Conservatoire National des Arts et Métiers

292 Rue St Martin 75141 Paris Cedex 03

Informatique - CNAM, Paris

Bases de données relationnelles
NFP 107 et NFP 107J

Exercices dirigés

C. Crochepeyre, M. Ferecatu, P. Rigaux, V. Thion et N. Travers

Solutions

2 mai 2011

Chapitre 9

Concurrence

9.1 Sérialisabilité et recouvrabilité

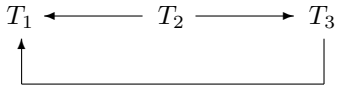
9.1.1 Graphe de sérialisation et équivalence des exécutions

Construisez les graphes de sérialisation pour les trois exécutions (histoires) suivantes. Indiquez les exécutions sérialisables et vérifiez si parmi les trois exécutions il y a des exécutions équivalentes.

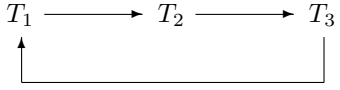
1. $H_1 : w_2[x]\ w_3[z]\ w_2[y]\ c_2\ r_1[x]\ w_1[z]\ c_1\ r_3[y]\ c_3$
2. $H_2 : r_1[x]\ w_2[y]\ r_3[y]\ w_3[z]\ c_3\ w_1[z]\ c_1\ w_2[x]\ c_2$
3. $H_3 : w_3[z]\ w_1[z]\ w_2[y]\ w_2[x]\ c_2\ r_3[y]\ c_3\ r_1[x]\ c_1$

Solution :

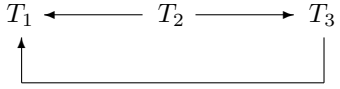
1. **H_1 : les conflits**
   sur x : $w_2[x]\ r_1[x]$; *sur* y : $w_2[y]\ r_3[y]$; *sur* z : $w_3[z]\ w_1[z]$
   le graphe de sérialisation



2. **H_2 : les conflits**
   sur x : $r_1[x]\ w_2[x]$; *sur* y : $w_2[y]\ r_3[y]$; *sur* z : $w_3[z]\ w_1[z]$
   le graphe de sérialisation



3. **H_3 : les conflits**
   sur x : $w_2[x]\ r_1[x]$; *sur* y : $w_2[y]\ r_3[y]$; *sur* z : $w_3[z]\ w_1[z]$
   le graphe de sérialisation



Conclusion *: H_1 et H_3 sont sérialisables (pas de cycle dans le graphe), H_2 ne l'est pas (cycle $T_1T_2T_3$).*

H_1 et H_3 sont toutes les deux équivalentes à l'exécution en série $T_2T_3T_1$. Cela est visible dans le graphe de sérialisation, car les arcs indiquent l'ordre des opérations conflictuelles (d'abord T_2, ensuite T_3, ensuite T_1).

En ce qui concerne l'équivalence d'exécutions, H_2 ne peut être équivalente aux autres, car elle n'est pas sérialisable et les autres le sont. Il reste à vérifier si H_1 est équivalent à H_3.

Il semblerait que H_1 est équivalent à H_3, car on retrouve les mêmes opérations et les mêmes conflits dans le même ordre. Cependant, H_1 et H_3 ne sont pas équivalentes *! Pour avoir équivalence, deux conditions sont nécessaires :* (i) *avoir les mêmes transactions et les mêmes operations, et* (ii) *avoir le meme ordre des operations conflictuelles.*

Ici la seconde condition est remplie, mais pas la premiere ! En effet, si on extrait la transaction T_1 de ces histoires, on remarque que pour H_1 on a $T_1 = r_1[x]w_1[z]c_1$, tandis que pour H_3, $T_1 = w_1[z]r_1[x]c_1$. Et c'est pareil pour T_2.

9.1.2 Recouvrabilité

Parmi les exécutions (histoires) suivantes, lesquelles sont recouvrables, lesquelles évitent les annulations en cascade et lesquelles sont strictes ? Indiquez s'il y a des exécutions sérialisables.

