Université de Montpellier - Master informatique Janvier 2022

Théorie des bases de données et de connaissances (HAI933I)

Examen (avec correction succinte)

Exercice 1 (Homomorphismes) - 1 pt

Soient deux ensembles d'atomes, où a est la seule constante :
$Q_1 = \{p(a, x_1), q(x_1, y_1)\}$
$Q_2 = \{p(x_2, y_2), q(y_2, z_2), \cancel{q(z_2, u_2)}\}$

Question 1 Existe-t-il un homomorphisme de Q_1 dans Q_2? De Q_2 dans Q_1?

Non : car la constante a n'apparait pas dans Q_2.
Oui : $x_2 \mapsto a$, $y_2 \mapsto x_1$, $z_2 \mapsto y_1$, $\cancel{u_2 \mapsto x_1}$.

Question 2 Si Q_1 et Q_2 représentent des requêtes conjonctives booléennes, a-t-on $Q_1 \sqsubseteq Q_2$? $Q_2 \sqsubseteq Q_1$?

Rappel: Etant données deux requêtes conjonctives booléennes Q et Q', la notation $Q \sqsubseteq Q'$ signifie que toute base de faits qui répond oui à Q répond aussi oui à Q'.

$Q \sqsubseteq Q'$ *ssi il y a un homomorphisme de* Q' *dans* Q. *On a donc* $Q_1 \sqsubseteq Q_2$ *et* $Q_2 \not\sqsubseteq Q_1$.

Exercice 2 (Chase) - 4,5 pts

Soit la base de connaissances $\mathcal{K} = (F, \mathcal{R} = \{R_1, R_2\})$ avec :
$F = \{p(a, b), r(a)\}$
$R_1 = p(x, y) \rightarrow \exists z\ p(y, z)$
$R_2 = r(x) \wedge p(x, y) \rightarrow p(y, y)$.

Nous allons considérer trois variantes du chase :

- l'*oblivious* chase, qui effectue toutes les applications de règles possibles ;
- le *restricted* chase, qui n'effectue l'application d'une règle $R = B \rightarrow H$ selon un homomorphisme h de B dans la base de faits courante F que si h ne *s'étend pas* à un homomorphisme de $B \cup H$ dans F ;
- le *core* chase, qui applique les règles comme le restricted chase, mais calcule le *core* de la base de faits obtenue après chaque application de règle.

Question 1 L'*oblivious chase* s'arrête-t-il sur cette base de connaissances ? Pourquoi ? (On ne vous demande pas de donner le résultat du chase).

Cette saturation est infinie :

$$\{p(a, b), r(a), p(b, z_0), p(b, b), p(b, z_1)\} \cup \{p(z_i, z_{i+2}) | i \geq 0\}$$

Question 2 Définir le résultat obtenu par le *restricted chase.* Si l'ordre d'application des règles change le résultat, considérez les différents cas possibles.

Selon qu'on applique d'abord R_2 ou R_1, la saturation est finie ou infinie :
- R_2 d'abord : on obtient $\{p(a, b), r(a), p(b, b)\}$ par R_2, ce qui rend l'application de R_1 redondante.
- R_1 d'abord : on obtient "une chaine infinie" (au lieu de 2 avec l'oblivious chase) :
$\{p(a, b), r(a), p(b, z_0), p(b, b)\} \cup \{p(z_i, z_{i+1}) | i \geq 0\}$

Question 3 Définir le résultat obtenu par le *core chase*.

On obtient $\{p(a,b), r(a), p(b,b)\}$. *L'ordre d'application des règles est indifférent : même si on commence par* R_1*, l'atome ajouté* $p(b, z_0)$ *est supprimé par le calcul du core après l'application de* R_2.

Question 4 La base de connaissances $\mathcal{K}$ possède-t-elle un modèle universel fini ?

Oui, notamment celui calculé par le core chase, cf. Q3.

