Correction du contrôle continu 2
Partie modèles stables

Jean-François Baget `baget@lirmm.fr` `https://www.lirmm.fr/~baget`

3 décembre 2025

Dans ce document, vous trouverez à la fois le sujet du contrôle continu (sur fond blanc), le corrigé lui-même (dans des boîtes jaunes), des versions alternatives du corrigé (dans des boîtes oranges), de nombreux commentaires (dans des boîtes vertes telles que celle-ci) et (nouveauté) des rappels de cours (dans des boîtes grises). Si vous avez des questions, n'hésitez pas à me contacter par mail.

A l'issue de la correction de la partie modèles stables (sur 8 points), je suis limite déprimé par vos copies. Mon objectif était pourtant un sujet facile, ne portant que sur les exercices types vus et revus en cours. Les statistiques sont pourtant celles-ci:

Nombre d'observations : 42
Min / max observés : 0 / 6,5
Moyenne : 2,18
Médiane : 2,0
Ecart-type : 1,75
Q1 (25 %) : 0,75
Q3 (75 %) : 3,0

Un quart des étudiants a moins de 2/20, et les trois quarts ont moins de 8/20. Il s'agit donc essentiellement d'un manque flagrant de *travail* (et peut-être de présence ou d'attention en cours). Même parmi ceux qui ont bien réussi (seuls 7 étudiants ont la moyenne, bravo à eux), il reste des incompréhensions. Pour tous, je ne peux que vous encourager à *travailler* cette correction (pas seulement la lire). Il n'est pas possible de chercher des optimisations (ni même de modéliser correctement) si les mécanismes de raisonnement de base ne sont pas parfaitement intégrés.

1 Partie modèles stables

Attention, dans tout ce qui suit, les règles sont des règles existentielles avec négation, et pas des règles ASP. Il peut y avoir de subtiles différences...

Ici, les subtiles différences concernaient les corps négatifs non normaux qui vous ont beaucoup perturbé...

Question 1 Calculez une dérivation persistante et complète du programme suivant. Vous détaillerez chaque étape de l'application, en explicitant pourquoi les déclencheurs sont ou ne sont pas bloqués, et en précisant pourquoi la dérivation est persistante et complète.

```
p(a, b).  q(b, c).  q(c, b).
p(a, a).  q(a, c).  q(a, a).
s(X) :- p(X, Y), not ( q(Y, Z), r(Z) ), not q(Z, Y).
```

On cherche d'abord les *déclencheurs* de la règle R dans les atomes de la base de faits F, c'est à dire les homomorphismes de $p(X, Y)$ (le corps positif) dans F. On en trouve deux $h_1 = \{X \to a, Y \to b\}$ et $h_2 = \{X \to a, Y \to a\}$. Or il existe un homomorphisme du deuxième corps négatif $q(Z, Y)$ qui étend h_1 (c'est à dire qui envoie les mêmes variables sur les mêmes valeurs), c'est $h'_1 = \{Z \to c, Y \to b\}$. Ceci suffit à prouver que h_1 est bloqué. De la même façon, le même deuxième corps négatif bloque h_2 par $h'_2 = \{Z \to a, Y \to a\}$. Tous les déclencheurs de R sont bloqués, et donc la règle n'est pas applicable.

Aucune règle n'est donc applicable sur le dernier (et premier) ensemble d'atomes de la dérivation, celle-ci est donc *complète*. Et comme aucune règle n'a été appliquée, aucune ne peut être remise en compte plus tard dans la dérivation, qui est donc *persistante*.

Tout d'abord, toutes mes excuses pour une typo, qui a rendu l'exercice plus facile que ce que j'avais prévu. En effet, le programme que je voulais donner était celui-ci (le dernier $q(a, a)$ devait être un $q(a, b)$).

```
p(a, b).  q(b, c).  q(c, b).
p(a, a).  q(a, c).  q(a, b).
s(X) :- p(X, Y), not ( q(Y, Z), r(Z) ), not q(Z, Y).
```

Ceci ne changeait pas le blocage de h_1, toujours avec le même h'_1. Par contre, h_2 n'était cette fois-ci plus bloqué, et il aurait fallu le justifier par le fait que ni le premier corps négatif, ni le second ne pouvaient bloquer h_2. Il aurait alors fallu dire, par exemple:

- il n'y a pas d'homomorphisme du premier corps négatif $q(Y, Z), r(Z)$ dans F qui étend h_2 car il n'y a pas d'homomorphisme de $q(a, Z), r(Z)$ dans F,
- de la même façon, il n'y a pas d'homomorphisme du second corps négatif $q(Z, Y)$ qui étend h_2 car il n'y a pas d'homomorphisme de $q(Z, a)$ dans F.

La règle est donc applicable suivant le trigger h_2, et cette application sur $F = F_0$ dans notre dérivation produit un ensemble d'atomes $F_1 = F \cup \{s(a)\}$. Il n'y a pas de nouveau trigger dans F_1, donc plus aucune règle n'est applicable, la dérivation est donc *complète*. On vérifie alors que l'application sur F_0 suivant h_2 est toujours une application sur F_1 suivant h_2 (évident, l'ajout d'un $s(a)$ ne risque pas de créer un blocage): la dérivation est alors *persistante*.

Soyez malins, gagnez du temps ! Dans de (trop) nombreuses copies, vous avez justifié (parfois sur plus de 10 lignes) que le premier corps négatif n'était pas bloquant avant de montrer que le second l'était. Non seulement c'est de la perte de temps (puisque un blocage suffit à bloquer), mais des arguments maladroits (je suis gentil) sur la partie inutile ne font qu'énerver le correcteur, sans possibilité de points à la clé. Plusieurs d'entre vous se sont plaint de la longueur de l'examen, mais il faut aussi que vous fassiez des efforts de votre côté.

La petite boutique des horreurs, épisode 1: dans 5 ou 6 copies, j'ai vu une "définition" de l'applicabilité qui ressemblait à ceci: je considère *toutes* les variables du corps, ici X, Y, Z, qu'elles soient dans le corps positif ou dans le corps négatif. Je choisis maintenant un mapping de ces variables dans les termes de F, par exemple $h = \{X \to a, Y \to b, Z \to b\}$. Si maintenant ce mapping est un homomorphisme du corps positif dans F mais n'est un homomorphisme d'aucun

corps négatif dans F, alors la règle est applicable suivant h. C'est joli, mais ce n'est pas ma définition, et ça ne donne pas le même résultat: d'après cette "définition", la règle serait applicable suivant h, c'est à dire h_1, et on a bien vu que ce n'était pas le cas.

Pour ceux qui ont fait ça (j'ai les noms), essayez de bien comprendre pourquoi c'est faux ! L'idée est la suivante:

- avec la définition donnée en cours, il suffit de trouver un homomorphisme du corps négatif pour bloquer;
- avec cette "définition", il suffit de trouver un mapping qui n'est pas un homomorphisme pour ne pas bloquer, même si il y en a un autre qui est un homomorphisme, et qui devrait donc bloquer suivant la définition standard (et correcte).

Question 2 Mettre le programme suivant sous forme propositionnelle. Veuillez indiquer soigneusement les étapes de la méthode qque vous suivez. Si le programme obtenu est trop long, donnez suffisamment de règles pour montrer que vous avez bien compris...

```
p(a).
s(X, Z) :- p(X), not ( q(X, Y), r(Y) ).
```

Un exercice qui aurait dû être facile, mais sur lequel j'ai vu des choses très surprenantes. Ma correction ci-dessous est volontairement longue, puisqu'elle contient de nombreux rappels que je juge malheureusement nécessaires.

Rappel de l'algorithme de propositionnalisation La mise sous forme propositionnelle se fait en trois étapes successives:

Skolemisation: on vérifie que la base faits ne contient pas de variables, sinon on associe à chaque nom de variable dans la base une nouvelle (fresh) constante, et on remplace chaque variable par sa constante associée (freeze).

Puis on vérifie chaque règle: à chaque nom de variable existentielle dans la tête de la règle (celles qui sont dans la tête mais pas dans le corps), on associe un terme fonctionnel construit de la façon suivante. Le nom de fonction est unique à la règle et à ce nom de variable (par exemple f_X^R pour la variable X dans la règle R, mais si il n'y a qu'une règle et une variable, f suffit largement). Pour les termes de cette fonction, nous avons vu en cours 3 méthodes:

- la *corps-skolemisation* utilise toutes les variables qui sont dans le *corps positif* de la règle;
- la *frontière-skolemisation* utilise toutes les variables de la *frontière de la règle* (celles qui sont communes à la tête et au corps négatif);
- la *pièce-skolemisation*, qui est plus compliquée.