1. $H_1 : r_1[x]\ w_2[y]\ r_1[y]\ w_1[x]\ c_1\ r_2[x]\ w_2[x]\ c_2$

   Solution :
   Généralement, on commence par chercher les écritures dans l'exécution. Ensuite, pour une écriture donnée, on vérifie s'il existe après *celle-ci d'autres lectures/écritures sur la même variable, réalisées* par d'autres transactions*. Une lecture peut indiquer un problème de recouvrabilité / annulation en cascade, une écriture un problème d'exécution stricte.*
   T_1 lit y de T_2 ($r_1[y]$ après $w_2[y]$), mais T_1 se termine avant $T_2 \Longrightarrow H_1$ non-recouvrable
   H_1 n'est pas sérialisable (les conflits $w_2[y]$ - $r_1[y]$ et $w_1[x]$ - $r_2[x]$ forment un cycle)

2. $H_2 : r_1[x]\ w_1[y]\ r_2[y]\ c_1\ w_2[x]\ c_2$

   Solution :
   T_2 lit y de T_1 ($r_2[y]$ après $w_1[y]$) avant que T_1 se termine $\Longrightarrow H_2$ recouvrable, mais n'évite pas les annulations en cascade
   H_2 est sérialisable (les seuls conflits sont $r_1[x]$ - $w_2[x]$ et $w_1[y]$ - $r_2[y]$)

3. $H_3 : r_1[y]\ w_2[x]\ r_2[y]\ w_1[x]\ c_2\ r_1[x]\ c_1$

   Solution :
   T_1 lit x de T_2 ($r_1[x]$ après $w_2[x]$) après la fin de T_2, mais écrit x ($w_1[x]$ après $w_2[x]$) avant la fin de T_2 $\Longrightarrow H_3$ évite les annulations en cascade, mais n'est pas stricte
   H_3 est sérialisable (les seuls conflits sont $w_2[x]$ - $w_1[x]$ et $w_2[x]$ - $r_1[x]$)

9.2 Contrôle de concurrence

9.2.1 Verrouillage à 2 phases

Un ordonnanceur avec verrouillage à 2 phases reçoit la séquence d'opérations ci-dessous.

$H : r_1[x]\ r_2[y]\ w_3[x]\ w_1[y]\ w_1[x]\ w_2[y]\ c_2\ r_3[y]\ r_1[y]\ c_1\ w_3[y]\ c_3$

Indiquez l'ordre d'exécution établi par l'ordonnanceur, en considérant que les opérations bloquées en attente d'un verrou seront exécutées en priorité dès que le verrou devient disponible, dans l'ordre de leur blocage. On suppose que les verrous d'une transaction sont relâchés au moment du Commit.

Solution :

• $r_1[x]$, $r_2[y]$ *exécutées*

• $w_3[x]$ *bloquée à cause de* $r_1[x]$

• $w_1[y]$ *bloquée à cause de* $r_2[y]$

L'opération $w_1[y]$ *qui est bloquée va aussi bloquer tout le reste de la transaction* T_1 *! Donc* $w_1[x]$ *ne peut pas s'exécuter, même si cette opération n'a pas de problème de verrou :*

• $w_1[x]$ *bloquée à cause de* $w_1[y]$

• $w_2[y]$ *exécutée*

• c_2 *relâche les verrous sur* $y \Rightarrow w_1[y]$, $w_1[x]$ *peuvent s'exécuter*

• $r_3[y]$ *bloquée car* T_3 *bloquée à cause de* $w_3[x]$

• $r_1[y]$ *exécutée*

• c_1 *relâche les verrous sur* x, $y \Rightarrow w_3[x]$, $r_3[y]$ *peuvent s'exécuter*

• $w_3[y]$, c_3 *exécutées*

Résultat *:* $r_1[x]\ r_2[y]\ w_2[y]\ c_2\ w_1[y]\ w_1[x]\ r_1[y]\ c_1\ w_3[x]\ r_3[y]\ w_3[y]\ c_3$