Question 5 L'ensemble de règles $\mathcal{R}$ assure-t-il que *pour toute base de faits* F, $(F, \mathcal{R})$ a un modèle universel fini?

Non. Par exemple, si l'on prend $F = \{p(a,b)\}$ *, seule* R_1 *est applicable et le core chase est infini (donc la KB n'a pas de modèle fini).*

Exercice 5 (OWA/CWA) - 2,5 pts

On considère des bases de faits et requêtes conjonctives munies de la négation. On se place dans un premier temps en monde ouvert, autrement dit dans le cadre de la logique classique.

Soit la base de faits $F = \{q(a), q(b), p(a,a), \neg p(b,b)\}$ où a et b sont des constantes.

Question 1. Soit $Q_1() = \exists x \exists y \exists z\ (p(x,y) \wedge q(z) \wedge \neg p(y,z))$. Déterminer $Q_1(F)$, l'ensemble des réponses à Q_1 sur F.
Tout modèle de F *rend vrai* $p(a,b)$ *ou* $\neg p(a,b)$*. Dans le premier cas, on a la "bonne affectation"* $\{x \mapsto a, y \mapsto b, z \mapsto b\}$*, dans le second cas, on a la "bonne affectation"* $\{x \mapsto a, y \mapsto a, z \mapsto b\}$*. Donc tout modèle de* F *est un modèle de* Q_1*. Donc* F *répond oui à* Q_1*, autrement dit* $Q_1(F) = \{()\}$.

Question 2. Même question avec $Q_2(x) = \exists y \exists z\ (p(x,y) \wedge q(z) \wedge \neg p(y,z))$.
$Q_2(F) = \{(a)\}$ *car dans tous les modèles de* F *l'une des deux bonnes affectations précédentes envoie* x *sur* a*. Autrement dit,* $F \models Q_2[x \mapsto a]$ *où* $Q_2[x \mapsto a]$ *est la requête booléenne* $\exists y \exists z\ (p(a,y) \wedge q(z) \wedge \neg p(y,z))$.

Question 3. Même question avec $Q_3(y) = \exists x \exists z\ (p(x,y) \wedge q(z) \wedge \neg p(y,z))$.
$Q_3(F) = \emptyset$ *car aucune des requêtes booléennes obtenues en remplaçant* y *par une constante n'est conséquence logique de* F.

Question 4. On considère la base de faits $F' = \{q(a), q(b), p(a,a)\}$ et on fait l'hypothèse du monde clos. Quelles sont alors les réponses à Q_1, Q_2 et Q_3 sur F' ?
Soit $Q^+ = \{p(x,y), q(z)\}$ *et* $Q^- = \{p(y,z)\}$*. Il y a un seul homomorphisme* h *de* Q^+ *dans* F' *tel que* $h(Q^-) \cap F' = \emptyset$*:* $x \mapsto a, y \mapsto a, z \mapsto b$*. On en conclut que* $Q_1(F') \neq \emptyset$, $Q_2(F') = Q_3(F') = \{(a)\}$.

Université de Montpellier - Master informatique Janvier 2022

Théorie des bases de données et de connaissances (HAI933I)

Examen

Exercise 4 (First Order Queries) - 2 pts

Consider the following database:

Films

Title	Director	Actor
The Imitation Game	Tyldum	Cumberbatch
The Imitation Game	Tyldum	Knightley
...	...	...
Internet's Own Boy	Knappenberger	Swartz
Internet's Own Boy	Knappenberger	Lessig
Internet's Own Boy	Knappenberger	Berners-Lee
...	...	...
Dogma	Smith	Damon
Dogma	Smith	Affleck

Venues

Cinema	Address	Phone
UFA	St. Petersburger Str. 24	4825825
Schauburg	Königsbrücker Str. 55	8032185
...	...	...