Ensuite, il n'y a plus qu'à remplacer chaque variable existentielle par son terme fonctionnel associé. Ici, les trois skolémisations produisaient la même chose:

```
p(a).
s(X, f(X)) :- p(X), not ( q(X, Y), r(Y) ).
```

Mise sous forme normale: un corps négatif est *normal* quand toutes ses variables apparaissent dans le corps positif. Une règle est *normale* quand tous ses corps négatifs sont normaux. Ici, la règle n'est pas normale car la variable Y du corps négatif n'est pas dans le corps positif.

Si on a un corps négatif anormal, on peut le normaliser de la façon suivante: on associe un *nouveau* (fresh) nom de prédicat m à ce corps négatif, et on considère le tuple $\vec{X}$ de variables qui

sont à la fois dans le corps positif et ce corps négatif. On remplace le corps négatif par $m(\vec{X})$ (en gardant bien le `not`). On rajoute une règle dont le corps est le corps négatif original (le `not` disparaît) est dont la tête est $m(\vec{X})$. Ici on obtient:

```
p(a).
s(X, f(X)) :- p(X), not m(X).
m(X) :- q(X, Y), r(Y).
```

Instanciation (grounding): Il faut calculer maintenant le *domaine de Herbrand* $\mathcal{H}$. Vous l'initialisez avec toutes les constantes de la base de connaissances (celles des faits, et éventuellement celles des règles), puis calculez le plus petit ensemble tel que: si f est une fonction d'arité k (issue de votre skolémisation) et $\vec{h}$ un k-tuple d'éléments de $\mathcal{H}$, alors $f(\vec{h})$ est aussi un élément de $\mathcal{H}$. Ici, nous obtenions:

$$\mathcal{H} = \{a, f(a), f(f(a)), f(f(f(a))), \cdots\}$$

Ensuite, il faut instancier (ce qui peut être infini) chaque règle par l'ensemble de règles obtenues en substituant de toutes les façons possibles chacune de ses variables par des termes de $\mathcal{H}$. Ici nous aurons la base de connaissances avec un nombre infini de règles:

```
p(a).
s(a, f(a) :- p(a), not m(a).
s(f(a)), f(f(a))) :- p(f(a)), not m(f(a)).
...
m(a) :- q(a, a), r(a).
m(a) :- q(a, f(a)), r(f(a)).
m(f(a)) :- q(f(a), a), r(a).
...
```

Une réponse suffisante pourrait être:

Skolemisation:

```
p(a).
s(X, f(X)) :- p(X), not ( q(X, Y), r(Y) ).
```

Mise sous forme normale:

```
p(a).
s(X, f(X)) :- p(X), not m(X).
m(X) :- q(X, Y), r(Y).
```

Instanciation: Le domaine de Herbrand

$$\mathcal{H} = \{a, f(a), f(f(a)), f(f(f(a))), \cdots\}$$

est infini, nous avons donc l'instanciation infinie suivante:

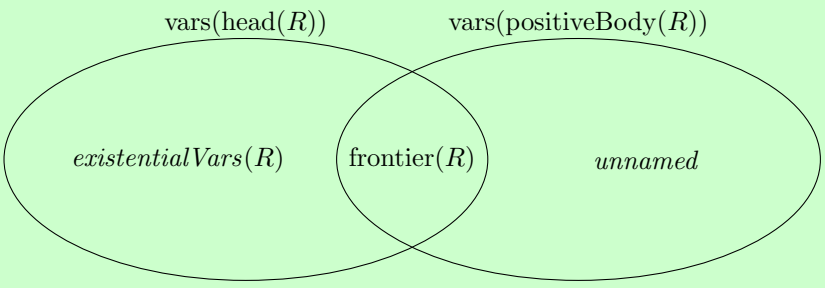
```
p(a).
s(a, f(a) :- p(a), not m(a).
s(f(a)), f(f(a))) :- p(f(a)), not m(f(a)).
...
m(a) :- q(a, a), r(a).
m(a) :- q(a, f(a)), r(f(a)).
m(f(a)) :- q(f(a), a), r(a).
...
```

La petite boutique des horreurs, épisode 2: je ne pensais pas voir autant d'inventivité dans des skolémisations qui ne marchent pas.

- il y a tout d'abord les 3 étudiants qui ont répondu à cette question avec le programme de la question 1, mais en faisant quelque chose comme ça: si on se donne une règle `p(X, Y) :- q(X), not r(X, Z)`, alors leur skolémisation obtenue est $p(X,Y) \rightarrow q(X), not\ r(X, f(X))$.
  - alors soit ils n'ont pas compris que le `h :-b` en "notation prolog" se traduit par un $b \rightarrow h$ en "notation logique" (ce que je pourrai regretter, mais admettre), mais dans ce cas ils n'ont pas compris qu'*il n'y a jamais de* `not` *dans une tête de règle*;
  - soit c'est juste une étourderie dans le sens de la flèche $\leftarrow$ traduisant `:-` qui devient machinalement un $\rightarrow$, mais dans ce cas ils n'ont rien compris à l'étape de skolémisation qui doit skolémiser les variables existentielles (dans la tête), et pas les variables "anormales" des corps négatifs (celles qui sont dans un corps négatif mais pas dans le corps positif).

  Bref, ces trois étudiants m'ont fait entrer dans un univers alternatif de la correction, où non seulement on mélange les questions posées mais aussi toutes les notions de base du cours...

- il y a aussi les nombreux étudiants qui ont répondu, comme je le fais parfois sur mes corrections, *"déjà skolémisé"*. J'admets que la remarque suffit, quand c'est effectivement le cas. Mais ici ce n'est juste pas vrai. Et je me demande alors ce qu'ils ont compris de la skolémisation. Cas particulier, un étudiant a justifié la non nécessité de skolémiser par l'argument suivant (je paraphrase) *"il n'y a pas de variables existentielles car il n'y a pas de* $\exists X$ *dans la tête de la règle"*. Je dois alors rappeler que la définition d'une variable existentielle est "*une variable qui est dans la tête mais pas dans le corps (positif)*". La notation $h(X) \rightarrow \exists Y p(X,Y)$ est redondante, et a été adoptée par Marie-Laure Mugnier pour *souligner* que Y est une variable existentielle, à l'attention de nos collègues qui viennent du monde ASP et qui, intuitivement, les comprendraient comme des variables universelles ou de nos étudiants qui ont du mal à calculer l'intersection de deux petits ensembles. Et même si, par souci d'uniformité, j'adoptais cette convention pour la "notation logique" des règles, j'aurais du mal dans la "notation prolog" `p(X, Y) :- h(X)`, en ASCII.



- enfin, parmi ceux qui ont effectivement skolemisé la variable existentielle de la tête de la règle, j'ai quand même pu voir des méthodes aussi fausses qu'étranges:

– il y a ceux qui suppriment purement et simplement la variable existentielle ! Ainsi `p(X, Y) :- h(X)` devient `p(X) :- h(X)`. Ca a l'avantage de la simplicité, mais c'est du grand n'importe quoi, une raison suffisante étant qu'on transforme un prédicat binaire en prédicat unaire...

– il y a ceux (trop nombreux) qui utilisent les variables des corps négatifs comme variables de la fonction de Skolem. Ceci ne fait pas partie des 3 méthodes vues en cours (et rappelées ci-dessus), et donne un résultat surprenant. Prenons par exemple `p(X, Y) :- q(X), not r(X, Z)`. Cette "skolémisation" produirait `p(X, f(X, Z)) :- q(X), not r(X, Z)`. Et ceci n'a aucun sens, puisque $f(X, Z)$ serait fonction de l'image qui n'existe pas de la variable Z !

– enfin, il y a ceux qui skolémisent les variables anormales du corps négatif. J'avais bien dit, plusieurs fois, qu'il ne fallait pas. Car si on skolémise (toujours le même mini-example) par `p(X, f(X)) :- q(X), not r(X, g(X))`, cette règle "skolémisée" ne sera pas applicable sur `q(a), r(a, a)`, alors que la règle initiale l'est !

La petite boutique des horreurs, épisode 3: je croyais avoir tout vu avec la skolémisation, mais la mise sous forme normale me réservait encore des surprises. La bonne nouvelle, c'est que tout le monde a vu qu'il fallait normaliser. La mauvaise, c'est que la façon dont la normalisation est faite montre que vous n'avez peut-être pas tous compris pourquoi il faut normaliser...

```
p(a).
s(X, Z) :- p(X), not ( q(X, Y), r(Y) ).
```

Tout d'abord, il fallait bien considérer que, dans la règle proposée, $q(X, Y), r(Y)$ était un unique corps négatif (ce que j'avais souligné en le parenthésant, comme promis en cours). Ensuite il fallait bien comprendre que ce corps négatif était anormal *pour la seule présence de la variable* Y. Petite tentative de compréhension et de critique de différentes réponses que j'ai pu voir...

```
p(a).
[R1] s(X, Z) :- p(X), not m(X, Y).
[R2] m(X, Y) :- q(X, Y), r(Y).
```

Ici, il y a au moins un point positif, c'est que la transformation est correcte (elle préserve les raisonnements). Le problème, c'est que la règle `R1` n'est toujours pas sous forme normale (la variable Y n'est pas dans le corps positif). Donc le programme a juste été inutilement complexifié.

```
p(a).
[R1] s(X, Z) :- p(X), not ( m(X), r(*) ).
[R2] m(X) :- q(X, Y).
```

Il y a aussi ceux qui, comme ci-dessus, ont créé un prédicat intermédiaire qui ne portait que sur le premier atome. Pourquoi ? Les deux contenaient la "variable anormale" Y. Alors pourquoi pas les deux ? J'avoue n'avoir aucune idée de la justification, mais en tout cas ça ne marche pas:

- il y a ceux qui ont gardé la variable $* = Y$, dans ce cas il y a deux problèmes, dont chacun est suffisant: *(i)* le programme obtenu n'est toujours pas sous forme normale; et *(ii)* on perd la jointure nécessaire entre la variable * et la variable Y à l'intérieur du corps de la règle.
- il y a ceux qui ont renommé la variable en $* = X$ pour obtenir un truc sous forme normale. Mais le problème *(ii)* du cas précédent est toujours d'actualité.