9.2.2 Verrouillage, interblocage

On considère l'exécution concurrente suivante :

$H : r_1[x]\ r_2[y]\ r_3[x]\ w_3[x]\ w_1[y]\ r_2[x]\ c_1\ c_2\ c_3$

1. Trouvez les conflits dans H et montrez que H n'est pas sérialisable.

   Solution :
   Conflits : $r_1[x]$-$w_3[x]$, $w_3[x]$-$r_2[x]$, $r_2[y]$-$w_1[y]$
   Le graphe de sérialisation contient un cycle $T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$*, donc H n'est pas sérialisable.*

2. Trouvez l'exécution obtenue à partir de H par verrouillage à deux phases. On considère que les verrous d'une transaction sont relâchés au Commit et qu'à ce moment les opérations bloquées en attente de verrou sont exécutées en priorité.

   Solution :
   • $r_1[x]$ *et* $r_2[y]$ *s'exécutent*
   • $r_3[x]$ *s'exécute (pas de conflit avec* $r_1[x]$*)*
   • $w_3[x]$ *est bloquée par* $r_1[x]$
   • $w_1[y]$ *est bloquée par* $r_2[y]$
   • $r_2[x]$ *s'exécute (pas de conflit avec* $r_1[x]$ *ou avec* $r_3[x]$*)*
   • c_1 *est bloquée, car* T_1 *est bloquée (*$w_1[y]$ *bloquée)*

• *c_2 s'exécute et relâche les verrous de $r_2[y]$ et $r_2[x]$*
⇒ *$w_1[y]$ et c1 sont débloquées et exécutées*
⇒ *c_1 relâche les verrous de T_1*
⇒ *$w_3[x]$ est débloquée et exécutée*
• *c_3 est exécutée*
Résultat *: $r_1[x]$ $r_2[y]$ $r_3[x]$ $r_2[x]$ c_2 $w_1[y]$ c_1 $w_3[x]$ c_3*

3. Le raisonnement suivant mène à une conclusion paradoxale.
   Considérons une histoire quelconque H non-sérialisable, traitée par verrouillage à deux phases. H contient donc un cycle dans son graphe de sérialisation. Chaque arc $T_i \rightarrow T_j$ de ce cycle provient d'une paire d'opérations conflictuelles de type "a_i[x] avant b_j[x]". Quand a_i[x] s'exécute, elle prend le verrou sur x, ce qui bloquera b_j[x].
   Donc T_i bloque T_j, mais comme les arcs forment un cycle, il y aura un blocage circulaire, donc un interblocage. Conclusion : toute histoire non-sérialisable traitée par verrouillage à deux phases produit un interblocage ! Cependant, l'exemple précedent montre que cette conclusion est fausse, car H n'est pas sérialisable et elle ne produit pas d'interblocage. Expliquez pourquoi il n'y a pas d'interblocage et pourquoi le raisonnement ci-dessus est faux.

   Solution :
   Parmi les 3 conflits, $r_1[x]$ bloque $w_3[x]$, $r_2[y]$ bloque $w_1[y]$, mais $w_3[x]$ ne bloque pas $r_2[x]$, car $w_3[x]$ est elle-même bloquée (et donc ne prend pas de verrou).
   Donc dans le couple $a_i[x]$-$b_j[x]$, $a_i[x]$ peut ne pas bloquer $b_j[x]$ si elle-même est déjà bloquée.

4. Montrez que H n'évite pas les annulations en cascade. Montrez qu'en avançant des Commit, H peut éviter les annulations en cascade.

   Solution :
   $r_2[x]$ a lieu après $w_3[x]$ (T_2 lit x de T_3) avant que T_3 se termine, donc H n'évite pas les annulations en cascade. On peut éviter les annulations en cascade en avançant c_3 avant $r_2[x]$, ce qui est possible, car la dernière opération de T_3 est avant $r_2[x]$.