Program

Cinema	Title	Time
Schauburg	The Imitation Game	19:30
Schauburg	Dogma	20:45
UFA	The Imitation Game	22:45

Write the following queries as first order queries (half a point each):

1. Who are the directors that have worked with the actor "Cumberbatch"?

$$\exists y_T.Films(y_T, x_D, Cumberbatch)$$

2. List the directors that have directed a film shown at "UFA".

$$\exists y_T, y_A, z_T.Films(y_T, x_D, y_A) \land Program(UFA, y_T, z_T)$$

3. Find out all the actors that do not have "Tyldum" as one of their directors.

$$\exists y_T, y_D.Films(y_T, y_D, x_A) \land \forall z_T, z_D.(Films(z_T, z_D, x_A) \rightarrow z_D \not\approx Tyldum)$$

4. Write a Boolean query to determine if two different directors have directed the same movie.

$$\exists y_T, y_D, z_D, y_A, z_A.Films(y_T, y_D, y_A) \land Films(y_T, z_D, z_A) \land y_D \not\approx z_D$$

Exercise 5 (Join Trees) - 2 pts

Consider the following queries:

1. $\exists x, y, z, w.P(x, y, z) \land Q(x, z, w) \land R(x, x, y, w) \land P(y, w, z)$
2. $\exists z_1, \ldots, z_6.V(z_1, z_2, z_6) \land O(z_1, z_2, z_3) \land M(z_2, z_4, z_3) \land L(z_1, z_3, z_5)$

If possible, define a join tree for each of these queries. If it is not possible to do so, explain why.

To determine if we can define a join tree for a query, we can apply the following algorithm:

GYO-reduction algorithm to check acyclicity:
(after Graham [1979] and Yu & Özsoyoğlu [1979])

Input: hypergraph $H = \langle V, E \rangle$ (we don't need relation labels here)
Output: GYO-reduct of H

Apply the following simplification rules as long as possible:

(1) Delete all vertices that occur in at most one hyperedge

(2) Delete all hyperedges that are empty or that are contained in other hyperedges

Definition

A hypergraph is **acyclic** if its GYO-reduct is $\langle \emptyset, \emptyset \rangle$.
A CQ is **acyclic** if its associated hypergraph is.

Applying the algorithm, we can check that:

- Query (1) is not acyclic.
- Query (2) is acyclic. Furthermore, here's a join tree for this query: $\{O(z_1, z_2, z_3) \rightarrow V(z_1, z_2, z_6), O(z_1, z_2, z_3) \rightarrow L(z_1, z_3, z_5), O(z_1, z_2, z_3) \rightarrow M(z_2, z_4, z_3)\}$

Correction Partie “Modèles stables” - HAI933I

Exercice 6 – Modèles stables (4 points)

Question 1 Utilisez la méthode vue en cours pour mettre le programme suivant sous forme propositionnelle. Vous détaillerez soigneusement les étapes de cette transformation.

$$p(a), q(a, b).$$
$$p(X), \text{not } q(X, Y) \rightarrow r(X).$$

Étape 1 : le programme est déjà skolemisé (pas de variable existentielle). *Étape 2 :* on doit normaliser la règle car toutes les variables de son corps négatif ne sont pas dans le corps positif. On obtient :

$$\begin{array}{ll} (F) & p(a), q(a, b). \\ (R) & p(X), \text{not } s(X) \rightarrow r(X). \\ (S) & q(X, Y) \rightarrow s(X). \end{array}$$

Étape 3 (grounding) : le domaine de Herbrand est $\mathcal{H} = \{a, b\}$, on instancie les règles de toutes les façons possibles avec les constantes de $\mathcal{H}$.

$$\begin{array}{ll} (F) & p(a), q(a, b). \\ (R_1) & p(a), \text{not } s(a) \rightarrow r(a). \\ (R_2) & p(b), \text{not } s(b) \rightarrow r(b). \end{array}$$

$$\begin{array}{ll} (S_1) & q(a, a) \rightarrow s(a). \\ (S_2) & q(a, b) \rightarrow s(a). \\ (S_3) & q(b, a) \rightarrow s(b). \\ (S_4) & q(b, b) \rightarrow s(b). \end{array}$$

Dans le cours, j'avais donné une étape 4 de propositionalisation, qui consiste par exemple à donner le nom a_1 à l'atome $p(a)$, a_2 à l'atome $q(a, b)$ ...Mais j'avais également indiqué à l'oral que cette étape est facultative, puisqu'on peut déjà considérer que $p(a)$ est un atome propositionnel : il est juste écrit de façon un peu bizarre.