La petite boutique des horreurs, épisode 4: Ici peu de choses vraiment surprenantes, à part la construction du domaine de Herbrand. Sinon, vous arrivez à faire des substitutions correctes.

- quand vous n'avez pas skolémisé, le domaine de Herbrand devrait être $\mathcal{H} = \{a\}$, et donc vous obtenez 1 ou deux règles (suivant que vous ayez normalisé ou pas).
- il y a ceux qui ont skolémisé, mais qui ont laissé le domaine de Herbrand à $\mathcal{H} = \{a\}$. Un peu de cohérence s'il vous plait.
- je ne sais plus si il avait skolémisé ou pas, mais un étdudiant m'a donné $\mathcal{H} = \{a, b, c\}$. Certainement parce qu'il a vu que a ne suffisait pas pour exprimer la variable existentielle, mais alors pourquoi 3 éléments ? Et pas 2 ? Ou une infinité ? En tout cas, ça ne marche pas.

Question 3 On considère le programme propositionnel suivant.

```
a.
e :- a, d, not (b, c).
b :- a, not e.
c :- d, not b.
d :- a, not f.
f :- b, not d.
```

La première chose à faire, c'est de nommer tous les objets du programme Π pour simplifier les explications.

```
[F] a.
[R1] e :- a, d, not (b, c).
[R2] b :- a, not e.
[R3] c :- d, not b.
[R4] d :- a, not f.
[R5] f :- b, not d.
```

1. dire, en utilisant la méthode du point fixe, si $\{a, c, d, e\}$ est un modèle stable du programme
2. dire, en utilisant la méthode su point fixe, si $\{a, b, f\}$ est un modèle stable du programme
3. dire si $\{d, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)
4. dire si $\{a, c, d, e, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)
5. dire si $\{a, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)
6. sans dessiner le graphe de dépendance des prédicats, déduire de ce qui précède que le programme n'est pas stratifiable.

Certainement la partie la mieux réussie, mais le `not (b, c)` a posé quelques problèmes.

1. dire, en utilisant la méthode du point fixe, si $\{a, c, d, e\}$ est un modèle stable du programme

1. $\mathbf{E_1} = \{\mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}$
On calcule tout d'abord le programme réduit $\Pi_{|E_1}$.

```
[F] a.  % on garde toujours les faits
[R1] e :- a, d.  %, not (b, c).  on positive car {b, c} ⊈ E1
%[R2] b :- a, not e.  on supprime car {e} ⊆ E1
[R3] c :- d.  %, not b.  on positive car {b} ⊈ E1
[R4] d :- a.  %, not f.  on positive car {f} ⊈ E1
%[R5] f :- b, not d.  on supprime car {d} ⊆ E1
```

Ensuite on calcule $(\Pi_{|E_1})^* = \{a, c, d, e\}$ la saturation de ce programme. On a $(\Pi_{|E_1})^* = E_1$, et on en déduit que E_1 est un modèle stable de Π.

Rappel du cours un petit rappel du cours avant de passer à la critique de vos productions. Pour prouver que E (un ensemble d'atomes propositionnels) est un modèle stable d'un programme Π par la méthode du point fixe, on commence par construire le programme réduit $\Pi_{|E}$, qui est un programme positif (donc sans négation dans les règles). Comme c'est un programme propositionnel positif, on peut donc parler de *sa* saturation $(\Pi_{|E})^*$. Et le théorème vu en cours nous dit que E est un modèle stable si et seulement si $E = (\Pi_{|E})^*$.

Construction du programme réduit (version faible): je vous ai donné en cours une seule version de cette construction. C'est ce que j'appelle la *version faible* du programme réduit. Elle diffère de la *version forte* (les noms sont entièrement de mon fait, et donc ne les utilisez pas en dehors des examens que je devrai corriger) en ce qu'elle ne s'intéresse pas au corps positif des règles. Pour chaque règle:

- si il existe un corps négatif $not\,(a1, \cdots, a_k)$ qui bloquerait dans E un déclenchement éventuel de la règle, alors on *supprime* la règle. Remarquons que cette condition s'exprime plus simplement par $\{a_1, \cdots, a_k\} \subseteq E$.
- si il n'existe aucun corps négatif qui bloquerait dans E un déclenchement éventuel de la règle, alors on *positive* la règle, c'est à dire qu'on lui enlève tous ses corps négatifs.

Construction du programme réduit (version forte): cette fois-ci, on ne regarde pas seulement les corps négatifs des règles, mais on s'intéresse également au corps positif. Le principe est le suivant:

- si la règle n'est pas déclenchable dans E, alors elle n'est déclenchable dans aucun sous-ensemble de E, et en particulier dans aucun des ensembles d'atomes obtenus au cours de la saturation $(\Pi_{|E})^*$. En conséquence, si son corps positif C n'est pas inclus dans E, alors on peut supprimer la règle.
- sinon, on procède comme dans la version faible.

Alors, me direz-vous, si la version forte n'est pas vraiment plus compliquée à comprendre (où à implémenter) que la version faible, et qu'elle peut donner des programmes réduits plus petits (ce qui n'est pas le cas pour cette question 1, mais le sera pour la question 2), pourquoi ne pas avoir directement donné la version forte en cours ? Je n'ai pas le temps de vraiment développer cette question ici, et donc je dirai juste que ces deux versions ont des propriétés subtilement différentes (et moins intuitives dans le cas de la version forte). Pour plus d'information, vous pouvez vous référer à la question 9 de l'examen de janvier 2025 `https://www.lirmm.fr/~baget/docs/hai933i/2025.01.correction.pdf`

La petite boutique des horreurs, épisode 5: Dans l'ensemble, cette partie était meilleure et je n'y ai pas vu (ou j'ai oublié depuis hier) de choses *vraiment* abominables. Voici cependant quelques erreurs récurrentes.

Il y a d'abord tous ceux qui m'ont écrit quelque chose comme *"par la méthode du point fixe, je prouve que E_1 est un modèle stable"*. Point, exercice suivant. Si je vous demande une méthode, je veux pouvoir vérifier que cette méthode a bien été appliquée, et ici ça passe par l'écriture du programme réduit. Je suis peut-être un peu trop old school, mais je n'ai pas à faire confiance à celui qui m'affirme, en mode "tkt mon frère", qu'il a bien fait ma méthode et qu'il a trouvé le résultat (surtout quand ce résultat est un booléen).

Le plus gros problème a été, pour la majorité des étudiants, la gestion d'un corps négatif contenant 2 atomes dans la règle R_1. Il vous a ainsi suffi de trouver un des deux atomes dans E_1 pour en déduire que la règle était bloquée, et donc la supprimer. Pourtant, c'est exactement le thème du premier exercice "application de règle" du premier cours des modèles stables: si un corps négatif contient 2 atomes, alors on bloque si on trouve un homomorphisme *des 2 atomes* qui étend le déclencheur dans la base de faits. En version propositionnelle, ça se traduit naturellement (puisqu'on n'a pas besoin d'homomorphisme) par: on trouve les 2 atomes dans la base de faits.

Un truc qui ne me plaît vraiment pas: parmi les étudiants qui ont supprimé par erreur la règle R_1, beaucoup m'ont assuré, la main sur le coeur promis juré craché, qu'ils avaient trouvé E_1 par la saturation et donc que E_1 était un modèle stable. Or la règle générant e avait disparu, et votre affirmation était trivialement fausse. Et comme je suppose que, en M2, vous pouvez sans vous tromper saturer un petit programme propositionnel de 2 règles, ça veut dire que vous essayez juste de cacher une erreur que vous ne trouvez pas sous le tapis. Ceci n'a pas sa place en sciences.