9.2.3 Verrouillage hiérarchique

Considérons une relation **Compte**(*Numéro, Titulaire, Solde*) et trois opérations qui s'exécutent sur cette relation à deux niveaux de granularité : relation et enregistrement. Supposons que dans cette relation il y a deux comptes pour le titulaire "Dupont". Les trois opérations sont les suivantes :
– un programme qui lit tous les comptes ;
– un programme qui réalise un transfert entre les deux comptes de Dupont ;
– un programme qui lit tous les comptes, afin de trouver les comptes de Dupont et de créditer leur solde avec les intérêts de l'année.

1. Montrer que l'exécution suivante est une exécution concurrente des trois opérations ci-dessus. Identifier les données qui correspondent à chaque variable.
   $H : r_1[z]\ r_2[z]\ w_1[x]\ r_3[x]\ w_3[x]\ c_2\ w_1[y]\ r_3[y]\ c_1\ w_3[y]\ c_3$

   Solution :
   Les transactions extraites de cette exécution sont les suivantes :
   T_1 : $r_1[z]$ $w_1[x]$ $w_1[y]$ c_1
   T_2 : $r_2[z]$ c_2
   T_3 : $r_3[x]$ $w_3[x]$ $r_3[y]$ $w_3[y]$ c_3
   T_2 est clairement la lecture de tous les comptes, donc z est la relation Compte.
   T_3 est le transfert entre les comptes de Dupont, donc x et y sont les enregistrements comptes de Dupont.

T_1 est la lecture de tous les comptes (z) suivie de l'écriture des nouveaux soldes dans les comptes x et y (la lecture des comptes x et y est faite pendant la lecture de la relation z).

2. Trouver l'exécution obtenue par verrouillage hiérarchique à partir de H. On considère que le verrouillage hiérarchique utilise des verrous de type SIX (lecture et intention d'écriture), dans le cas présent pour le rajout des intérêts sur les comptes de Dupont.
   Les verrous sont relâchés au Commit et les opérations bloquées en attente de verrou sont traitées en priorité, dans l'ordre de leur blocage.

   Solution :
   $r_1[z]$ prend le verrou SIX(z) (lecture relation z suivi d'écriture x et y) et s'exécute
   $r_2[z]$ bloquée, car le verrou S(z) est en conflit avec SIX(z) $\rightarrow T_2$ bloquée
   $w_1[x]$ prend les verrous IX(z) (pas de conflit avec SIX(z) - même transaction) et X(x) et s'exécute
   $r_3[x]$ bloquée par $w_1[x] \rightarrow T_3$ bloquée
   $w_3[x]$ bloquée, car T_3 bloquée
   c_2 bloquée, car T_2 bloquée
   $w_1[y]$ prend les verrous IX(z) et X(y) et s'exécute
   $r_3[y]$ bloquée, car T_3 bloquée
   c_1 s'exécute et relâche les verrous de T_1
   $\Rightarrow$ *$r_2[z]$ peut prendre S(z) et s'exécuter*
   $\Rightarrow$ *$r_3[x]$ peut prendre IS(z) et S(x) et s'exécuter*
   $\Rightarrow$ *$w_3[x]$ reste bloquée, car ne peut pas obtenir IX(z) à cause de S(z)*
   $\Rightarrow$ *c_2 peut s'exécuter et relâcher les verrous de T_2*
   $\Rightarrow$ *$w_3[x]$ peut maintenant obtenir IX(z) et X(x) et s'exécuter*
   $\Rightarrow$ *$r_3[y]$ peut s'exécuter (T_3 est restée seule)*
   $w_3[y]$ et c_3 s'exécutent
   Résultat *: $r_1[z]\ w_1[x]\ w_1[y]\ c_1\ r_2[z]\ r_3[x]\ c_2\ w_3[x]\ r_3[y]\ w_3[y]\ c_3$*

3. Que se passe-t-il dans le verrouillage hiérarchique de H si on n'utilise pas de verrou SIX au niveau de la relation (mais seulement du S et du IX séparemment) ?

