Université de Montpellier – Master Informatique
Intelligence Artificielle – HMIN107

Systèmes à base de règles

(logique des propositions)

Dans les systèmes à base de règles, une base de connaissances est composée d'une **base de faits**, qui est un ensemble de faits, représentant des observations factuelles, et d'une **base de règles**, qui est un ensemble de règles, représentant des connaissances générales sur un domaine d'application. Une **règle** est une connaissance de la forme « **si** <hypothèse> **alors** <conclusion> » et a la sémantique intuitive suivante : « si la partie hypothèse est vérifiée par l'ensemble des faits, alors la partie conclusion peut être considérée comme un (nouveau) fait ».

Vocabulaire : on dit aussi « si <prémisses> alors <conclusion> », ou « si <condition> alors <conclusion> »; ou encore, en programmation logique (Prolog notamment), on parle de corps pour l'hypothèse et de tête pour la conclusion (car on voit les règles sous la forme « <tête> si <corps> »).

Exemples de règles :

`Eau ∧ TempératureSup90 → EauBout`
(règle en logique des propositions, ou logique d'ordre 0)

`∀x (Température(x) > 90 ∧ x = Eau → Etat(x,Ebullition))`
(règle en logique du premier ordre)

`Température > 90 ∧ Liquide = Eau → Etat = Ebullition`
(règle dite d'ordre 0+ : les symboles propositionnels peuvent avoir des valeurs non booléennes, et on dispose des comparateurs booléens =, >, …, mais il n'y a ni variables, ni quantificateurs).

Un **fait** peut être vu comme une règle à hypothèse vide (interprétée comme toujours vraie). La forme des faits varie donc selon la forme des règles.

Historiquement, les premiers systèmes à base de règles ont été les **systèmes experts**, systèmes qui visent à modéliser l'expertise humaine. Ces systèmes ont connu un certain succès dans les années 70-80 (voir par exemple MYCIN pour le diagnostic de maladies du sang, ou le système industriel R1 de DEC pour la configuration d'ordinateurs). Toutefois, ces systèmes reposaient sur une combinaison de techniques plus ou moins ad hoc, directement liées au cas applicatif. En particulier, les algorithmes n'étaient pas indépendants d'une base de règles particulière. La **maintenance** de ces systèmes (pour prendre en compte une modification de la base de règles) était donc difficile et on ne pouvait pas **transférer** un système expert à un autre cas applicatif. D'autre part, on ne pouvait **valider** un système expert que de façon **empirique** : en testant le système sur des exemples particuliers pour lesquels on connaissait le résultat voulu, on pouvait dire que le système donnait un résultat correct dans *x*% des cas, mais on ne pouvait pas **expliquer** quel raisonnement avait été effectué pour obtenir tel résultat.

Actuellement, on considère des **systèmes à bases de connaissances**, qui distinguent clairement deux composantes : la **base de connaissances** elle-même, qui est propre à une application, et les **services de raisonnement** qui sont **génériques**, au sens où ils pourraient s'appliquer à n'importe quelle base de connaissances. D'autre part, ils ont des bases mathématiques solides : logiques principalement, mais aussi théorie des probabilités par exemple pour prendre en compte des incertitudes sur les connaissances. L'un des types de connaissances les plus répandus dans les

systèmes à base de connaissances, sont les **règles**. La base de connaissances est alors composée de faits et de règles (parfois les faits sont vus comme des règles à hypothèse vide). Les services de raisonnement, qui peuvent s'appliquer à n'importe quelle base de connaissances composée de faits et de règles, reposent sur deux techniques fondamentales : le **chaînage avant** et le **chaînage arrière**.

Les systèmes à base de règles se sont largement développés dans le cadre de la **programmation logique** (Prolog typiquement) et des **bases de données déductives** (Datalog typiquement) et on les trouve aujourd'hui dans divers domaines.

Dans ces notes de cours, nous étudierons le traitement de règles simples en **logique des propositions** (ou logique d'ordre 0).