Enfin, une remarque plus cosmétique. J'ai parfois eu du mal, dans votre programme réduit, à voir ce qui était supprimé ou positivé. J'ai déjà indiqué à l'une d'entre vous que ça aurait pu lui coûter des points. En peu de pitié pour le correcteur! Vous ne pouvez pas, comme dans ce document, commenter ce que vous souhaitez supprimer, mais vous pouvez le rayer, comme je le faisais au tableau. Et vous aurez mon éternelle reconnaissance si, en plus, vous le rayez d'une couleur différente (pour le différencier de vos erreurs), et que vous rajoutez un micro commentaire comme je l'ai fait ici.

2. dire, en utilisant la méthode su point fixe, si $\{a, b, f\}$ est un modèle stable du programme

2. $\mathbf{E_1 = \{a, b, f\}}$
On calcule tout d'abord le programme réduit $\Pi_{|E_2}$.

```
a.  % on garde car c'est un fait
e :- a, d.  %, not (b, c).% on positive car {b, c} ⊈ E2
b :- a.  %, not e.  % on positive car {e} ⊈ E2
%c :- d, not b.  % on supprime car {b} ⊆ E2
%d :- a, not f.  % on supprime car {f} ⊆ E2
f :- b.  %, not d.  % on positive car {d} ⊈ E2
```

Ensuite on calcule $(\Pi_{|E_2})^* = \{a, b, f\}$ la saturation de ce programme. On a $(\Pi_{|E_2})^* = E_2$, et on en déduit que E_2 est un modèle stable de Π.

Cette fois-ci, on avait un résultat légèrement différent en utilisant la version forte du programme réduit, comme le montre la réponse alternative suivante.

2. $\mathbf{E_2} = \{\mathbf{a}, \mathbf{b}, \mathbf{f}\}$
On calcule tout d'abord le programme réduit $\Pi_{|E_2}$.

```
a.  % on garde car c'est un fait
%e :- a, d, not (b, c).  % on supprime car {a, d} ⊈ E2 (version forte)
b :- a.  %, not e.  % on positive car {e} ⊈ E2
%c :- d, not b.  % on supprime car {b} ⊆ E2
%d :- a, not f.  % on supprime car {f} ⊆ E2
f :- b.  %, not d.  % on positive car {d} ⊈ E2
```

Ensuite on calcule $(\Pi_{|E_2})^* = \{a, b, f\}$ la saturation de ce programme. On a $(\Pi_{|E_2})^* = E_2$, et on en déduit que E_2 est un modèle stable de Π.

Un problème de conscience pour le correcteur: ici, comme à la question précédente, la grande majorité des étudiants a supprimé la règle R_1. J'aurais pu considérer qu'ils avaient raison de le faire, puisque la version forte l'y autorise. Bon, elle n'était pas vue en cours, mais vous auriez pu la découvrir en parcourant mes archives (ou dans les cours de Michel Leclère). J'ai compté juste quand vous aviez bien justifié la suppression de la règle ou quand vous l'aviez gardée à la question 1 et supprimée à la question 2. Sinon, j'ai enlevé un quart de point en considérant que vous aviez raison, mais par hasard.

3. dire si $\{d, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)

3. $\mathbf{E_3} = \{\mathbf{d}, \mathbf{f}\}$
L'atome a est un fait de Π. Il sera donc dans tous les programmes réduits, donc dans tous les saturés de programme réduit, donc dans tout les modèles stables. Comme $a \notin E_3$, E_3 n'est pas un modèle stable.

La question 3 a été en général bien faite, même si les explications étaient parfois laborieuses.

Pour aller plus loin: cet argument disant que les faits sont présents dans tous les modèles stables est un cas particulier de la notion d'*atomes garantis* sur lesquels vos camarades avaient travaillé l'année dernière (exercice 5) `https://www.lirmm.fr/~baget/docs/hai933i/2024.12.correction.pdf`.

Pourtant, dans cette question comme dans la 4 et 5, certains ont systématiquement continué à appliquer la méthode du point fixe (parfois en supplément d'un autre argument). Ceci appelle deux remarques:

- ne donnez jamais deux arguments distincts pour une même réponse! Si les deux sont justes, ça ne donne pas plus de points. Si l'un des deux est faux, ça montre que vous n'avez pas bien compris et tout devient faux.
- la formulation des questions 3, 4 et 5 montrait bien que, cette fois, je n'attendais pas la méthode du point fixe qui prend du temps. Appliquer systématiquement cette méthode peut vous rassurer, mais vous fait perdre beaucoup de temps... Ne venez pas vous plaindre après que le sujet est trop long...

4. dire si $\{a, c, d, e, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)

4. $\mathbf{E_4} = \{\mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}\}$
Nous avons $E_1 \subseteq E_4$. Or E_1 est un modèle stable, et nous savons que ceux-ci sont maximaux par inclusion. Donc E_4 n'est pas un modèle stable.

Ici, plusieurs étudiants ont utilisé un autre argument, correct mais souvent avancé de façon maladroite. Je pouvais accepter quelque chose ressemblant à ceci:

4. (version 2)
La règle R_5 est la seule pouvant générer l'atome f. Or cette règle est supprimée dans le programme réduit $\Pi_{|E_4}$ (car $\{d\} \subseteq E_4$). Donc la saturation $(\Pi_{|E_4})^*$ ne contiendra pas f et ne sera pas un modèle stable.

Cette explication a l'intérêt d'utiliser les propriétés du programme réduit, sans le calculer entièrement. Une autre version, utilisant cette fois-ci les dérivations persistantes et complètes, aurait pu être:

4. (version 3)
Supposons que E_4 est un modèle stable, alors il existe une dérivation persistante et complète dont il est le résultat. L'atome f n'a pu être produit que par la règle R_5. On sait que $d \in E_4$. Donc aucune application de R_5 ne peut être persistante. E_4 n'est donc pas un modèle stable.

Regardez attentivement ces petites démonstration 4.(version 2) et 4.(version 3). Ce sont des "patterns de preuve" que vous pouvez réutiliser. Il est important de lire et de travailler de telles preuves, car les votres sont souvent maladroites. De nombreuses preuves similaires peuvent être trouvées dans les corrections `https://www.lirmm.fr/~baget/docs/hai933i/2024.12.correction.pdf` et `https://www.lirmm.fr/~baget/docs/hai933i/2025.01.correction.pdf`, si vous souhaitez vous entrainer.

5. dire si $\{a, f\}$ est un modèle stable du programme, par la méthode de votre choix (qui devra être soigneusement justifiée)

5. $\mathbf{E_5} = \{\mathbf{a}, \mathbf{f}\}$
Nous avons $E_5 \subseteq E_2$. Supposons E_5 modèle stable. Alors, puisque les modèles stables sont maximaux, E_2 ne serait pas un modèle stable, ce qui est absurde puisque nous avons prouvé (2.) qu'il en était un. E_5 n'est donc pas un modèle stable.

Cette question a été un peu moins bien faite que la précédente, puisque plusieurs étudiants ont fait la même démonstration directe qu'à la Q4, outrepassant ainsi ce qu'affirmait le théorème vu en cours. Mais je pense avoir fait cette preuve au moins 2 fois au tableau. D'autres ont essayé d'autres idées: le point fixe, bien sûr (la méthode magique que vous voulez essayer à toutes les sauces, un peu comme les tables de vérité que découvrent les étudiants avec la logique des propositions), mais aussi d'autres schémas de preuve que j'essaye de préciser ici.

5. (version 2)
La règle R_5 est la seule permettant de générer l'atome f dans E_5. Il faut donc ne pas supprimer cette règle dans le programme réduit. Or la version forte la supprime, puisque b n'est pas dans E_5. L'ensemble E_5 n'est donc pas un modèle stable.

5. (version 3)
La règle R_5 est la seule permettant de générer l'atome f dans E_5. Considérons une dérivation persistante et complète qui produit E_5. Alors R_5 doit nécessairement avoir été appliquée. Mais elle ne peut être appliquée que sur un ensemble d'atomes contenant son corps positif b, et b sera alors produit par la dérivation, qui ne produit donc pas E_5. Alors E_5 n'est pas un modèle stable.

6. sans dessiner le graphe de dépendance des prédicats, déduire de ce qui précède que le programme n'est pas stratifiable.

6. Nous avons vu que le programme avait au moins deux modèles stables, E_1 et E_2. Or nous avons vu en cours qu'un programme stratifiable avait un et un seul modèle stable. Donc le programme n'est pas stratifiable.

Ici, je n'ai pas vraiment vu d'horreur, mais plutôt des maladresses.

Non respect des consignes: plusieurs, pour une raison ou une autre, n'ont pas respecté la consigne "sans dessiner le graphe de dépendances". Alors ils ont déguisé plus ou moins ça en parlant des corps négatifs qui bloquaient des corps positifs et finissaient par exhiber un circuit, mais c'était du bla-bla pour éviter de dessiner le graphe. Ca respectait effectivement la lettre de la consigne, mais pas l'esprit: je ne voulais pas savoir si vous pouviez calculer la stratifiabilité, mais si vous connaissiez le rapport de cette notion avec les modèles stables.

