Systèmes à base de règles (II)

Nous nous intéressons maintenant aux règles positives en logique du premier ordre (sans fonctions), *et aux mécanismes de traitement de ces règles (chaînages avant et arrière).*

2. Les règles en logique du premier ordre

2.1 Définitions de base

On généralise les définitions de la section précédente de la façon suivante :

Un **terme** est une **variable** ou une **constante** (on ne considère pas de fonctions). Un **atome** est de la forme $p(e_1 \ldots e_k)$, où p est un **prédicat** et les e_i sont des termes. Un **littéral** est un atome ou la négation d'un atome. Une formule est **complètement instanciée** si elle ne comporte que des constantes.

Une **clause** est une disjonction de littéraux, où toutes les variables sont quantifiées **universellement**. Une **clause définie** a exactement un littéral positif (et une **clause de Horn** au plus un littéral positif).

Une **règle (conjonctive)** est de la forme $\forall x1 \ldots x_n$ (H → C), où :
- H est une conjonction de littéraux
- C est composée d'un seul littéral
- $x_1 \ldots x_n$ sont les variables de H
- **toutes** les variables de C apparaissent dans H.

Remarque : puisque toutes les variables de la conclusion apparaissent dans l'hypothèse, le seul cas où une règle peut avoir une hypothèse vide, c'est lorsque la conclusion ne comporte que des constantes.

Une **règle (conjonctive) positive** ne comporte que des littéraux positifs. Un **fait** est une règle à hypothèse vide, c'est donc un atome complètement instancié. Faits et règles sont des clauses définies. Voir que toutes les clauses définies ne sont pas forcément des règles ou faits : par exemple, $\forall x\ p(x)$ n'est pas une règle au sens défini plus haut.

[On pourrait aussi considérer un fait comme une conjonction d'atomes fermée existentiellement, par exemple $\exists x \exists y \exists z\ (p(x,y) \wedge p(y,z))$. Tous les résultats que nous obtiendrons avec des faits complètement instanciés sont encore valables avec ces faits plus complexes.]

2.2 Chaînage avant

Le chaînage avant ne consiste plus à appliquer un simple Modus Ponens mais un Modus Ponens généralisé, qui repose sur la **règle d'instanciation universelle** :

si on a R = $\forall x1 \ldots x_n$ (H → C) et qu'on remplace $x_1 \ldots x_n$ par des constantes, on obtient une formule complètement instanciée R' : H' → C' qui est conséquence de R.

Le **Modus Ponens généralisé** consiste à :
(1) instancier les variables universelles d'une règle, produisant R' = H' → C' (règle sans variables)
(2) puis à appliquer le Modus Ponens de base : de H' et H' → C', on déduit C'.

[Exemple du menu]

Rappels de définitions de logique : de façon générale, une **substitution** est une application d'un ensemble de *variables* dans un ensemble de *termes* (ici des variables ou des constantes). On peut la noter sous la forme d'un ensemble de couples $S = \{(x_1, t_1) \ldots (x_n, t_n)\}$, où $\{x_1 \ldots x_n\}$ est le domaine de la substitution. Tous les x_i sont des variables distinctes, mais tous les t_i ne sont pas forcément des termes distincts (on peut remplacer deux variables par le même terme).
Etant donnée une substitution S et un ensemble d'atomes H, *S(H)* désigne l'ensemble d'atomes obtenu en remplaçant dans H chaque variable x_i par t_i. Les constantes ne sont pas remplacées, et une variable autre que $x_1 \ldots x_n$ n'est pas remplacée non plus. Il est commode de voir l'hypothèse d'une règle, sa conclusion, ou la base de faits comme des ensembles d'atomes.

On va considérer des **substitutions** qui permettent de remplacer toutes les variables d'une règle par des constantes de la base de faits.