1. Les règles en logique des propositions : définitions de base

Un **atome** est un symbole propositionnel. Un **littéral** est un atome (littéral positif) ou la négation d'un atome (littéral négatif). Une **clause** est une disjonction de littéraux (ex : ¬A∨B∨¬C∨D).

Une **règle conjonctive** a pour hypothèse une conjonction de littéraux et sa conclusion est réduite à un littéral.

Ex : A ∧ ¬B ∧ C → D

Un **fait** pouvant être vu comme une règle à hypothèse vide, c'est donc un littéral das le contexte des règles conjonctives.

Remarque : Toute règle conjonctive s'écrit de façon équivalente sous la forme d'une clause. Toute clause s'écrit de façon équivalente sous la forme d'une règle conjonctive (on a plusieurs règles équivalentes possibles).

Une **règle (conjonctive) positive** ne contient pas de négation : tous les littéraux sont positifs.

Ex : A ∧ B ∧ C → D

Les règles (conjonctives) positives correspondent exactement aux **clauses définies** : clauses ayant *exactement un* littéral positif.

Ex : ¬A ∨ ¬B ∨ ¬C ∨ D (correspondant à la règle précédente).

Les systèmes à base de règles considèrent le plus souvent des **règles positives** et des **faits**, c'est-à-dire des clauses définies. Ils peuvent aussi s'étendre aux **clauses de Horn**, qui sont des clauses ayant *au plus* un littéral positif . Ceci permet de représenter non seulement des règles positives et des faits, mais aussi des règles qui expriment des contraintes (clauses sans aucun littéral positif).

Ex (contrainte) : ¬A ∨ ¬B ∨ ¬C, ou de façon équivalente ¬(A ∧ B ∧ C), ou encore A ∧ B ∧ C → ⊥ (où ⊥ se lit « absurde » ou symbole toujours faux).

Ex (contrainte) : IlFaitJour ∧ IlFaitNuit → ⊥

Les contraintes sont utilisées pour vérifier que la base de faits est cohérente, soit initialement, soit après avoir été enrichie par un certain nombre d'applications de règles.

Dans la suite, on considère qu'une **base de connaissances** est composée d'une base de **faits** et d'une base de **règles**.

2. Chaînage avant

Deux mécanismes de raisonnement sont spécifiquement associés aux règles :

- le **chaînage avant** (forward chaining), qui permet de produire de nouveaux faits en appliquant les règles. On le dit dirigé par les données. Par exemple une règle $A \wedge B \wedge C \rightarrow D$ peut être appliquée si les atomes A, B et C appartiennent à la base de faits ; l'atome D est alors produit et ajouté à la base de faits.
- le **chaînage arrière** (backward chaining), qui consiste à prouver un atome appelé but en « remontant » le long des règles. On le dit dirigé par un but. Par exemple, si l'on cherche à prouver l'atome D et que l'on a la règle $A \wedge B \wedge C \rightarrow D$, on peut essayer de prouver $A \wedge B \wedge C$ (si on y arrive, on a une preuve de D).

Ces mécanismes sont **adéquats** (ou **corrects**) et **complets** pour les faits et règles positives (on pourrait même dire pour les clauses de Horn, car les contraintes ne peuvent de toutes façons pas intervenir pour déclencher l'application de règles, puisque leur conclusion n'apparaît dans aucune règle).

Adéquation (ou « correction ») : si l'atome A est produit par le mécanisme de chaînage avant (ou : prouvé par le mécanisme de chaînage arrière), alors A est effectivement conséquence logique de la base de connaissances.

Complétude : si l'atome A est conséquence logique de la base alors le mécanisme de chaînage avant le produit effectivement (ou : le mécanisme de chaînage arrière le prouve effectivement).

