$ avec les dépendances $\{A \rightarrow b : b \in B\}$.

-F- Si aucun des ensembles d'attributs dans Res **n'est une superclé de** R, alors :
- Déterminer une clé K de R et ajouter K dans Res.
- Cela donne une relation $R[K]$ sans dépendance fonctionnelle.

-RET- Retourner Res.

Algorithme de synthèse BCNF (suite)

Détails. Pour la partie F, si on ne connaît pas les clés de R, on peut procéder comme suit :

Calculer la clôture de chacun des S_i dans Res.

S'il y a un i **tel que** $S_i^+ = S$, **alors** la décomposition est terminée.

Sinon, calculer un clé de R en utilisant l'algorithme de calcul des clôtures d'attributs et en s'arrêtant dès que la première clé est obtenue.

Exemples (1/6)

Soit $R(a, b)$ avec la base $\{a \rightarrow b, b \rightarrow a\}$.

On a déjà vu que R est en BCNF : R a deux clés (a) et (b), et la base donnée est minimale.

L'algorithme de synthèse fournit la décomposition BCNF $R = R[\underline{a}, b] * R[\underline{b}, a]$, où $\{a \rightarrow b\}$ est la base de $R[\underline{a}, b]$ et $\{b \rightarrow a\}$ est la base de $R[\underline{b}, a]$.
(*N.b.*, il n'est pas nécessaire d'ajouter une autre relation car ab est superclé de R.)

Cette décomposition bien sûr est redondante : $R[\underline{a}, b] = R[\underline{b}, a]$.

Exemples (2/6)

Soit $R(l, o, s, b)$ avec la base $\{l \rightarrow os, s \rightarrow b\}$ ou $\{l \rightarrow o, l \rightarrow s, s \rightarrow b\}$.

Exemples (3/6)

Soit $R(l, o, s, b)$ avec la base $\{l \to os,\ s \to b,\ o \to b\}$ ou $\{l \to o,\ l \to s,\ s \to b,\ o \to b\}$.

La base donnée est minimale donc l'algorithme de synthèse donne la décomposition $R = R[\underline{l}, o, s] * R[\underline{s}, b] * R[\underline{o}, b]$, où $\{l \to o, l \to s\}$ est la base de $R[\underline{l}, o, s]$, $\{s \to b\}$ est la base de $R[\underline{s}, b]$, et $\{o \to b\}$ est la base de $R[\underline{o}, b]$.
(*N.b.*, il n'est pas nécessaire d'ajouter une autre relation car *los* est superclé de *R*.)

Cette décomposition n'est pas redondante.

Rappel : l'algorithme récursif BCNF donne $R = R[\underline{l}, o, s] * R[\underline{s}, b]$ ou $R = R[\underline{l}, o, s] * R[\underline{o}, b]$, qui ne préservent pas les dépendances (soit $o \to b$, soit $s \to b$ est perdue).

Exemples (4/6)

Soit $R(p, m, e)$ avec la base $\{p \to m,\ me \to p\}$.

La base donnée est minimale donc l'algorithme de synthèse donne la décomposition $R = R[\underline{p}, m] * R[\underline{m, e}, p]$, où $\{p \to m\}$ est la base de $R[\underline{p}, m]$ et $\{me \to p\}$ est la base de $R[\underline{m, e}, p]$.
(*N.b.*, il n'est pas nécessaire d'ajouter une autre relation car *mep* est superclé de *R*.)

Cette décomposition est redondante car $\{pm\} \subseteq \{mep\}$: en fait la seconde relation est exactement R.

Rappel : l'algorithme récursif BCNF donne $R = R[\underline{p}, m] * R[\underline{p, e}]$, qui ne préserve pas la dépendance $me \to p$.

Exemples (5/6)

Soit $R(c, p, h, s, e, n)$ avec la base minimale $\{c \to p,\ hs \to c,\ hp \to s,\ he \to s,\ ce \to n\}$ calculée précédemment.

L'algorithme de synthèse donne la décomposition $R = R[\underline{c}, p] * R[\underline{h, s}, c] * R[\underline{h, p}, s] * R[\underline{h, e}, s] * R[\underline{c, e}, n]$, où

- $\{c \to p\}$ est la base de $R[\underline{c}, p]$,
- $\{hs \to c\}$ est la base de $R[\underline{h, s}, c]$,
- $\{hp \to s\}$ est la base de $R[\underline{h, p}, s]$,
- $\{he \to s\}$ est la base de $R[\underline{h, e}, s]$ et
- $\{ce \to n\}$ est la base de $R[\underline{c, e}, n]$.

(*N.b.*, il n'est pas nécessaire d'ajouter une autre relation car *hes* est superclé de *R*.)