Détaillez le calcul d'une dérivation persistante et complète utilisant les règles que vous avez obtenues.

Remarque : si le résultat de cette dérivation n'est pas conforme à votre intuition, peut-être avez vous oublié une étape lors de votre transformation.

On part de $F = F_0 = \{p(a), q(a, b)\}$, on applique (S_2) pour obtenir $F_1 = \{p(a), q(a, b), s(a)\}$. Cette dérivation est bien complète (plus aucune règle n'est applicable – voir que (R_1) est bloquée par $s(a)$) et elle ne peut être que persistante puisqu'aucune règle avec corps négatif n'a été appliquée.

La deuxième partie (facile) de la question a souvent été oubliée : pensez à lire les sujets jusqu'au bout !

Question 2 Nous considérons maintenant le programme Π défini ci-dessous (7 lignes au total), et souhaitons savoir si les ensembles d'atomes $E_1 = \{a, c, d, f\}$ et $E_2 = \{a, e, f\}$ sont des modèles stables de ce programme.

$a.$
$a, \text{not}\, c \rightarrow b.$
$a \rightarrow f.$
$f, \text{not}\, d \rightarrow e.$
$b \rightarrow c.$
$f, \text{not}\, e \rightarrow d.$
$d \rightarrow c.$

Vous construirez les programmes réduits associés à ces deux ensembles d'atomes et les utiliserez pour répondre à la question en utilisant la définition par point fixe.

Le programme Π_1 obtenu par la réduction de Π par E_1 est :

$a.$
$a \rightarrow f.$
$b \rightarrow c.$
$f \rightarrow d.$
$d \rightarrow c.$

La saturation de Π_1 donne l'ensemble d'atomes $\Pi_1^* = \{a, f, d, c\}$. On a bien $\Pi_1^* = E_1$ donc E_1 est un modèle stable de Π.

De même, le programme Π_2 obtenu par la réduction de Π par E_2 est :

$a.$
$a \rightarrow b.$
$a \rightarrow f.$
$f \rightarrow e.$
$b \rightarrow c.$
$d \rightarrow c.$