Autrement dit, un mécanisme est *adéquat* s'il ne fait que des inférences correctes, et il est *complet* s'il fait au moins toutes les inférences correctes. L'adéquation permet de croire le système lorsqu'il répond "oui" à la question "a-t-on A ?", et la complétude permet de le croire lorsqu'il répond "non" (ou "je ne sais pas", voir la distinction monde ouvert / monde clos) à cette question.

Les chaînages avant et arrière ne sont plus complets si les faits ou les règles comportent des **négations**. Et ils nécessiteraient d'être étendus pour prendre en compte des règles avec **disjonction** en conclusion.

Nous considérons maintenant des bases de connaissances de la forme **K = (BF, BR)**, où BF est la base de faits et BR est la base de règles. Les faits et règles sont positifs.

Principe du chaînage avant :

- une règle est **applicable** si sa partie hypothèse est incluse dans la base de faits
- **appliquer** une règle consiste à ajouter sa conclusion à la base de faits (si cette conclusion n'appartient pas déjà à la base de faits : dans ce cas, on dit que l'application de la règle est **utile**)
- le chaînage avant consiste à appliquer des règles, tant que c'est possible, ou tant qu'un certain atome (que l'on cherche à produire) n'a pas été obtenu.
- la base de faits est dite **saturée** si toutes les règles applicables ont été appliquées (plus aucun fait nouveau ne peut être obtenu par application de règle).

Remarque : une règle s'applique au plus une fois (autrement dit, il n'y a pas deux façons différentes d'appliquer une règle, ce qui ne serait plus le cas en logique d'ordre 1)

Pour saturer la base de faits avec un ensemble de règles, on peut procéder ainsi : à chaque tour, on détermine quelles règles sont applicables de façon utile (c'est-à-dire que ces règles n'ont pas déjà été appliquées et que leur conclusion n'apparaît pas déjà dans la base de faits) et on applique ces règles; on continue jusqu'à ce que plus aucun nouveau fait ne puisse être produit.

Cet algorithme a un temps d'exécution **polynomial** en la taille K (la taille de K étant « nombre de faits + somme des tailles (= nombre de symboles) des règles ») : en effet, on effectue un nombre de tours borné par "1+min(nombre de symboles dans K, nombre de règles)" [à chaque tour, sauf le dernier, on effectue au moins une application de règle et on produit au moins un fait]. Et à chaque tour, on parcourt toutes les règles, on vérifie si une règle est applicable de façon utile en un temps polynomial en "taille de la règle x taille max de la base de faits" (la base de faits grossit, mais on peut grossièrement borner sa taille par le nombre de symboles dans K). Très très grossièrement, on peut dire que l'algorithme « naïf » a une complexité en O(taille(K)^3).

L'algorithme ci-dessous procède autrement : au lieu de parcourir les règles, il maintient une liste de faits à traiter. A chaque règle est associé un compteur, dont la valeur correspond au nombre d'atomes de son hypothèse qui n'ont pas encore été reconnus comme faits (le compteur est donc initialement égal à la taille de l'hypothèse de la règle). Initialement, tous les atomes de la base de faits sont à traiter. Traiter un atome revient à décrémenter le compteur des règles qui le contiennent en hypothèse ; une règle devient applicable lorsque son compteur passe à zéro ; l'appliquer effectivement consiste à ajouter sa conclusion à la base de faits et à la liste des atomes à traiter.

Cet algorithme est implémentable en un temps **linéaire** en la taille de K.

```
Algorithme FC(K) // saturation de la base K
// Données : K = (BF, BR)
// Résultat : BF saturée par application des règles de BR
Début
  ATraiter ← BF
  Pour toute règle R de BR
    Compteur(R)← Nombre de symboles de l'hypothèse de R
  Tant que ATraiter n'est pas vide
    Retirer un atome A de ATraiter
    Pour toute règle de BR ayant A dans son hypothèse
      Décrémenter Compteur(R)
      Si Compteur(R) = 0 // R est applicable
           Soit C la conclusion de R
           Si C ∉ (BF ∪ Atraiter) // l'application de R est utile
                 Ajouter C à ATraiter
                 Ajouter C à BF
    FinPour
  FinTantQue
Fin
```

[Réfléchir aux structures de données permettant d'implémenter cet algorithme en temps linéaire en la taille de K, voir TD*]*

Le chaînage avant effectue un raisonnement de type « Modus Ponens » :
de H et (H → C), on conclut C.
Dans notre cas, H représente une partie de BF et (H → C) est une règle de BR.
Il est facile de vérifier que le Modus Ponens est adéquat : C est effectivement conséquence logique de (H ∧ H → C).

