Swan ROCHER

Outils d'Analyse d'une Base de Règles

2012 Université Montpellier 2

Table des matières

Chapitre 1

Contexte

Base de données sans ontologie =¿ ne permet pas de répondre à la requête-exemple. On ajoute l'ontologie =¿ on peut déduire de nouveaux faits, ...

Requête / base de connaissance (vision globale).

Universelles / Existentielles.

Non décidabilité de la réponse à une requête dans une base contenant des existentielles.

Exemples avec des MOTS.

Malgré le fait que de manière général, il n'existe aucun algorithme permettant de répondre à ce problème, certaines règles peuvent entrer dans des catégories (qui seront nommées *classes de règles*) qui en ajoutant des contraintes sur la forme des règles s'assurent que le problème soit décidable. Selon quelles contraintes sont satisfaites, il est nécessaire d'appliquer différentes méthodes de réponse sur différents sous-ensembles des règles.

L'objectif de ce TER est donc d'implémenter un outil permettant d'analyser une base de règles afin de construire son graphe de dépendances associés, de déterminer quelles contraintes sont satisfaites, sur quel sous-ensemble, et si la base est décidable d'en déduire quels algorithmes utiliser sur chacun d'eux. De plus, cet outil doit pouvoir charger des bases de règles à partir de fichiers, ainsi que les y écrire, et être suffisamment modulable pour permettre l'ajout de nouvelles vérifications de contrainte.

Chapitre 2

Notions de base

2.1 Prédicat

- Un prédicat noté $p\backslash n$ est un symbole relationnel d'arité n.
- On suppose que tout nom de prédicat est unique.
- On note p_i la i^{eme} position de p.

2.2 Atome

Un atome $a = p(a_1, a_2, ..., a_n)$ associe un terme à chaque position d'un prédicat $p\backslash n$. On note :

- a_i le terme en position i dans a. Un terme peut être une *constante* ou une *variable*. Une variable peut être *libre* ou *quantifiée* universellement (notée $\forall - var$) ou existentiellement (notée $\exists - var$).
- $dom(a) = \{a_i : \forall\ i \in [1, n]\}$, l'ensemble des termes de a
- $var(a)$ l'ensemble des variables de a
- $cst(a)$ l'ensemble des constantes de a

2.3 Conjonction d'atomes

Une conjonction de n atomes A est définie telle que :
$A = \bigwedge_{i=1}^{n} k_i$ avec $\forall\ i \in [1, n]$ $a_i = p_i(a_{i1}, a_{i2}, ..., a_{in_i})$ un atome de prédicat $p_i\backslash n_i$.

2.4 Représentation graphique d'une conjonction d'atomes

Une conjonction d'atomes peut être représentée par le graphe non orienté $G_A = (V_A, E_A, \omega)$ avec V_A son ensemble de sommets, E_A, son ensemble d'arêtes et ω une fonction de poids sur les arêtes construits de la manière suivante :

– $V_A = P_A \cup T_A$ avec $P_A = \{i : a_i \in A\}$ et $T_A = \{t_j \in dom(A)\}$
– $E_A = \{(i, t_j) : \forall a_i \in A, \forall t_j \in dom(a_i)\}$
– $\omega : E_A \to \mathbb{N}$ telle que $\omega(i, t_j) = j : \forall (i, t_j) \in E_A$

Cette représentation a de nombreux avantages, elle permet notamment de parcourir rapidement les atomes liés à un terme (et réciproquement), ainsi que de pouvoir être visualisée agréablement (voir figure 2.4).

FIGURE 2.1 – Exemple de représentation d'une conjonction d'atomes

On remarque que G_A admet une bipartition de ses sommets, en effet toutes les arêtes ont une extrémité dans P_A et l'autre dans T_A, or par construction $P_A \cap T_A = \emptyset$.