"Instanciation brutale" de la base de règles :

Une première façon de définir le chaînage avant consiste à éviter les difficultés de l'application de règles d'ordre 1 en se ramenant à des règles d'ordre 0. On remplace chaque règle R d'ordre 1 par l'ensemble des règles complétement instanciées que l'on peut obtenir en remplaçant les variables de R par des constantes de la base de connaissances (noter qu'il peut y avoir des constantes qui ne sont pas dans la base de faits initiale, mais sont apportées par des applications de règles : ceci peut arriver lorsque la conclusion d'une règle comporte une constante qui n'apparaît pas dans son hypothèse ; il faut prendre en compte ces constantes lorsqu'on « propositionnalise » la base de règles). Les règles obtenues sont dites d'ordre 0 car leur application se définit comme pour les règles de la logique des propositions : il suffit de tester des égalités d'atomes (l'atome p(a,b) par exemple, où a et b sont des constantes, peut être vu comme un seul symbole propositionnel). Autrement dit, on se ramène à un simple Modus Ponens.

Bien sûr, le gain au niveau de l'algorithme de chaînage avant se paie : le nombre de règles obtenu est *exponentiel* en la taille de la base de connaissances originelle. Une façon d'éviter cette explosion combinatoire de l'espace mémoire consiste à définir un mécanisme de chaînage avant qui fonctionne *directement* sur des règles du premier ordre en généralisant la notion d'application de règle.

Chaînage avant du premier ordre :

Une règle $\forall x_1 \ldots x_n$ H $\rightarrow$ C est **applicable** s'il existe une substitution S des variables $x_1 \ldots x_n$ de H dans les constantes de BF, telle que **S(H) est inclus dans BF**. Une telle substitution S s'appelle un **homomorphisme** de H dans BF.

Appliquer la règle selon S consiste à ajouter **S(C)** dans BF. Cette application est **utile** si S(C) n'apparaît pas déjà dans BF.
Saturer la base de faits consiste à appliquer les règles de toutes les façons possibles jusqu'à ce qu'aucun fait ne puisse plus être ajouté.

On peut montrer que le chaînage avant est **adéquat** et **complet** (les arguments sont similaires à ceux utilisés pour la preuve en logique des propositions, mais plus complexes : utilisation du modus ponens généralisé au lieu du simple modus ponens, et notion d'interprétation plus complexe en logique du premier ordre).

L'algorithme ci-dessous procède *en largeur* : à chaque étape, il cherche quelles sont toutes les règles applicables puis les applique toutes, avant de passer à une nouvelle étape. Il s'arrête lorsqu'à une certaine étape il n'est plus possible de produire de nouveaux faits.

```
Algorithme FC(K) // saturation de la base K
// Données : K = (BF, BR)
// Résultat : BF saturée par application des règles de BR
Début
Fin ← faux
Tant que non fin
  new ← ∅ // ensemble des nouveaux faits obtenus à cette étape
  // on peut voir new comme une autre base de faits, temporaire
  Pour toute règle R = H → C de BR
    Pour tout homomorphisme S de H dans BF
      Si S(C) n'est ni dans BF ni dans new
           Ajouter S(C) dans new
    FinPour
  FinPour
  Si new = ∅
    Fin ← vrai
  Sinon ajouter tous les éléments de new à BF
FinTantQue
Fin
```

Imaginons que l'on ait une requête Q *:*

- si celle-ci a la même forme qu'un fait, elle est conséquence de K si et seulement si elle appartient à la base saturée ;
- si elle est de la forme « trouver toutes les valeurs de $x_1 \ldots x_n$ tels que Q' », une réponse est une instanciation de Q' par des constantes, telle que cette instanciation est conséquence de K ; chaque *homomorphisme* S de Q' dans la base *saturée* définit alors une réponse.

Cet algorithme a un **temps d'exécution exponentiel** en la taille de la base (dans le pire des cas). Il n'est pas possible d'obtenir un algorithme polynomial (sauf si P = NP) car le problème qui consiste à déterminer si une règle est applicable (« étant donnés H et BF, existe-t-il un homomorphisme de H dans BF ? ») est déjà NP-complet.

Exercice (pour les amateurs de la théorie de la complexité) : Montrer que le problème du test de l'existence d'un homomorphisme d'un ensemble d'atomes dans un autre (par exemple d'une hypothèse de règle dans une BF) est NP-complet, en effectuant une réduction polynomiale à partir du problème 3-COLORATION : « étant donné un graphe non orienté G, est-il possible d'associer l'une des trois couleurs 1, 2 ou 3, à chaque sommet de G, de façon à ce que deux sommets adjacents aient des couleurs différentes ? ».