Propriété : le mécanisme de chaînage avant est **adéquat (correct)** et **complet** (par rapport à la conséquence en logique des propositions).

Preuve :
K peut-être vue comme une seule formule : on fait la conjonction de tous les faits et de toutes les règles. Le mécanisme de chaînage avant produit une base de fait saturée que l'on note ici BF*. On va montrer que pour tout atome A, $\mathbf{K} \models \mathbf{A}$ (A est conséquence de K) si et seulement si $\mathbf{A} \in \mathbf{BF^*}$.
Rappel : « A est conséquence de K » signifie que tout modèle de K est un modèle de A.

Adéquation : on montre que « pour tout atome A, si A est dans BF* alors A est conséquence de K ». Montrons-le par récurrence sur le nombre *i* d'applications de règles ayant conduit à ajouter A à BF*. On note BF^i la base de faits obtenue après i applications de règles. BF^0 = BF. On prouve la propriété P suivante pour tout i ≥ 0 : « si A est obtenu en *i* applications de règles, alors A est conséquence de K ». Si *i* = 0, P est vraie car A appartient à BF, donc à BF*. Supposons que P soit vraie pour $i \geq 0$ et montrons qu'elle est alors vraie pour (i+1). Soit A produit en (i+1) applications de règles et soit R : H → A la règle appliquée pour produire A. On sait que A est conséquence de (H ∧ H→A). Il reste à vérifier que (H ∧ H→A) est conséquence de K. Puisque R est applicable, tout atome de H est dans BF^i, donc a été produit en au plus *i* applications de règles. Par hypothèse de récurrence, chacun de ces atomes est conséquence de K, donc H est conséquence de K. Quant à la règle H→ A, elle appartient à K donc est conséquence de K. Par transitivité de la conséquence logique, A est conséquence de K. La propriété P est donc vraie pour (i+1). On a montré qu'elle était vraie pour tout i ≥ 0.

Complétude : on montre que « pour tout atome A, si A est conséquence de K, alors A est dans BF*».
Supposons que A soit conséquence de K, autrement dit que « tout modèle de K est un modèle de A ». On va montrer que BF* peut être vue comme un modèle de K : ensuite, comme on a supposé que tout modèle de K était un modèle de A, on en concluera que BF* contient A.

En effet, considérons BF* : à partir de BF*, on construit un tableau de booléens T indicé par les atomes apparaissant dans les faits et les règles, et tel que pour tout atome B, T[B] = vrai si et seulement si B appartient à BF* . Ce tableau peut être vu comme une interprétation (au sens logique) de l'ensemble des symboles propositionnels apparaissant dans K. Appelons cette interprétation I : pour un symbole p, I(p) = vrai si et seulement si T[p]=vrai. I est forcément un modèle de K : sinon, cela signifierait qu'un fait de BF n'est pas dans BF* (c'est impossible puisque l'algorithme ne supprime aucun fait), ou que I rend une règle (H → C) fausse, c'est-à-dire I rend H vraie et C fausse (c'est impossible car cette règle serait applicable, donc sa conclusion est forcément dans BF*). Par hypothèse, tout modèle de K est un modèle de A, donc I est aussi un modèle de A. Donc T[A] = vrai, puisque T et I donnent les mêmes valeurs aux symboles propositionnels. A est donc bien dans BF*.