Le (mauvais) raisonnement statistique J'ai vu des assertions du type "comme nous l'avons vu dans les questions précédentes, le programme n'a pas de (ou n'a qu'un) modèle(s) stable(s)". Mauvaise nouvelle: vous avez seulement exploré 5 ensembles d'atomes possibles, il en reste $2^6 - 5$ à explorer. Vous n'avez donc exploré que 7.8% des cas. Vous ne pouvez donc pas déduire le nombre de modèles stables du minime échantillon proposé dans les questions 1 à 5.

Le cas où vous avez trouvé un modèle stable: vous ne pouvez doublement pas conclure. Tout d'abord pour la raison précédente (il peut y en avoir plein d'autres que vous n'avez pas vus). Ensuite parce que le théorème dit: "si le programme est stratifiable, alors il admet un et un seul modèle stable". Bien entendu, la *contraposée* est toujours vraie: "si le nombre de modèles stables est différent de 1, alors le programme n'est pas stratifiable". C'est ce que j'ai utilisé dans ma réponse (j'ai trouvé deux MS, il y en a peut-être plus, mais c'est en tout cas différent de 1). Par contre, j'avais lourdement insisté en cours sur le fait que la *réciproque* n'était pas vraie en général. Donc, même si vous aviez pu prouver, d'une manière ou d'une autre (et dans ce cas la meilleure méthode reste ASPERIX), qu'il y a exactement un modèle stable, vous n'auriez pas pu conclure.

Question 4 En utilisant l'algorithme ASPERIX (dans sa version propositionnelle), donnez tous les modèles stables du programme de la question 3.

Tout d'abord, toutes mes excuses: le graphe ASPERIX complet était plus gros que prévu, et donc l'objectif de donner *tous* les modèles stables du programme était un peu ambitieux. J'ai donc classé vos productions en trois niveaux approximatifs:

- ceux qui arrivent à développer un arbre sans trop se tromper;
- ceux qui arrivent à le faire sans se tromper, même sur le cas difficile (R_1);
- ceux qui, en plus, ont développé une branche complète et l'ont correctement analysée.

J'ai cependant vu de grosses erreurs dans le développement de l'arbre, aussi on va commencer par de petits rappels.

Rappel: évaluation d'une règle sur une feuille de l'arbre Chaque sommet de l'arbre est représenté par un rectangle divisé en trois parties: de gauche à droite, IN, OUT, MBT.

- la partie IN contient un ensemble d'atomes et représente ce qui a été prouvé;
- la partie OUT contient un ensemble de contraintes et représente ce qui est interdit;
- la partie MBT contient un ensmble de disjonctions de contraintes et représente ce qui reste à prouver.

Pour gagner en lisibilité, je n'indique sur chaque sommet que les éléments qui ont été créés à l'étape de création de ce sommet. On lira donc le contenu d'un champ d'un sommet en faisant l'union des contenus des champs de même type sur tous ses ancêtres.

Evaluabilité d'une règle sur une feuille Soit s une feuille de l'arbre ASPERIX. Une règle R est évaluable sur s quand il y a un déclencheur non bloqué σ de R dans $\text{IN}(s)$. Dans ce cas, on peut évaluer la règle. C'est à dire quand elle est applicable au sens du premier cours MS `https://www.lirmm.fr/~baget/docs/hai933i/cours1.pdf` (slide 12), comme expliqué dans l'exercice qui suivait (slide 14).

Si la règle R est évaluable sur s suivant σ, alors on peut maintenant l'évaluer. Notons au passage que si plusieurs évaluations sont possibles sur s, alors il faut en choisir une. Après l'évaluation, s ne sera plus une feuille, condition nécessaire pour l'évaluation. Mais ces évaluations non choisies seront peut-être encore évaluables sur les feuilles du sous-arbre issu de s. Au vu de certaines erreurs sur vos copies, j'ai décidé d'adopter ici une nouvelle représentation de l'arbre d'évaluation. J'espère qu'elle évitera à l'avenir ces erreurs. Vous me direz ce que vous pensez de cette nouvelle version, si vous la trouvez mieux, je l'adopterai à l'avenir (et je devrai refaire mes slides).

Structure induite par l'évaluation Lorsque j'évalue R sur s suivant σ, je crée tout d'abord un fils r de s, qui identifie à la fois la règle R et l'homorphisme σ utilisé. C'est ce qui est nouveau dans cette représentation. J'espère que ça va vous forcer à comprendre que le fils gauche et le fils droit doivent être issus d'une même évaluation. Ce sommet/évaluation , est représenté ici par un ovale. Le sommet/évaluation a 2 successeurs (1 seul si la règle est positive, c'est à dire ne contient pas de corps négatifs). Le ou les successeurs sont des sommets/état (contenant les 3 champs IN, OUT et MBT). L'unique successeur dans le cas d'une règle positive représente l'application de la règle. Sinon, par convention, le successeur gauche représente l'application de la règle et le successeur droit sa non application.

Sommet représentant l'application de règle (sommet gauche) On rajoute au champ IN les atomes de $\sigma^{safe}(tete(R))$. Puis, pour chaque corps négatif C de la règle (qui est une conjonction), on rajoute dans le champ OUT la contrainte (C, σ), symbolisant qu'il ne fait pas trouver dans IN un homomorphisme de C qui étend σ. Dans le cas d'une règle positive (pas de corps négatif), ce champ OUT reste vide. On ne rajoute rien au champ MBT. Alors cette notion de contrainte est un peu compliquée, mais c'est le cas général. On peut parfois l'exprimer de façon plus simple. Supposons une règle avec deux corps négatifs $C_1 = p(X,Y), r(X)$ et $C_2 = q(Z,Y), r(Y)$. Je vous conseille *fortement* de suivre l'explication suivante avec papier/crayon en main. Examinons ce qu'on rajoute au champ OUT suivant différents homomorphismes σ:

- **Si toutes les valeurs de σ sont des constantes** Si $\sigma = \{X \to a, Z \to b\}$, alors on rajoute à OUT les contraintes $(p(X,Y) \wedge r(X), \sigma)$ et $(q(Z,Y) \wedge r(Y), \sigma)$. Mais ces contraintes sont complètement équivalentes (vérifiez le) à $(p(a,Y) \wedge r(a), \emptyset)$ et $(q(b,Y) \wedge r(Y), \emptyset)$, que l'on

peut simplifier en $p(a, Y) \wedge r(a)$ et $q(b, Y) \wedge r(Y)$. On peut donc rajouter au champ OUT $p(a, Y) \wedge r(a)$, $q(b, Y) \wedge r(Y)$. Remarquez l'utilité de mon introduction de la notation $\wedge$: la virgule sépare les contraintes, le $\wedge$ sépare les atomes qui sont dans une même contrainte (et qui, pour obtenir une violation de contrainte, devront être *tous* envoyés dans IN par homomorphisme). Nous erons dans ce cas quand la base de faits ne contient pas de variables et si les règles ne créent pas de variables: c'est à dire soit ce sont des règles datalog, soit ce sont des règles propositionnelles que l'on a skolémisées (donc ça peut être intéressant de le faire pour avoir des représentations plus lisibles).

- **Sinon, on ne peut pas (autant) simplifier** et on a dans ce cas bien besoin de la notion de contraintes. En effet, prenons $\sigma = \{X \to a, Z \to B\}$ (cette fois-ci, l'image de Z n'est pas une constante mais une variable). Imaginons qu'on fasse la même simplification que dans le cas précédent pur obtenir $p(a, Y) \wedge r(a)$, $q(B, Y) \wedge r(Y)$. Alors si on a $q(C, D), r(D)$ dans IN, il y aura violation de la deuxième contrainte (avec $\{B \to C, Y \to D\}$). Et ce n'est pas ce que l'on veut ! Car ce que l'on voulait, c'est interdire tout $q(*, Y), r(Y)$ mais avec $* = B$! On est donc obligés de garder l'homomorphisme à étendre, même si on peut simplifier au maximum avec par exemple $p(a, Y) \wedge r(a)$, $(q(Z, Y) \wedge r(Y), \{Z \to B\})$.