2.5 Règle

Une règle $R = (H, C)$ est constituée de deux conjonctions d'atomes H et C représentant respectivement l'hypothèse (le corps) et la conclusion (la tête) de R. Toutes les variables apparaissant dans H sont quantifiées universellement tandis que celles apparaissant uniquement dans C le sont existentiellement. Ainsi une règle est toujours sous la forme $R : \forall x_i (H \to \exists z_j (C))$.

On note :
– $dom(R) = dom(H) \cup dom(C)$
– $var(R) = var(H) \cup var(C)$
– $cst(R) = cst(H) \cup cst(C)$
– $fr(R) = var(H) \cap var(C)$ l'ensemble des variables frontières de R
– $cutp(R) = fr(R) \cup cst(R)$ l'ensemble des points de coupure de R

2.6 Représentation graphique d'une règle

Tout comme une simple conjonction d'atomes, une règle $R = (H, C)$ peut être représentée par un graphe similaire, en ajoutant une coloration à deux couleurs : une pour les atomes de l'hypothèse, l'autre pour ceux de la conclusion.

Ainsi le graphe associé $G_R = (V_R, E_R, \omega, \chi)$ est défini comme :
– $V_R = P_R \cup T_R$ avec $P_R = \{i : r_i \in R\}$ et $T_R = \{t_j \in dom(R)\}$
– $E_R = \{(i, t_j) : \forall r_i \in R, \forall t_j \in dom(R_i)\}$
– $\omega : E_R \to [1, p]$
  $\omega(i, t_j) = j : \forall (i, t_j) \in E_R$
– $\chi : P_R \to \{1, 2\}$
  $\chi(r_i) = 1$ si $r_i \in H$ et $\chi(r_i) = 2$ si $r_i \in C$.

Par exemple la règle $R : \forall x \forall y \ (salle(x) \wedge date(y) \wedge reservee(x, y) \to \exists z \ (cours(z) \wedge aLieu(z, x, y)))$ peut être visualisée de la façon suivante :

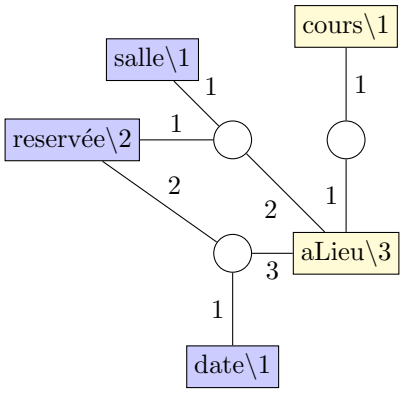

FIGURE 2.2 – Exemple de représentation d'une règle

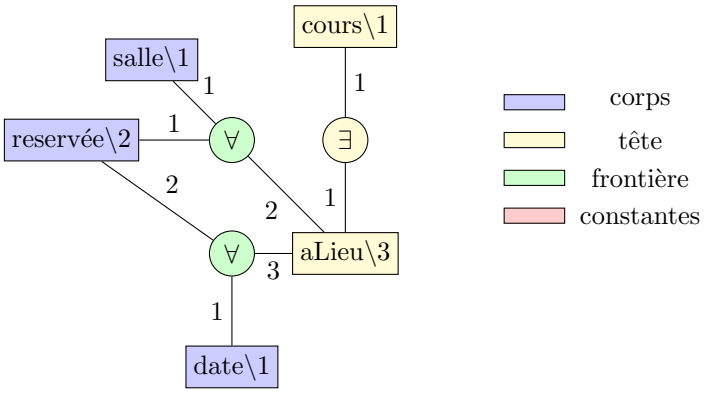

FIGURE 2.3 – Les différents éléments d'une règle

2.7 Règle à conclusion atomique

Une règle à conclusion atomique ajoute une contrainte sur la forme de sa conclusion qui ne doit contenir qu'un seul atome. Ces règles ont l'avantage d'être plus simples à unifier (voir section 3.2), et la plupart des algorithme présentés dans ce rapport sont plus efficaces sur ce type de règle, tandis que d'autres ne fonctionnent uniquement sur celles-ci.