On peut montrer plus précisément que BF* correspond au **plus petit modèle de K** ("plus petit" au sens du nombre d'atomes ayant la valeur vraie) : le chaînage avant "met à vrai" les symboles qui doivent **forcément** être vrais dans tout modèle de BF et BR.

Propriété :
Soit le problème suivant :
Données : K = (BF,BR) , une base de connaissance formée de faits et règles positives d'ordre 0, et Q un littéral positif
Question : Q est-il conséquence de K ?
Ce problème est **polynomial** en temps (et peut même être résolu par un algorithme **linéaire** en la taille de K comme on l'a vu plus haut).

Preuve : Comme le chaînage avant est adéquat et complet par rapport à la conséquence logique, la base de faits obtenue par FC(K) contient exactement tous les faits conséquences de K. On peut donc

exécuter FC(K) puis tester si Q apparaît dans la base de fait saturée [ou bien : modifier FC(K) pour s'arrêter quand on a produit Q].

Remarque : si K était une formule quelconque et Q un littéral (qu'on peut même prendre positif), le problème « Q est-il conséquence de K ? » serait **co-NP-complet**, ce qui revient à dire que le co-problème « est-ce que Q *n'est pas* conséquence de K ? » est NP-complet.

3. Chaînage arrière

Etant donnée une question (ou requête) Q, on se demande « quelles règles permettraient de prouver Q ». Q est le « but » initial. Q est prouvé lorsque le but courant est vide (on dit « effacé »).
Voici un algorithme brutal implémentant ce mécanisme (de façon récursive) :

```
Algorithme BC(K,Q) // chaînage arrière
// Données : K = (BF,BR) et Q une liste d'atomes
// Remarque : on voit un fait A comme une règle → A (à hypothèse vide)
// Résultat, si l'algorithme s'arrête : vrai ssi Q peut être produit par
application des règles de BR sur BF (mais l'algorithme peut ne pas s'arrêter)
Début
Si Q = vide
  retourner vrai
Pour toute règle R = H1 ∧ … ∧ Hn → C de BR et BF
  telle que C = premier(Q)   // premier(Q) = premier atome de la liste Q
     Q' <- Concaténer [H1 … Hn] et reste(Q)
     // reste(Q) = Q privé de son 1er atome
      Si BC(K,Q')
            retourner vrai
FinPour
Retourner faux
Fin
```

Cet algorithme implémente le chaînage arrière avec une stratégie « en profondeur d'abord ». Notez qu'il peut « boucler » indéfiniment s'il ajoute à Q' un atome Hi qu'il cherchait justement à prouver (par exemple : pour prouver Q, on peut remonter la règle « si A et B alors Q », puis pour prouver A, remonter la règle « si Q alors A », …) .

On va adopter une autre stratégie qui consiste à remonter le graphe ET-OU de la base de règles (voir diapos de cours) et à générer un appel récursif sur chaque atome à prouver (voir algorithme BC2 des diapos de cours). De plus on va gérer récursivement une liste (L) des atomes à ne pas générer, sous peine de boucler : cette liste correspond aux atomes qui sont sur le chemin menant de la racine de l'arbre à l'atome courant, c'est-à-dire ceux que l'on est en train de chercher à prouver. D'où l'algorithme ci-dessous :

```
Algorithme BC3(K,Q,L)   // appel initial : BC3(K,Q,{})
// Données : K = (BF,BR), Q atome, L ensemble d'atomes à ne pas générer
// Résultat : vrai ssi Q est prouvable
Début
Si Q ∈ BF
  retourner vrai
Pour toute règle R = H1 ∧ … ∧ Hn → Q de BR
  Si aucun des H1 ... Hn n'appartient à L // sinon on va boucler
       i ← 1
       Tant Que i ≤ n  et BC3(K, Hi, L U {Q})
          incrémenter i
        Si i > n, retourner vrai   // hypothèse(R) prouvée, donc Q aussi
FinPour
Retourner faux
Fin
```