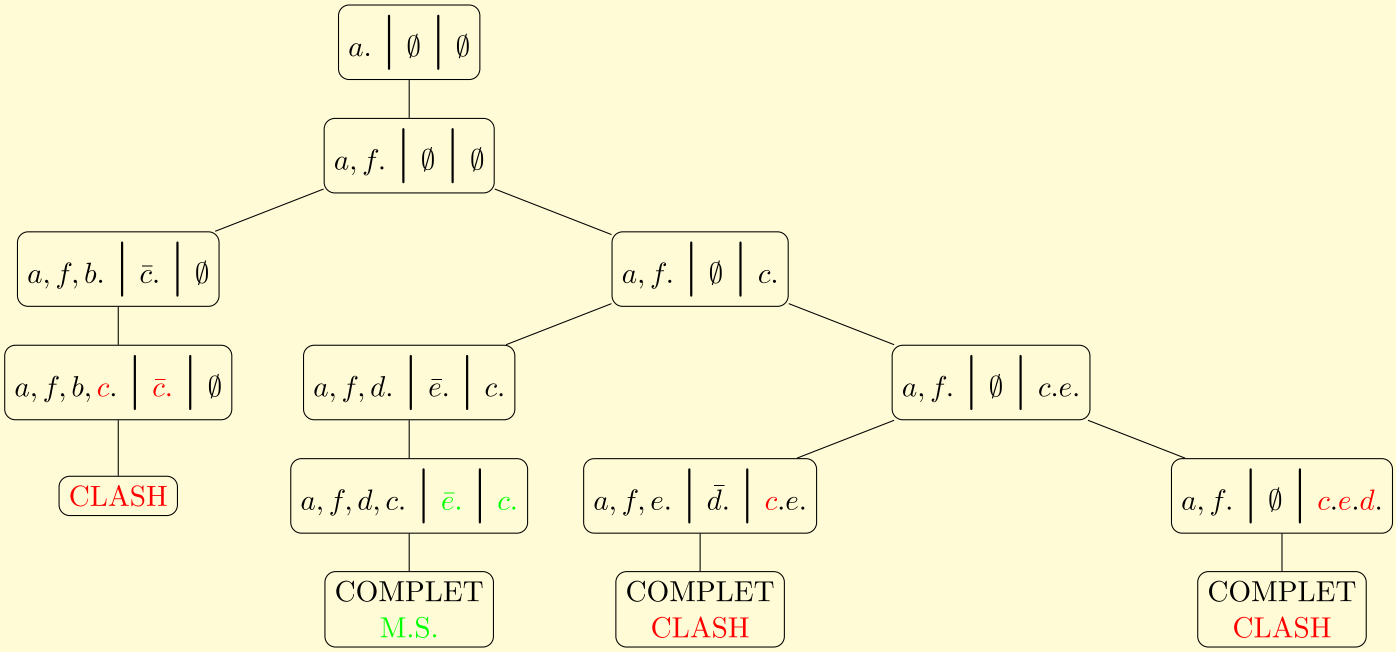
La saturation de Π_2 donne l'ensemble d'atomes $\Pi_2^* = \{a, b, f, e, c, d\}$. On n'a pas $\Pi_2^* = E_2$ donc E_2 n'est pas un modèle stable de Π.

Question 3 Reprenons le programme Π de la question 2. Les ensembles d'atomes $E_4 = \{a, c, d, e, f\}$, et $E_5 = \{c, d, e, f\}$ sont-ils des modèles stables de ce programme ? Votre réponse devra être argumentée, mais n'utilisera pas nécessairement la définition par point fixe.

L'ensemble d'atome E_4 ne peut pas être un modèle stable car on a $E_1 \subset E_4$: or E_1 est un modèle stable, et ceux-ci sont maximaux par inclusion.

De même l'ensemble d'atomes E_5 n'est pas un modèle stable : il ne contient pas a qui est un fait et doit donc être dans tous les modèles stables.

Question 4 Utilisez l'algorithme ASPERIX vu en cours pour trouver tous les modèles stables du programme Π de la question 2. *Vous vérifierez que vos réponses aux questions précédentes sont cohérentes.*



La seule branche complète sans clash nous donne le modèle stable E_1 de la question précédente.

Comme très souvent, vous vous êtes perdus dans des arbres ASPERIX beaucoup trop grands ! Un peu de stratégie : appliquez d'abord les règles positives, qui ne créent pas de nouvelles branches, puis des règles dont la conclusion permettra d'appliquer de nouvelles règles positives. Et c'est encore mieux quand ces applications permettent la violation d'une contrainte négative...

Considérant le nombre de modèles stables que vous avez trouvé, pouvez-vous en déduire si le programme Π est stratifiable ?

On sait que si Π est stratifiable, alors il y aura un et un seul modèle stable. Mais la réciproque n'est pas vraie : on ne peut pas déduire que le programme est stratifiable du fait qu'il admette un et un seul modèle stable.

Exercice 7 – Modélisation en ASP – 4 pts

Les points sont accordés si au moins la moitié de l'exercice est correctement traitée.