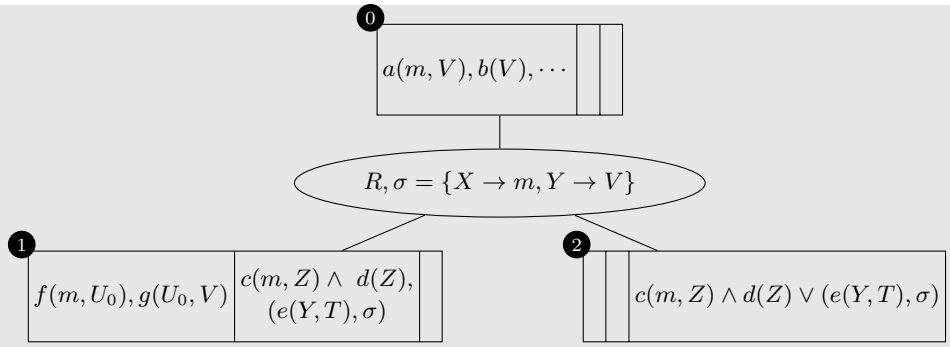
Sommet représentant la non application de la règle (sommet droit) Pour avoir eu raison de ne pas appliquer, il faut qu'au moins un des corps négatifs bloque à un moment dans le sous-abre de s. Le champ IN reste inchangé (on n'a pas appliqué, donc on ne déduit rien), le champ OUT non plus, mais on va ajouter au champ MBT une disjonction des contraintes (C_i, σ), pour chaque corps négatif C_i de la règle. Attention, lorsque je lis `not (a, b, c)` dans une règle, c'est bien `a, b, c` que j'appelle corps négatif: le symbole `not` n'est qu'un moyen syntaxique de l'identifier comme un corps négatif (ne rigolez pas, j'ai trop souvent vu l'erreur). Comme dans le sommet gauche, on va pouvoir simplifier l'écriture lorsque toutes les images du déclencheur sont des constantes.

- **Si toutes les valeurs de σ sont des constantes** Si $\sigma = \{X \to a, Z \to b\}$, alors on rajoute à MBT la disjonction $(p(X, Y) \wedge r(X), \sigma) \vee (q(Z, Y) \wedge r(Y), \sigma)$. Comme dans le cas précédent, on peut simplifier l'écriture des contraintes pour obtenir $p(a, Y) \wedge r(a) \vee q(b, Y) \wedge r(Y)$. Dans le cas où il n'y a qu'un corps négatif, la disjonction de taille 1 est réduite à une seule contrainte, et on n'aura pas beoin du symbole $\vee$.
- **Sinon, on ne peut pas (autant) simplifier** Si $\sigma = \{X \to a, Z \to B\}$, on est, comme pour le fils gauche, obligés de garder l'homomorphisme à étendre, mais on peut également simplifier au maximum avec $p(a, Y) \wedge r(a), \vee(q(Z, Y) \wedge r(Y), \{Z \to B\})$.

Bon, désolé d'avoir été aussi pontilleux sur la théorie, mais je pense que ce qui précède est à bien comprendre. Pour se reposer (bon, pas moi, je dois taper le tikz), on va voir les conséquences de ce qui précède sur quelques exemples, du plus compliqué au plus simple (car oui, je commence par le cas général et on voit ce que ça donne dans les cas où ça peut se simplifier).

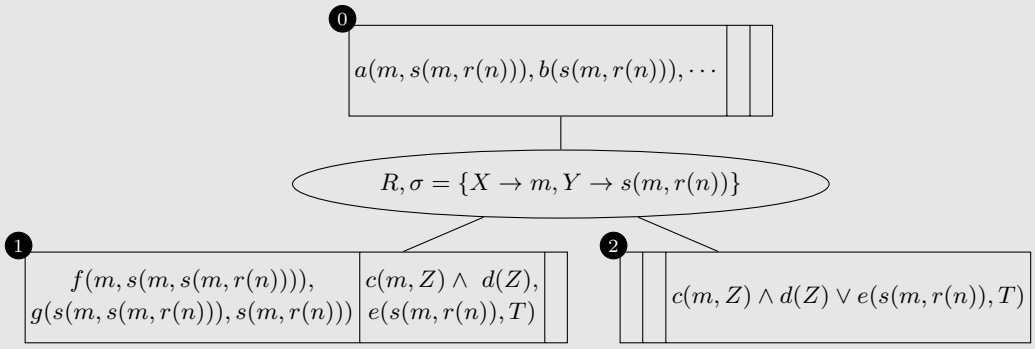
Exemple 1: règle existentielle non skolémisée On se donne la règle R représentée ci-dessous, et un sommet feuille d'identifiant 0 telle que R est évaluable suivant σ dans son champ IN. Supposons $\sigma = \{X \to m, Y \to V\}$. Notre critère d'évaluabilité veut dire que IN contient $a(m, V), b(V)$, mais ne contient pas quelque chose de la forme $c(m, *), d(*)$, ni quelque chose de la forme $e(V, *)$, ou plus formellement: il n'y a pas d'homomorphisme de $c(m, Z), d(Z)$ dans IN, ni d'homomorphisme de $e(Y, T)$ dans IN qui étend σ.

```
[R] f(X, U), g(U, Y) :- a(X, Y), b(Y), not ( c(X, Z), d(Z) ), not e(Y, T).
```



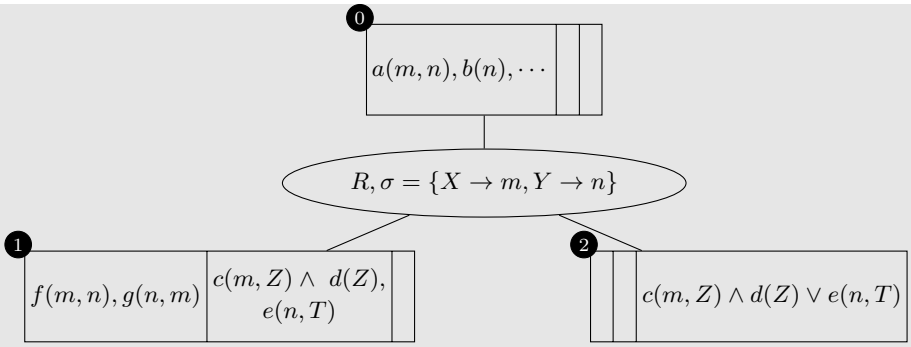
Exemple 2: règle existentielle skolémisée On considère maintenant que nos règles on été skolémisées, que la base de faits ne contient pas de variables, et donc que les images des déclencheurs ne seront jamais des variables. La règle représentée ci-dessous est la skolémisation de la règle précédente. La règle est évaluable suivant un déclencheur $\sigma = \{X \to m, Y \to s(m, r(n))\}$. Notre critère d'évaluabilité veut dire que IN contient $a(m, s(m, r(n))), b(s(m, r(n)))$, mais qu'il n'y a pas d'homomorphisme de $c(m, Z), d(Z)$ dans IN, ni d'homomorphisme de $e(s(m, r(n)), T)$ dans IN.

```
[R] f(X, s(X, Y)), g(s(X, Y), Y) :- a(X, Y), b(Y), not ( c(X, Z), d(Z) ), not e(X, T).
```



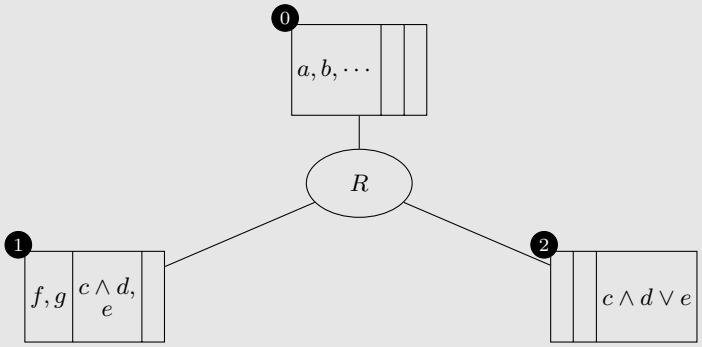
Exemple 3: règles datalog On considère maintenant que nos règles n'ont pas de variables existentielles, que la base de faits ne contient pas de variables, et donc que les images des déclencheurs sont toujours des constantes. Ca va alléger l'écriture. La règle ci-dessous est évaluable suivant un déclencheur $\sigma = \{X \to m, Y \to n\}$. Notre critère d'évaluabilité veut dire que IN contient $a(m, n), b(n)$, mais qu'il n'y a pas d'homomorphisme de $c(m, Z), d(Z)$ dans IN, ni d'homomorphisme de $e(n, T)$ dans IN.

```
[R] f(X, Y), g(Y, X) :- a(X, Y), b(Y), not ( c(X, Z), d(Z) ), not e(X, T).
```



Exemple 4: règles propositionnelles On considère maintenant que les faits et règles sont propositionnels (ce qu'on peut voir comme "tous les prédicats sont d'arité 0"). On suppose que la règle ci-dessous est évaluable (et il n'y a pas besoin de déclencheurs, puisque ce serait un déclencheur vide). Ceci veut dire que $\{a, b\} \subseteq \text{IN}$, que $\{c, d\} \not\subseteq \text{IN}$, et que $\{e\} \not\subseteq \text{IN}$.