4. **Règles conjonctives avec négation du monde clos**

4.1. Monde ouvert/Monde clos

C'est une distinction fondamentale en représentation des connaissances :

- Hypothèse du **monde ouvert** : on considère que l'on ne sait pas tout. La base de connaissances encode un ensemble de mondes possibles : tous ses modèles. Une proposition P est inférée de K si elle est conséquence logique de K ($K \models P$: tout modèle de K satisfait P), sinon sa valeur de vérité est simplement inconnue. C'est l'hypothèse faite en logique classique.
- Hypothèse du **monde clos** : on considère que l'on a une connaissance complète du monde. La base de connaissances encode exactement ce qui est vrai, tout le reste est faux. Il existe un seul modèle de K. Une proposition est inférée de K si elle est vraie dans ce modèle. C'est l'hypothèse faite en bases de données.

Exemple : BF = {A,B,C}. Que nous dit BF sur D ? En monde ouvert, rien : D peut être vrai ou faux. En monde clos, D est absent de la BF, donc il est faux (« s'il était vrai on l'aurait su et on l'aurait mis dans la base de faits »).

- **Négation en monde ouvert (notation $\neg$)** : **$\neg$A** découle de K si $K \models \neg A$. En particulier à partir d'une base de faits positives, et de règles positives, on n'infère jamais rien de négatif (en effet, prenons l'interprétation qui rend tous les symboles vrais : c'est un modèle de K, donc pour tout littéral négatif $\neg$A, il existe au moins un modèle de K qui ne satisfait pas $\neg$A).
- **Négation en monde clos (notation not)** : **not A** découle de K si le plus petit modèle de K (plus petit = en nombre de symboles prenant la valeur vrai) n'est pas un modèle de A. Ceci revient à dire que $K \not\models A$ (A n'est pas conséquence logique de K). Cette notion est proche de la notion de négation par l'échec (cf. Prolog).

4.2. Règles conjonctives avec négation du monde clos

Nous savons que le chaînage avant sur les règles conjonctives (avec négation) n'est pas complet, si l'on considère la négation au sens de la logique classique (monde ouvert). Nous allons maintenant faire l'hypothèse du monde clos.

- un **fait** est un symbole (littéral positif)
- une **règle** est de la forme $H \rightarrow C$ où H est une conjonction de littéraux et C est un littéral positif.

On notera aussi une règle par $H+, H- \rightarrow C$, où :
H+ est l'ensemble des symboles des littéraux positifs
H- est l'ensemble des symboles de littéraux négatifs

Exemple : $A \wedge B \wedge \text{not } C \wedge \text{not } D \rightarrow E$ (on peut aussi remplacer le $\wedge$ par une virgule)
avec $H+ = \{A,B\}$ et $H- = \{C,D\}$

La négation ne peut intervenir que dans l'hypothèse des règles (sinon, on produirait des faits "négatifs", ce qui n'a pas de sens en monde clos).

Comment adapter le chaînage avant ? Une règle $H+, H- \rightarrow C$ est dite **bloquée** si $H- \cap BF \neq \emptyset$. Elle est **applicable** si *(1)* elle n'est pas bloquée et *(2)* $H+ \rightarrow C$ est applicable (autrement dit $H+ \subseteq BF$). On considère qu'un littéral découle de K si et seulement si il appartient à la base de faits saturée. La réponse à une requête sous forme de littéral est donc *vrai* si ce littéral appartient à la base de faits saturée et *faux* sinon.

Problème : Selon l'**ordre d'application des règles**, on n'obtient pas la même base de faits saturée

Exemple 1 : BF = {A}
R1 : A, not B → C
R2 : A, not C → B

Selon l'ordre d'application entre R1 et R2, on obtient BF* = {A,C} ou BF* = {A,B}

La logique classique ne pose pas ce problème car elle est **monotone** : si une formule f est conséquence logique d'un ensemble de formules E, alors f reste conséquence si l'on ajoute à E de nouvelles formules. La négation du monde clos rend la conséquence **non-monotone** (cf. exemple 1 : de {A, R1} on infère C, mais de {A, R1, B}, on n'infère plus C).