Dans le jeu “Le compte est bon”, il faut réussir à obtenir un nombre cible à partir de 6 nombres initiaux, en utilisant les 4 opérations arithmétiques élémentaires : addition, soustraction, multiplication, division. Les conditions initiales du jeu seront codées comme dans l'exemple suivant :

$$\begin{array}{l} \text{disponible}(a, 25, 0). \\ \text{disponible}(b, 10, 0). \\ \text{disponible}(c, 10, 0). \\ \text{disponible}(d, 7, 0). \\ \text{disponible}(e, 3, 0). \\ \text{disponible}(f, 4, 0). \\ \text{cible}(284). \end{array}$$

L'atome disponible$(a, 25, 0)$. se lit *“Le nombre qui a pour identifiant a, dont la valeur est 25, est disponible à l'étape 0”*. Coder uniquement le fait que *“le nombre 25 est disponible à l'étape 0”* ne permettrait pas d'avoir des occurences multiples d'un même nombre. En supposant qu'on ait pu choisir 2 nombres et une opération à chaque étape, 4 règles conçues sur le modèle de la règle suivante (qui concerne l'addition) permettent de générer un nouveau nombre à l'étape N+1 :

$$\text{choisi1}(X1, N), \text{choisi2}(X2, N), \text{opchoisie}(\text{addition}, N),$$
$$\text{disponible}(X1, V1, N), \text{disponible}(X2, V2, N) \rightarrow \text{disponible}(Y, V1 + V2, N + 1).$$

Le but de l'exercice est d'écrire un programme dont les modèles stables encodent toutes les manières d'arriver au nombre cible. Il s' agit d'une version simplifiée du jeu, en particulier, on ne cherche pas à générer un nombre “le plus proche possible” de la cible.

Question 1 Ecrire une règle permettant de générer l'atome $fini()$ dès que la valeur cible a été trouvée. Complétez ensuite le programme pour que les modèles stables contiennent uniquement les cas où cette valeur cible a été trouvée.

```
fini(N) :- disponible(X, V, N), cible(V).
% Deuxieme partie de la question
autredisponible(X, N) :- disponible(X, V, N), disponible(Y, W, N), X != Y.
:- disponible(X, V, N), not fini(N), not autredisponible(X, N).
```

Question 2 L'atome $choisi1(X, N)$ signifie que le nombre d'identifiant X a été choisi comme premier argument de notre calcul à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :
— il n'y a plus besoin de faire de choix quand le jeu est fini ;
— on ne peut pas faire de choix quand on n'a pas au moins deux identifiants de nombres disponibles ;
— un seul identifiant de nombre peut être choisi.

```
choisi1(X, N) :- disponible(X, V, N), not fini(N), not autrechoisi1(X, N).
autrechoisi1(X, N) :- disponible(X, V, N), choisi1(Y, N), X != Y.
```

C'est exactement le choix n-aire vu en cours. Voir qu'il y a nécessairement deux disponibles sinon la contrainte de la réponse à la question 1 serait déclenchée.

Question 3 L'atome $choisi2(X, N)$ signifie que le nombre d'identifiant X a été choisi comme deuxième argument de notre calcul à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :
— on ne fait le choix 2 qu'après le choix 1 ;
— on ne peut pas faire le même choix en choix2 qu'en choix1 ;
— un seul identifiant de nombre peut être choisi.

```
choisi2(X, N) :- disponible(X, V, N), choisi1(Y, N), X!= Y, not autrechoisi2(X, N).
autrechoisi2(X, N) :- disponible(X, V, N), choisi2(Y, N), X != Y.
```

Question 4 L'atome *opchoisie*(X, N) signifie que l'opération X a été choisie à l'étape N. Ecrire la (ou les) règle(s) permettant de générer cet atome. Vous vous inspirerez du choix multiple vu en cours, et prendrez en compte les points suivants :

— on ne peut choisir que parmi les opérations codées dans les atomes suivants : $op(addition), op(soustraction), op(multiplication), op(division)$;

— il n'y a plus besoin de faire de choix quand le jeu est fini ;