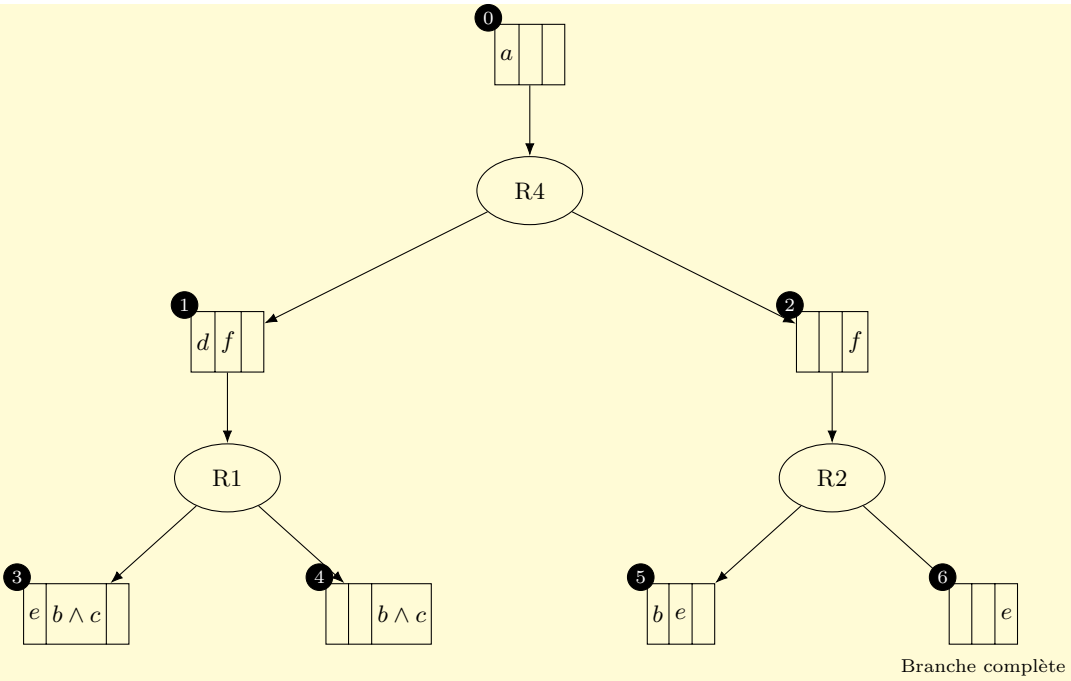
```
[R] f, g :- a, b, not ( c, d ), not e.
```



On a maintenant tous les outils en main pour commencer à construire l'arbre ASPERIX demandé dans l'exercice, dont je rappelle ici le programme.

```
[F] a.
[R1] e :- a, d, not (b, c).
[R2] b :- a, not e.
[R3] c :- d, not b.
[R4] d :- a, not f.
[R5] f :- b, not d.
```

Arbre ASPERIX, partie 1



Par exemple, ici, la règle `R1` peut s'évaluer sur le sommet 1 car son corps positif $\{a, d\} \subseteq$ IN, et son corps négatif $\{b, c\} \not\subseteq$ IN. On rappelle que, malgré notre représentation simplifiée, le IN du sommet 1 est l'union des IN de la racine jusqu'à ce sommet.

La petite boutique des horreurs, épisode 6 Avec les rappels que vous venez de lire attentivement, vous allez maintenant comprendre facilement pourquoi je me suis arraché les cheveux en regardant certains de vos arbres ASPERIX. Notez au passage qu'un simple début de développement de l'arbre bien réalisé comme ci-dessus vous aurait assuré la moitié des points. Il y a malheureusement eu beaucoup d'imprécisions, et surtout deux grosses erreurs que j'ai vu régulièrement.

- Certains, trop nombreux, avaient le vague souvenir qu'il fallait développer un arbre binaire, et l'ont fait en générant un fils gauche qui était le résultat de l'application d'une règle et un fils droit qui était le résultat de l'application d'une autre règle. Autant dire que ça génère n'importe quoi, qu'on n'est plus assuré de trouver tous les modèles stables (j'en suis certain), et je ne vois pas très bien ce que pourrait être le test pour vérifier si on a bien un modèle stable. C'est pour ne plus *jamais* voir ça que j'adopte ici une représentation qui met bien en lumière le mécanisme d'évaluation dans le dessin de l'arbre.

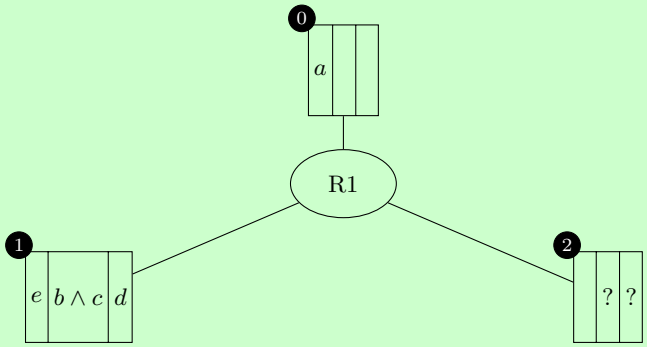
- Un truc que je ne m'explique pas, et que j'ai vu très souvent, est l'évaluation de la règle `R1` directement, parfois sur le sommet racine de l'arbre, alors que cette règle n'est même pas déclenchable. En effet, le corps positif de la règle $\{a, d\}$ n'est pas inclus à ce moment dans le IN. Mais ce n'est pas grave, vous avez quand même lancé l'évaluation. Puis, saisis de remords et d'effroi, vous vous êtes quand même rendu compte qu'il manquait un d quelque part. Alors, au choix, ce d a été soit supprimé, soit rajouté dans le IN, le OUT ou le MBT, du sommet gauche et/ou du sommet droit. Et, pendant la correction, je me suis dit que c'était du grand n'importe quoi. Mais le lendemain matin, la nuit étant censée porter conseil, je me suis dit qu'il y avait peut-être une solution pour faire marcher ça, et je me suis inquiété d'avoir arnaqué des étudiants...

Evaluer une règle non déclenchable On va se donner une règle R, non déclenchable sur une feuille donnée (mais quand même non bloquée parce qu'il ne faut pas exagérer), et on va essayer

de l'évaluer quand même, malgré ce que dit le cours. Et pour ça, on va reprendre la règle `R1` de l'examen.

```
[R1] e :- a, d, not (b, c).
```

On va tenter d'imaginer, dans le cas du fils gauche qui correspond à l'application, une construction valable pour l'évaluation.



Tout d'abord, on ne peut pas supprimer purement et simplement d: ça voudrait dire que la règle `e:- a, d` est équivalente à `e :- a`. On ne va pas non plus (comme beaucoup ont fait) le mettre dans le IN: ça voudrait dire que la règle `e :- a, d` est équivalente à `e, d :- a`. Le mettre dans le OUT n'a vraiment aucun sens. Mais le mettre dans le MBT peut sembler intéressant. C'est ce que j'ai représenté dans le dessin ci-dessus. En effet, ça pourrait se comprendre comme "il me manque le d pour l'instant pour évaluer, mais j'évalue quand même et il me faudra prouver d plus tard". OK, admettons. Mais maintenant, considérons la règle suivante:

```
[R2] d :- e.
```

L'application de cette règle sur la feuille 1 produit un sommet dont le IN est $\{a, e, d\}$, dont le OUT est $\{b \wedge c\}$, et dont le MBT est $\{d\}$. Comme nous le préciserons un peu plus loin, IN serait alors bien un modèle stable. Mais pourtant, je vous mets au défi de trouver la moindre dérivation (et a fortiori la moindre dérivation persistante et complète) qui produirait cet ensemble. Donc $\{a, e, d\}$ n'est pas un modèle stable et ce mécanisme d'évaluation, aussi satisfaisant soit-il à première vue, ne fonctionne pas.

C'est peut-être une mauvaise nouvelle pour ceux d'entre vous qui auraient proposé cette construction, mais c'est une bonne nouvelle pour moi qui n'ai arnaqué personne et n'aurai pas à me replonger dans vos copies.

Rappel: analyse d'un arbre APERIX Nous avons vu précédemment comment construire un arbre ASPERIX. Nous allons maintenant voir comment l'analyser pour en extraire les modèles stables. Il est important de noter que, par souci de simplicité, toutes les définitions données ici sont dans le cas d'un arbre *fini*. Tout serait beaucoup plus long à mettre en place dans le cas infini, et cette correction a déjà pris beaucoup trop d'ampleur.

Complétude d'une branche Une branche est dite *complète* lorsque toutes les évaluations possibles sur la feuille de cette branche ont été évaluées le long de la branche. Attention, dans le cas propositionnel, cette condition est équivalente à "toutes les règles évaluables ont été évaluées", mais en premier ordre, ceci veut dire "si R est évaluable pour le déclencheur σ sur la feuille, alors R a déjà été évaluée suivant σ le long de la branche."

Violation d'une interdiction Soit s un sommet quelconque d'un arbre, et (c, σ) un élément de son champ OUT (je rappelle que dans le cas général, la contrainte (c, σ) est donnée par une conjonction d'atomes c et un homomorphisme partiel σ). Alors s *viole* (c, σ) si il existe un homomorphisme de c dans le champ IN de s qui étend σ.