   Solution :
   $r_1[z]$ prend le verrou S(z) et s'exécute
   $r_2[z]$ partage le verrou S(z) avec $r_1[z]$ et s'exécute
   $w_1[x]$ bloquée, car IX(z) est en conflit avec S(z) de $T_2 \rightarrow T_1$ bloquée
   $r_3[x]$ obtient IS(z) et S(x) et s'exécute
   $w_3[x]$ bloquée, car IX(z) est en conflit avec S(z) de T_1,$T_2 \rightarrow T_3$ bloquée
   c_2 s'exécute et relâche les verrous de T_2
   $\Rightarrow$ *$w_1[x]$ reste bloquée à cause de $r_3[x]$ qui a pris S(x)*
   $\Rightarrow$ *$w_3[x]$ reste bloquée à cause de S(z) de T_1*
   Donc T_1 et T_3 restent bloquées, on est dans une situation d'interblocage*. L'une des deux transactions sera annulée et relancée.*

9.3 Reprise après panne

9.3.1 Journalisation

Soit le journal physique ci-dessous, dans lequel on a marqué les opérations Commit et Abort réalisées :

$[T_1, x, 3], [T_2, y, 1], [T_1, z, 2], c_1, [T_2, z, 4], [T_3, x, 2], a_2, [T_4, y, 3], c_3, [T_5, x, 5]$

1. Indiquez le contenu de *liste_commit*, *liste_abort*, *liste_active*.

   Solution :
   liste_commit = $\{T_1, T_3\}$ *;* liste_abort = $\{T_2\}$ *;* liste_active = $\{T_4, T_5\}$ *;*

2. En supposant qu'une nouvelle écriture vient de s'ajouter au journal, lesquelles des écritures suivantes sont compatibles avec une exécution stricte : $[T_5, y, 4]$, $[T_4, z, 3]$ ou $[T_6, x, 1]$?

   Solution :
   $[T_5, y, 4]$ *écrit* y *;* y *déjà écrit par* T_4 *(encore active)* $\Rightarrow$ *exécution non-stricte*
   $[T_4, z, 3]$ *écrit* z *;* z *déjà écrit par* T_1 *et* T_2*, déjà terminées* $\Rightarrow$ *exécution stricte*
   $[T_6, x, 1]$ *écrit* x *;* x *déjà écrit par* T_5 *(encore active)* $\Rightarrow$ *exécution non-stricte*

3. Quelles sont les entrées récupérables par l'algorithme de ramasse-miettes ?

   Solution :
   $[T_2, y, 1]$, $[T_2, z, 4]$ *récupérables, car* T_2 *rejetée*
   $[T_1, x, 3]$ *récupérable, car* T_3 *validée a écrit ensuite* x

4. Si les valeurs initiales des enregistrements étaient $x = 1$, $y = 2$,$z = 3$, et si une panne survenait à ce moment, quelles seraient les valeurs restaurées pour x, y et z après la reprise ?

   Solution :
   T_3 *est la dernière transaction validée à avoir écrit dans* x *(*$[T_3, x, 2]$*), donc la valeur restaurée est* x*=2.*
   Aucune transaction validée n'a écrit dans y*, donc la valeur restaurée est* y*=2, valeur initiale.*
   T_1 *est la dernière transaction validée à avoir écrit dans* z *(*$[T_1, z, 2]$*), donc la valeur restaurée est* z*=2.*

5. En pratique, on stocke souvent dans chaque entrée aussi l'image avant de l'écriture (la valeur de l'enregistrement avant l'écriture). Aussi, les positions des Commit et Abort ne sont pas connues, mais seulement les listes de transactions *liste_commit*, *liste_abort* et *liste_active* au moment courant (la fin du journal). Quel est dans ce cas le contenu du journal ? Les valeurs initiales sont celles précisées ci-dessus ($x = 1$, $y = 2$,$z = 3$), ainsi que les listes courantes de transactions.