Cette décomposition évite les redondances.

Exemples (6/6)

Soit $R(p, h, s, e)$ avec la base minimale $\{hp \to s\}$.

Bien sûr, cette base est minimale. L'algorithme de synthèse fournit d'abord $R[\underline{h, p}, s]$, où $\{hp \to s\}$ est la base de $R[\underline{h, p}, s]$.

Et comme aucune clé de *R* n'est dans *phs*, on doit ajouter une relation.

L'unique clé de *R* est *phe*.

Donc on obtient la décomposition BCNF préservant les dépendances : $R = R[\underline{h, p}, s] * R[\underline{p, h, e}]$, où $\{hp \to s\}$ est la base de $R[\underline{h, p}, s]$ et $\emptyset$ est la base de $R[\underline{p, h, e}]$.

Cette décomposition évite les redondances.

Preuve de l'algorithme : terminaison

-I- : temps constant.

-U- : Polynomial en $|\mathcal{D}_{min}| * |S|$.

-F- : Il y a $2^{|S|} - 1$ sous-ensembles non-vides de S. Pour déterminer une clé ou l'absence de superclé, il faut calculer les clôtures de toute ou partie de ces sous-ensembles, chaque clôture a un coût polynomial en $|\mathcal{D}_{min}| * |S|$. D'où, une complexité finie mais exponentielle ($\leq e^{O(|\mathcal{D}_{min}| * |S|)}$).

-RET- : temps constant.

Preuve de l'algorithme : correction (1/3)

Rappel : on pose $Res = \{S_1, \ldots, S_n\}$ à la fin de l'algorithme.

Notons $\mathcal{D}^{S_i}_{min} \subseteq \mathcal{D}_{min}$ l'ensemble des dépendances fonctionnelles de $R[S_i]$.

À la fin de U, on a $Res = \{S_1, \ldots, S_x\}$ avec $x \in \{n-1, n\}$.

Tout d'abord, par construction, il n'y pas de perte de dépendances fonctionnelles : $\forall A \to b \in \mathcal{D}_{min}$, $\exists i \in \{1, \ldots, x\}$ tel que $A \cup \{b\} \in S_i$.

Ensuite, tout élément de $S - (S_1 \cup \ldots \cup S_x)$ n'apparait dans aucune dépendance fonctionnelle de $\mathcal{D}_{min}$, donc appartient à toute clé de R et donc à toute superclé de R.
Donc, par F, $S - (S_1 \cup \ldots \cup S_n)$ est nécessairement vide.
Ainsi, on peut conclure que $S_1 \cup \ldots \cup S_n = S$, donc $R \subseteq R[S_1] * \ldots * R[S_n]$.

Preuve de l'algorithme : correction (2/3)

Toujours d'après F, il existe $S_i \in Res$ tel que S_i est une superclé de R.

Or, $R[S_1] * \ldots * R[S_n]$ et R ont les mêmes dépendances fonctionnelles (il n'y a pas de pertes) et les mêmes attributs ($S_1 \cup \ldots \cup S_n = S$).

Donc S_i est une superclé de $R[S_1] * \ldots * R[S_n]$. Donc, $|R[S_1] * \ldots * R[S_n]| = |R[S_i]| \leq |R|$.

Puisque $R \subseteq R[S_1] * \ldots * R[S_n]$ et $|R[S_1] * \ldots * R[S_n]| \leq |R|$, on a $R = R[S_1] * \ldots * R[S_n]$, c'est-à-dire, $R[S_1], \ldots, R[S_n]$ est une décomposition de R.

Preuve de l'algorithme : correction (3/3)

Par construction, **après U mais avant F**, $\forall i \in \{1, \ldots, x\}$, $R[S_i]$ a au moins une dépendance fonctionnelle et pour la dépendance fonctionnelle $A \to b$ de $R[S_i]$, A est superclé de $R[S_i]$.

Pour ensemble d'attributs X de $R[S_i]$, **si aucun sous-ensemble de X n'est partie gauche de la dépendance fonctionnelle de** $R[S_i]$, alors $X^+ = X$ relativement à $\mathcal{D}^{S_i}_{min}$;
sinon $X^+ = S_i$ relativement à $\mathcal{D}^{S_i}_{min}$. (d'après l'algorithme de clôture)

Si F modifie *Res*, on a en plus $R[K]$ sans dépendance fonctionnelle.
Donc pour tout sous-ensemble X de K (c'est-à-dire, tout sous-ensemble d'attributs de $R[K]$), on a $X^+ = X$ relativement à $\mathcal{D}^{K}_{min}$.

Ainsi, $\forall i \in \{1, \ldots, n\}$, $R[S_i]$ est BCNF.