Exercice :
Donner une borne supérieure de la taille de BF saturée. Cette borne est-elle exponentielle en la taille de BF initiale ? Et si l'on suppose que l'arité des prédicats est bornée par une constante ?

Comment améliorer cet algorithme :

- **calculer efficacement** l'ensemble des homomorphismes de H dans BF ;
- restreindre **l'ensemble des homomorphismes à considérer** : à l'étape t, un homomorphisme S est susceptible de produire une nouvelle application d'une règle R= H → C s'il utilise au moins un fait ajouté à l'étape t-1, c'est-à-dire s'il envoie l'un des atomes de H sur un fait ajouté à l'étape t-1 ;
- tester efficacement si un atome est dans la base de faits courante ;
- ne pas considérer toutes les règles à chaque étape : pour ce faire, exploiter le « **graphe de dépendances des règles** ».

Principe de ce graphe : coder les dépendances entre règles.

Une règle R2 **dépend** d'une règle R1 s'il est possible qu'une application de R1 produise une nouvelle application de R2 : plus précisément, on dit que R2 dépend de R1 s'il existe une base de faits BF telle que :

- R1 est applicable sur BF
- Soit BF' obtenue après application de R1 sur BF : R2 est applicable sur BF' par un homomorphisme de son hypothèse dans BF', et cet homomorphisme n'est pas un homomorphisme dans BF (autrement dit, cet homomorphisme utilise l'atome ajouté par l'application de R1).

Le graphe de dépendances des règles est un graphe orienté ayant pour ensemble de sommets l'ensemble des règles et tel qu'il existe un arc (Ri,Rj) si et seulement si Rj dépend de Ri (« Ri peut déclencher Rj »). Remarquer qu'une règle peut dépendre d'elle-même : le graphe de dépendances peut donc contenir des boucles.

→ Comment calcule-t-on la relation de dépendance ? (penser aux cas où les règles peuvent contenir des constantes) : on peut montrer que R2 dépend de R1 si et seulement si un atome de l'hypothèse de R2 est **unifiable** avec la conclusion de R1.
[Rappel : deux atomes A1 et A2, *sans variables communes*, sont *unifiables* s'il existe une substitution S telle que S(A1) = S(A2) ; pour unifier une conclusion de règle avec un atome d'une hypothèse de règle, on renomme donc les variables si nécessaire]

→ Comment exploite-t-on ce graphe pour ne pas considérer toutes les règles à chaque étape ? A chaque étape, on ne considère que les successeurs des règles ayant alimenté « new ».

→ Peut-on intégrer les faits dans ce graphe ? Oui, comme des règles à hypothèse vide, qui deviennent des sources du graphe. Ceci permet de ne considérer à l'étape 1 que les règles successeurs des faits.

→ Si on cherche à répondre à une requête conjonctive Q (une conjonction d'atomes fermée existentiellement), peut-on encore restreindre l'ensemble des règles à considérer ? Oui, on classifie la question « comme si » elle était une règle de la forme "Q → gagné", où "gagné" est une prédicat 0-aire qui n'apparaît pas dans la base de connaissances : elle devient un puit du graphe. Ensuite, l'algorithme restreint la base de règles aux règles qui sont sur un chemin allant des faits à la question. Toute autre règle est inutile : soit parce qu'elle ne sera jamais applicable, soit parce qu'elle n'est d'aucune aide pour répondre à la question.

2.3. Chaînage arrière

La base de connaissances permet de prouver une requête conjonctive Q s'il existe une suite d'application de règles partant de la BF initiale et produisant une base de faits BF' sur laquelle Q s'envoie par homomorphisme. En chaînage arrière, on ne construit pas BF', on tente de reconstruire une suite d'applications de règles en « remontant » à partir de Q. Q est appelé le *but*.