   Solution :
   On rajoute dans chaque entrée l'image avant (en dernière position) :
   $[T_1, x, 3, 1]$, $[T_2, y, 1, 2]$, $[T_1, z, 2, 3]$, $[T_2, z, 4, 2]$, $[T_3, x, 2, 3]$, $[T_4, y, 3, 2]$, $[T_5, x, 5, 2]$
   liste_commit = $\{T_1, T_3\}$ *;* liste_abort = $\{T_2\}$ *;* liste_active = $\{T_4, T_5\}$
   Remarque *: Pour trouver l'image avant on cherche la dernière écriture non-annulée sur le même enregistrement. Il faut donc tenir compte des écritures annulées, par exemple pour l'écriture de* T_4 *sur* y*, l'image avant n'est pas celle écrite par* T_2 *(annulée), mais la valeur précédente à cette écriture.*

6. Expliquez le fonctionnement de l'algorithme de reprise avec annulation et avec répétition sur ce dernier journal.

   Solution :
   Les enregistrements à restaurer sont x*,* y *et* z*. On parcourt le journal de la fin vers le début :*
   • $[T_5, x, 5, 2]$ *n'est pas validée (*$T_5 \in$ liste_active*), donc la valeur à restaurer pour* x *est l'image avant, c'est à dire 2.*
   • $[T_4, y, 3, 2]$ *n'est pas validée non plus (*$T_4 \in$ liste_active*), donc la valeur à restaurer pour* y *est l'image avant, c'est à dire 2.*
   • $[T_3, x, 2, 3]$ *est ignorée, car* x *a déjà été restauré*
   • $[T_2, z, 4, 2]$ *n'est pas validée (*$T_2 \in$ liste_abort*), donc la valeur à restaurer pour* z *est l'image avant, c'est à dire 2.*
   • *on s'arrête ici, car tous les enregistrements ont été restaurés :* $x = 2, y = 2, z = 2$*.*

9.4 Concurrence : Gestion Bancaire

Les trois programmes suivants peuvent s'exécuter dans un système de gestion bancaire. *Débit* diminue le solde d'un compte `c` avec un montant donné `m`. Pour simplifier, tout débit est permis (on accepte des découverts). *Crédit* augmente le solde d'un compte `c` avec un montant donné `m`. *Transfert* transfère un montant `m` à partir d'un compte source `s` vers un compte destination `d`. L'exécution de chaque programme démarre par un **Start** et se termine par un **Commit** (non montrés ci-dessous).

```
Débit (c:Compte;      |    Crédit (c:Compte;      |    Transfert (s,d:Compte;
       m:Montant)     |            m:Montant)     |               m:Montant)
begin                 |    begin                  |    begin
  t := Read(c);       |      t = Read(c);         |      Débit(s,m);
  Write(c,t-m);       |      Write(c,t+m);        |      Crédit(d,m);
end                   |    end                    |    end
```

Le système exécute en même temps les trois opérations suivantes :
(1) un transfert de montant 100 du compte A vers le compte B
(2) un crédit de 200 pour le compte A
(3) un débit de 50 pour le compte B

1. Écrire les transactions T_1, T_2 et T_3 qui correspondent à ces opérations. Montrer que l'histoire **H** : $r_1[A]$ $r_3[B]$ $w_1[A]$ $r_2[A]$ $w_3[B]$ $r_1[B]$ c_3 $w_2[A]$ c_2 $w_1[B]$ c_1 est une exécution concurrente de T_1, T_2 et T_3.

   Solution :

   `Débit` *et* `Crédit` *sont constitués chacun d'une lecture, suivie d'une écriture. Dans ce cas, les transactions T_1, T_2 et T_3 seront :*

   T_1 *:* $r_1[A]$ $w_1[A]$ $r_1[B]$ $w_1[B]$ c_1
   T_2 *:* $r_2[A]$ $w_2[A]$ c_2
   T_3 *:* $r_3[B]$ $w_3[B]$ c_3

   L'histoire **H** *contient toutes les opérations de T_1, T_2 et T_3 et respecte l'ordre des opérations dans chaque transaction. Donc* **H** *est une exécution concurrente de T_1, T_2 et T_3.*

2. Mettre en évidence les conflits dans **H** et construire le graphe de sérialisation de cette histoire. **H** est-elle sérialisable ? **H** est-elle recouvrable ?