— une seule opération peut être choisie à chaque étape ;

— (OPTIONNEL) on ne peut choisir la division que si son résultat est un nombre entier : on pourra tester dans le corps de la règle $X1\%X2 == 0$ pour savoir si la valeur $X1$ du premier nombre choisi est divisible par la valeur $X2$ du second.

```
opchoisie(O, N) :- op(O), chosi1(X, N), choisi2(Y, N), not autreop(O, N), not impossible(O, N).
valeurs(V, W, N) :- choisi1(X, N), choisi2(Y, N), disponible(X, V, N), disponible(Y, W, N).
impossible(division, N) :- valeurs(V, W, N), V \ W != 0.
```

Question 5 Ecrire la ou les règles permettant d'assurer que les nombres qui n'ont pas été choisis à une étape sont encore disponibles à l'étape suivante.

```
disponible(X, V, N+1) :- disponible(X, V, N), not choisi1(X, N), not choisi2(X, N).
```

Question 6 L'algorithme ASPERIX s'arrête-t-il sur votre programme ? Justifiez votre réponse. Montrez que le grounding de ce programme est infini. Identifiez-en toutes les causes. Proposez une modification de votre programme pour obtenir un grounding fini.

L'algorithme ASPERIX s'arrête sur ce programme car à chaque étape on choisit 2 nombres. Les autres nombres restent disponibles, ainsi que celui généré par l'opération. Le nombre de dispobibles décroît donc à chaque étape et donc pour chaque branche, au bout d'un nombre fini d'opérations, soit on aura généré `fini` et plus aucun choix ne sera effectué, soit la contrainte de la question 1 sera déclenchée.

Par contre, le grounding de CLINGO est infini. Les règles responsables de ceci sont :

1. la règle de calcul donnée dans l'énoncé qui :
   (a) crée une nouvel identificateur `Y`
   (b) crée un nouveau nombre `V1 + V2`
   (c) crée une nouvelle étape `N+1`
2. la règle de la question 5 qui crée une nouvelle étape `N+1`

Le problème 1.a peut être facilement réglé. Il suffit d'écrire la règle sous la forme :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N).
```

En gardant l'identifiant du choix 1 pour a valeur obtenue, on n'a pas besoin de générer de nouvel identificateur.

Pour les problèmes 1.c et 2, la solution vue en cours (borner le nombre d'étapes) peut être utilisée. On peut rajouter dans le programme :

```
#const maxetape = 5
etape(0..maxetape).
```

Puis modifier la règle précédente et la règle de la question 5 en utilisant cette borne :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N), etape(N+1).
disponible(X, V, N+1) :- disponible(X, V, N), not choisi1(X, N), not choisi2(X, N), etape(N+1).
```

On remarque que si l’instance du problème “Le compte est bon” commence avec N identificateurs de nombres, on reste complet en prenant `const maxetape = N`.

Curieusement, lorsqu’on effectue ces modifications, le grounding de CLINGO (qui est plus intelligent que la version naïve vue en cours) s’arrête maintenant si on ne considère que les opérations `addition`, `multiplication` et `soustraction`. Mais dès qu’on rajoute `division`, le grounding redevient infini.

Une solution consiste en la génération initiale de toutes les valeurs possibles pour aider le grounding de CLINGO. Par exemple :

```
possible(V, N):- disponible(X, V, 0).
possible (V+W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V-W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V*W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1).
possible (V/W, N+1):- possible(V, N), possible(W, M), N >= M, etape(N+1), V\W == 0.
```

Puis l’utilisation de ces possibles dans les règles générant de nouveaux nombres :

```
disponible(X, V+W, N+1) :- opchoisie(addition, N), chois1(X, V, N), valeurs(V, W, N), etape(N+1),
    possible(V+W, M).
```

La le grounding s’arrête, mais l’espace de recherche est tellement énorme que CLINGO plante à partir de `maxetape = 3`.