Satisfaction d'une obligation Soit s une *feuille d'une branche complète* et $d = (c_1, \sigma_1) \vee \cdots \vee (c_k, \sigma_k)$ un élément de son champ MBT (toujours dans le cas général). Alors s *satisfait* d si il existe une contrainte (c_i, σ_i) dans d et un homomorphisme de c_i dans le champ IN de s qui étend σ_i.

Simplification des tests de contraintes La violation comme la satisfaction reposent sur un test "il existe un homomorphisme de c dans le champ IN qui étend σ". Suivant la nature des règles que l'on considère, ce test peut s'exprimer de façon plus simple:

- dans le cas des règles skolémisées ou des règles datalog, une contrainte de OUT est réduite à la conjonction d'atomes c et un élément de MBT est une disjonction de conjonctions d'atomes. Le test précédent peut s'exprimer par "il existe un homomorphisme de c dans le champ IN" (il n'y a plus besoin de σ).
- dans le cas des règles propositionnelle s, ce test peut s'exprimer encore plus simplement puisqu'il suffit de tester "$c \subseteq$ IN".

Modèle stable Le théorème vu en cours dit que "un ensemble d'atomes E est un modèle stable du programme si et seulement si il est le champ IN de la feuille d'une branche complète qui ne viole aucun élément de son OUT et qui satisfait tous les éléments de son MBT".

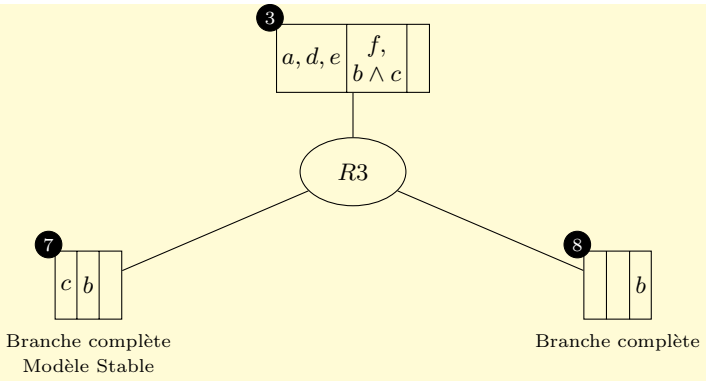
Optimisation Supposons qu'un sommet s viole un élément de son OUT. Voir que, pour tout descendant de s, il y aura encore violation de cet élément (qui sera encore dans le OUT). Donc aucun descendant de s ne sera un modèle stable. On peut donc couper l'exploration de l'arbre ASPERIX sur le sommet s.

Arbre ASPERIX, partie 2 La branche qui va de la racine au sommet 6 est une branche complète. En effet, son champ IN ne contient que a, les règles `R2` et `R4` ont déjà été évaluées, et plus aucune règle n'est déclenchable.

La feuille 6 ne satisfait cependant pas e, donc elle ne correspond pas à un modèle stable.

Un étudiant qui se serait arrêté ici aurait déjà pratiquement tous les points sur cet exercice. Je continue juste au cas où vous vous poseriez encore des questions et que, puisqu'on en est déjà à 18 pages, autant se lacher. Et puis, si je ne réponds pas complètement à la question qu j'ai moi-même eu le tort de poser, qui le fera ?

Arbre ASPERIX, partie 3 On continue le développement de l'arbre à partir du sommet 3 de l'*arbre ASPERIX, partie 1.*



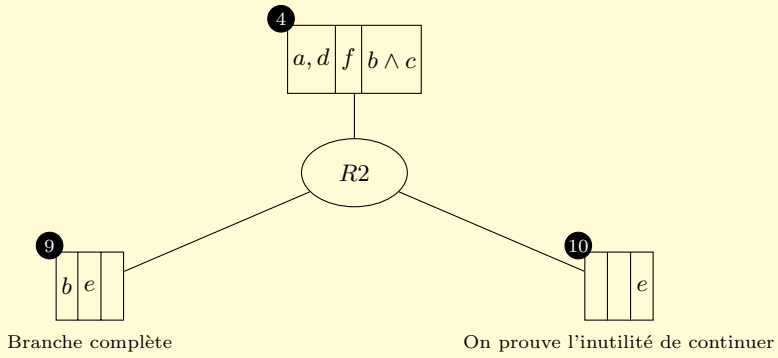
A partir du sommet 3 de l'arbre (partie 1), les règles `R2` et `R5` sont déjà bloquées (et le seront encore, donc jamais évaluables, dans tous les descendants de 3). Par contre, la règle `R3` est évaluable et on l'évalue comme ci-dessus. Les deux sommets obtenus sont les feuilles de branches complètes.

Le sommet 7 a un MBT vide, donc tous ses élements sont satisfaits, et on vérifie que 7 ne viole aucun élément du OUT. Le champ IN de 7 contient donc un modèle stable: c'est le E_1 de la question 3.1.

Le sommet 8 ne satisfait pas l'élément b de son MBT, ce n'est donc pas un modèle stable.

Avec ça en plus, c'était parfait, vous aviez tous les points et mes félicitations en prime. Mais moi je dois continuer. Et si, vous, vous n'êtes pas encore certain de la technique, continuez également, en essayant de le faire sur une feuille de papier.

Arbre ASPERIX, partie 4 On continue le développement de l'arbre à partir du sommet 4 de l'*arbre ASPERIX, partie 1.*



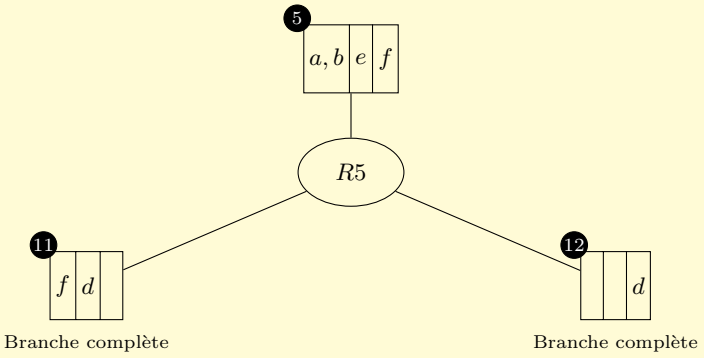
A partir du sommet 4 de l'arbre (partie 1), la règle `R2` est évaluable et on l'évalue comme ci-dessus.

Le sommet 9 est la feuille d'une branche complète: ni `R3` (bloquée par b), ni `R5` (bloquée par d) ne sont évaluables. Le sommet 9 ne satisfait cependant pas l'élément $b \wedge c$ de son MBT, il ne correspond pas à un modèle stable.

Le sommet 10 ne correspond pas à une branche complète, car la règle `R3` est encore évaluable. Par contre, la seule règle pouvant produire e, `R1`, a déjà été évaluée. Aucun successeur de 10 ne pourra donc contenir e dans son IN. Et donc aucun successeur de 10 ne pourra satisfaire l'élément e du MBT, et ainsi 10 ne peut pas mener à un modèle stable.

Le petit argument sur le sommet 10 ci-dessus vous montre comment, parfois, on peut couper plus tôt l'arbre ASPERIX *à condition de bien le justifier*. Dans le cas présent, faire l'économie de la dernière évaluation possible, `R3`, ne valait peut-être pas le coup de prendre le risque de se lancer dans une mini-démonstration.

Arbre ASPERIX, partie 5 On continue le développement de l'arbre à partir du sommet 5 de l'*arbre ASPERIX, partie 1.*



A partir du sommet 5 de l'arbre (partie 1), la règle `R5` est évaluable et on l'évalue comme ci-dessus.

Les deux sommets 11 et 12 sont les feuilles de branches complètes: ni `R1` (il manque d), ni `R3` (il manque également d) ne sont évaluables.

Voir que le sommet 11 ne viole aucun élément de son OUT (ni e ni d ne sont dans le IN), et qu'il satisfait tous les éléments de son MBT (f est dans le IN). Il correspond à un modèle stable, c'est même le modèle stable E_2 de la question 3.2.

Le sommet 12 ne satisfait pas l'élément d du MBT, ce n'est donc pas un modèle stable.

Voilà, c'est fini. On a bien retrouvé les deux modèles stables de la question 3, et on a prouvé (si on ne s'est pas trompé quelque part) qu'il n'y en a pas d'autres. Mais pour avoir confirmation, on peut demander à Clingo. Il faut modifier un peu le programme, car il n'accepte pas deux atomes dans une négation.

```
a.
e :- a, d, not bc.
bc :- b, c.  % simulation de b,c par l'unique atome bc
b :- a, not e.
c :- d, not b.
d :- a, not f.
f :- b, not d.
```

Et on obtient:

```
Solver: clingo version 5.8.0
Models: 2 (no)
Calls: 1
Time: Total: 0s, Solve: 0s, Model: 0s
Result: SATISFIABLE

Answer 1/2
a, f, b
```

Answer 2/2
a, d, c, e

Et pour finir **bonnes vacances** et surtout, **bonnes révisions**.