Avec les règles positives, nous avions la propriété qu'une base de connaissances avait un unique plus petit modèle, qui correspondait à la base de faits saturée. Nous avons perdu cette propriété. Dans l'exemple 1, on pourrait concevoir qu'on a deux mondes possibles. Certains langages de programmation logique partent sur cette idée (voir « Answer set programming »). Mais, même si l'on admettait plusieurs mondes possibles, certains mondes seraient refusés car ils font une utilisation incorrecte de la négation :

Exemple 2 : BF = {A}
R1 : A, not B → C
R2 : A → B

Si on applique R1 avant R2, on obtient {A,C,B}. Mais ceci est problématique car on s'est servi de l'absence de B pour produire C, alors qu'on a finalement B présent. Le résultat voulu est BF* = {A,B}.

L'idée est la suivante : si on utilise « not B » à un moment donné pour inférer quelque chose, il faut être sûr qu'on ne va pas produire B par la suite.

Nous allons maintenant imposer des contraintes syntaxiques sur les ensembles de règles qui assurent que :

1. la négation est utilisée correctement : l'application d'une règle dont l'hypothèse contient not B intervient toujours après les applications des règles qui produisent B.
2. le résultat de la saturation est unique.

- **Ensemble de règles semi-positif** : seuls les symboles qui n'apparaissent *pas* en conclusion de règles peuvent être niés. L'ordre des règles n'a alors aucune importance.

- **Ensemble de règles stratifié :** l'idée est de mettre les règles dans un ordre qui assure que lorsqu'on utilise un littéral négatif dans une hypothèse de règle, on ne peut plus produire ce littéral en positif. A chaque symbole p on associe un entier $\alpha(p)$. Les règles sont ensuite rangées en strates suivant le numéro α associé à leur conclusion. On exécute les règles en marche avant par ordre croissant des strates : à l'étape i, on sature la base de faits calculée à l'étape i-1 avec la règles de la strate i.

  Pour chaque symbole p, la numérotation α vérifie :
  1. Pour toute règle de la forme … p… → q, on $\alpha(p) \leq \alpha(q)$ (le $\leq$ permet la récursivité entre règles)
  2. Pour toute règle de la forme …not p… → q, on a $\alpha(p) < \alpha(q)$

Un ensemble de règles stratifié est ainsi découpé en une séquence de **sous-ensembles** de règles **semi-positifs.**

Un ensemble de règles est **stratifiable** si on peut le transformer en un ensemble de règles stratifié.

Propriété : si un ensemble de règles est **stratifiable**, alors **toutes** ses stratifications produisent la **même** base de faits saturée.

On peut ainsi associer un **unique modèle** à un ensemble de règles stratifiables.

Un outil utile : le **graphe de précédence** des symboles.

Les sommets de ce graphe sont les symboles qui apparaissent en conclusion de règles (on peut aussi mettre les autres symboles, mais ils ne joueront aucun rôle dans les décisions), et on a un arc de p à q si p apparaît dans l'hypothèse d'une règle de conclusion q ; cet arc est étiqueté + (respectivement -) si p apparaît dans un littéral positif (respectivement négatif).

Un ensemble de règles est stratifiable si et seulement si son graphe de précédence n'admet **aucun circuit avec un arc négatif**.

Une stratification s'obtient en associant à chaque composante fortement connexe (c.f.c.) une strate de façon compatible avec l'ordre partiel sur les c.f.c. : étant données deux c.f.c. Ci et Cj, s'il y a un arc négatif (respectivement positif) d'une règle de Ci vers une règle de Cj, alors strate(Ci) < strate(Cj) (respectivement strate(Ci) $\leq$ strate(Cj)).

Tous les ensembles de règles ne sont pas stratifiables, voir l'exemple 1.