   Solution :

   Les conflits sur A : $r_1[A]$-$w_2[A]$ *;* $w_1[A]$-$r_2[A]$ *;* $w_1[A]$-$w_2[A]$
   Les conflits sur B : $r_3[B]$-$w_1[B]$ *;* $w_3[B]$-$r_1[B]$ *;* $w_3[B]$-$w_1[B]$
   Le graphe de sérialisation **SG(H)** *:* $T_3 \rightarrow T_1 \rightarrow T_2$
   H *est sérialisable, car le graphe ne contient pas de cycle.*
   H *n'est pas recouvrable, car T_2 lit A de T_1 (après $w_1[A]$ on a $r_2[A]$), mais T_2 se termine avant T_1. La même conclusion est obtenue en considérant la suite $w_3[B]$ $r_1[B]$.*

3. Quelle est l'exécution **H'** obtenue à partir de **H** par verrouillage à deux phases ? On suppose que les verrous d'une transaction sont relâchés après le Commit de celle-ci. Une opération bloquée en attente d'un verrou bloque le reste de sa transaction. Au moment du relâchement des verrous, les opérations en attente sont exécutées en priorité, dans l'ordre de leur blocage.
   Si au début le compte A avait un solde de 100 et B de 50, quel sera le solde des deux comptes après la reprise si une panne intervient après l'exécution de $w_1[B]$?

   Solution :

   $r_1[A]$, $r_3[B]$ *reçoivent les verrous de lecture et s'exécutent*
   $w_1[A]$ *obtient le verrou d'écriture sur A (déjà obtenu en lecture par T_1) et s'exécute*
   $r_2[A]$ *bloquée en attente de verrou sur A* $\Rightarrow$ T_2 *bloquée*

$w_3[B]$ obtient le verrou d'écriture sur B (déjà obtenu en lecture par T_3) et s'exécute
$r_1[B]$ bloquée en attente de verrou sur B $\Rightarrow$ T_1 bloquée
c_3 s'exécute et relâche les verrous sur B $\Rightarrow$ $r_1[B]$ débloquée, obtient le verrou et s'exécute (T_1 débloquée)
$w_2[A]$ et c_2 bloquées car T_2 bloquée
$w_1[B]$ obtient le verrou et s'exécute
c_1 s'exécute et relâche les verrous sur A $\Rightarrow$ $r_2[A]$, $w_2[A]$ et c_2 s'exécutent.
Le résultat est **H'** *: $r_1[A]$ $r_3[B]$ $w_1[A]$ $w_3[B]$ c_3 $r_1[B]$ $w_1[B]$ c_1 $r_2[A]$ $w_2[A]$ c_2*
Si une panne intervient après l'exécution de $w_1[B]$, seule la transaction T_3 (le débit de 50 sur B) est validée à ce moment. Après la reprise, seul l'effet de T_3 sera retrouvé, donc le compte A aura un solde de 100 et B de 0.

9.5 Concurrence : Agence environnementale

Une agence environnementale utilise une base de données relationnelle pour gérer des informations sur les véhicules. Le schéma est le suivant (les clés sont soulignées) :

propriétaire(nom, âge)
véhicule(immatriculation, kilométrage, nom, idmodèle) (nom = nom propriétaire)
modèle(idmodèle, crashtest, consommation)
carburant(désignation, cours, pollution)
utilise(idmodèle, désignation)

L'exécution suivante est reçue par le système de l'agence environnementale :

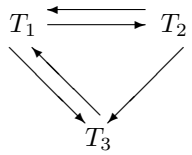
$H : r_1[x]r_2[y]r_3[x]w_2[y]w_1[x]r_3[y]r_1[z]w_2[z]w_1[y]c_1c_2w_3[z]c_3$

1. Parmi les programmes qui s exécutent dans le système, il y a *KilométrAge(i, k, a)*, qui fixe pour le véhicule d'immatriculation i le kilométrage à k et l'age du propriétaire à a. Montrez quelle transaction de H pourrait provenir de *KilométrAge*.
   On considère que les enregistrements (variables) de H sont des nuplets des relations de la base de données. On suppose que tout nuplet est accessible directement si l'on connaît sa clé.