Prenons le cas le plus simple : Q est un simple atome. Une règle R = H → C permet *éventuellement* de prouver Q s'il existe une substitution qui rend Q et C identiques (on considère que Q et C ont des ensembles de variables disjoints, au besoin on renomme les variables de R). Cette substitution s'appelle un *unificateur* de Q et C. On va unifier Q et C en cherchant un « unificateur le plus général », c'est-à-dire qui correspond à des modifications minimales de Q et C (voir une définition précise de cette notion dans le polycopié de logique 2, partie 2). Ceci garantit que si R est applicable en marche avant par un homomorphisme h, Q va pouvoir s'envoyer dans h(C).

Exemple :
R = p(x,y) ∧ q(y,z) → r(x,z,x)
Q = r(u,v,a) *// où a est une constante*
Un unificateur le plus général de Q et R est S ={(x,a), (z,v), (u,a)}. On a S(r(x,z,x)) = S(r(u,v,a)) = r(a,v,a). L'idée est que si R s'applique sur une BF en produisant un atome de la forme r(a,v,a), où v peut être remplacé par n'importe quelle constante, alors cette BF répondra à Q. Il faut maintenant vérifier que l'on arrive à prouver S(p(x,y) ∧ q(y,z)) c'est-à-dire (p(a,y) ∧ q(y,v)). Si, par exemple, la BF contient le fait p(a,b), ceci permet de prouver p(a,y) en substituant y par b, il ne reste plus alors qu'à prouver q(b,v). Bien sûr, cette tentative peut se solder par un échec, et il faut alors essayer d'autres choix de règles ou de faits possibles.

En général, on considère que Q n'est pas seulement un atome mais une liste d'atomes, et l'on cherche à prouver ces atomes l'un après l'autre (voir l'algorithme de chaînage arrière pour des règles d'ordre 0).

Algorithme BC(K,Q)

```
//Donnée : K = (BF,BR) et Q un but (liste d'atomes)
// Remarque : on voit un fait p(a1,…,ak) comme une règle → p(a1,…,ak)
// Résultat : vrai ssi Q peut être prouvé
Début
Si Q = vide alors retourner vrai
Pour toute règle R = H → C de BR et BF
  telle que premier(Q) s'unifie avec C par une substitution S
  // ceci peut nécessiter de renommer les variables de R
  // pour que les ensembles de variables de Q et R soient disjoints
    Q' ← Concaténer S(H) et S(reste(Q)) // calcul du nouveau but
    Si BC(K,Q') = vrai alors retourner vrai
FinPour
Retourner faux
Fin
```

Lorsque Q n'est plus une requête booléenne, on souhaite connaître toutes les réponses à Q, c'est-à-dire toutes les valeurs possibles pour les variables de Q. Il faut alors mémoriser les substitutions effectuées.

A chaque étape, le premier atome du but est unifié avec une conclusion de règle par une substitution S. La substitution Sf calculée pour les atomes déjà « effacés » est alors mise à jour par composition avec S. La composition S o Sf se calcule ainsi: pour tout (x,y) de Sf, s'il existe (y,e) dans S, alors (x,y) est remplacé par (x,e), sinon (x,y) est conservé ; et tous les couples de S sont ajoutés.

```
Algorithme BC(K,Q,Sf)
//Données : K = (BF,BR), Q un but (liste d'atomes non vide) et Sf la
substitution courante (au premier appel, Sf est vide)
//Résultat : la liste des substitutions associées aux preuves de Q

Début
  Réponses ← ∅
  Si Q = vide alors Réponses ← {Sf} // un ensemble réduit à Sf
  Sinon
   Pour toute règle R = H → C de BR et BF
    telle que premier(Q) s'unifie avec C par une substitution S
    // renommer les variables de R de façon à ce que l'ensemble des variables
    de R soit disjoint de l'ensemble de variables de Q et des variables
    apparaissant dans Sf (par exemple, à chaque unification, prendre pour les
    variables de la règle des noms non encore utilisés)
    Q' ← Concaténer S(H) et S(reste(Q))
    Sf ← S o Sf
   Réponses ← Réponses ∪ BC(K,Q',Sf)
   FinPour
  Retourner réponses
Fin
```

L'algorithme retourne un ensemble de substitutions, qu'il faut ensuite filtrer pour ne conserver que les couples de la forme (x,a) où x est une variable du but *initial* (seules ces variables nous intéressent pour donner une réponse).