   Solution :
   Les transactions extraites de l'exécution H sont :
   T_1 : $r_1[x]$ $w_1[x]$ $r_1[z]$ $w_1[y]$ c_1
   T_2 : $r_2[y]$ $w_2[y]$ $w_2[z]$ c_2
   T_3 : $r_3[x]$ $r_3[y]$ $w_3[z]$ c_3
   Dans KilométrAge*, on accède au nuplet de clé i de* **véhicule***. On lit ce nuplet pour récupérer le nom du propriétaire et on l'écrit pour actualiser le kilométrage. Ensuite, avec le nom du propriétaire (clé) et l'age a, on écrit le nuplet de* **propriétaire** *(pas besoin de lecture).*
   Il y a donc une mise-à-jour (lecture-écriture) d'un enregistrement et une écriture d'un autre enregistrement. La seule transaction qui correspond est T_2.

2. Identifiez tous les conflits dans H et vérifiez si l'exécution est sérialisable en construisant le graphe de sérialisation.

   Solution :
   Les conflits :
   sur x : $r_3[x]$-$w_1[x]$
   sur y : $r_2[y]$-$w_1[y]$, $w_2[y]$-$r_3[y]$, $w_2[y]$-$w_1[y]$, $r_3[y]$-$w_1[y]$
   sur z : $r_1[z]$-$w_2[z]$, $r_1[z]$-$w_3[z]$, $w_2[z]$-$w_3[z]$
   Le graphe de sérialisation contient plusieurs cycles, H n'est pas sérialisable.



3. A quel niveau de recouvrabilité se situe H (recouvrable, évitant les annulations en cascade, stricte) ?

   Solution :

   On regarde pour chaque écriture de H si entre l'écriture et sa validation il y a d'autres opérations sur le même enregistrement. La seule écriture dans ce cas est $w_2[y]$, car on a $r_3[y]$ avant c_2. Donc H n'évite pas les annulations en cascade, mais comme l'écriture $w_2[y]$ est validée avant $r_3[y]$, H est recouvrable.

4. Quelle est l'exécution obtenue par verrouillage à deux phases à partir de H ? Le relâchement des verrous d'une transaction se fait au Commit et à ce moment on exécute en priorité les opérations bloquées en attente de verrou, dans l'ordre de leur blocage.

   Solution :

   $r_1[x]$, $r_2[y]$ s'exécutent, en prenant les verrous de lecture
   $r_3[x]$ partage le verrou de lecture sur x avec $r_1[x]$ et s'exécute
   $w_2[y]$ prend le verrou d'écriture et s'exécute (pas de conflit avec $r_2[y]$)
   $w_1[x]$ bloquée par $r_3[x]$, donc T_1 bloquée
   $r_3[y]$ bloquée par $w_2[y]$, donc T_3 bloquée
   $r_1[z]$ bloquée, car T_1 bloquée
   $w_2[z]$ prend le verrou d'écriture et s'exécute
   $w_1[y]$, c_1 bloquées, car T_1 bloquée
   c_2 s'exécute et relâche les verrous de T_2
   $\rightarrow$ seule $r_3[y]$ peut obtenir le verrou et s'exécuter
   $w_3[z]$ s'exécute
   c_3 s'exécute et relâche les verrous de T_3
   $\rightarrow$ $w_1[x]$, $r_1[z]$, $w_1[y]$, c_1 s'exécutent
   Résultat *: $r_1[x]$ $r_2[y]$ $r_3[x]$ $w_2[y]$ $w_2[z]$ c_2 $r_3[y]$ $w_3[z]$ c_3 $w_1[x]$ $r_1[z]$ $w_1[y]$ c_1*