Mais ceci n'est pas un problème puisqu'il est possible de réécrire une règle à conclusion non atomique en un ensemble de règles à conclusions atomiques équivalent.

En effet, quelle que soit une règle $R = (H, C)$, nous pouvons définir un nouveau prédicat p_R d'arité $|var(R)|$ ainsi qu'un nouvel ensemble de règles à conclusions atomiques R^A dont le premier élément aura la même hypothèse que R et une conclusion de prédicat p_R contenant toutes ses variables, et dont les suivants auront pour hypothèse cette nouvelle conclusion, et comme conclusion atomique les atomes de C.

Cet ensemble est donc défini de la manière suivante :
$R^A = \{R_i^A = (H_i^A, C_i^A) : \forall\ i \in [0, |C|]\}$, tels que :

$$R_0^A = \begin{cases} H_0^A = H, \\ C_0^A = p_R(\{x_j \in var(R)\}) \end{cases} \qquad \forall\ i \in [1, |C|]\ R_i^A = \begin{cases} H_i^A = C_0^A, \\ C_i^A = c_i \in C \end{cases}$$

Ainsi nous pouvons utiliser l'algorithme 1 afin d'effectuer cette conversion.

Algorithm 1 Conversion d'une règle à conclusion non atomique

Require: $R = (H, C)$: une règle quelconque
Ensure: R^A : un ensemble de règles à conlusions atomiques équivalent à R

```
1 H_0^A ← H
2 C_0^A ← p_R({x_i ∈ var(R)})
3 R^A ← {(H_0^A, C_0^A)}
4 for all atome c_i ∈ C do
5       R_i^A ← (C_0^A, c_i)
6       R^A ← R^A ∪ {R_i^A}
7 end for
8 return R^A
```

Toute règle pouvant donc se réécrire de manière équivalente en un ensemble de règles à conclusion atomique, dans la suite nous ne considèrerons que des règles sous cette forme.

2.8 Base de connaissance

Une base de connaissance $K = (F, R)$ est constituée d'un ensemble de *faits* représenté par une conjonction d'atomes F, ainsi que d'une *ontologie* représentée par un ensemble de règles R.

Les faits sont souvent considérés comme complètement instanciés, c'est à dire ne contenant que des constantes, mais ici, les règles contenant des variables existentielles peuvent générer de nouveaux individus. Donc nous définissons F comme une conjonction d'atomes existentiellement fermés.

2.9 Chaînage avant

Afin de déterminer si une requête bouléenne Q peut être déduite d'une base de connaissance $B = (R, F)$, le chaînage avant *dérive* de manière itérative la base de faits F (qui peut être vue comme un fait unique) par l'ensemble de règles R de manière à générer (en cherchant de nouveaux homomorphismes) de nouveaux faits qui devront à leur tour être étudiés. Avant chaque dérivation, l'algorithme vérifie si F^i contient Q auquel cas la réponse est positive, et si $F^i = F^{i-1}$ afin de savoir si la chaîne de dérivation est finie et dans ce cas donner une réponse négative.

2.10 Chaînage arrière

Le chaînage arrière quant à lui consiste à réécrire la requête Q juqu'à ce qu'elle soit dans F.

Chapitre 3

Graphe de dépendances des règles

Le graphe de dépendances de règles est une représentation d'une base de règles très intéressante. En effet il permet de vérifier rapidement quelles règles pourront éventuellement être déclenchées après l'application d'une règle donnée.

De plus il permet de déterminer l'appartenance à certaines classes de règles, et le calcul de ses composantes fortement connexes permet de "découper" la base de manière à effectuer les requêtes de manières différentes selon celles-ci.

3.1 Définition

Le graphe de dépendances des règles associé à une base de règles B_R est défini comme le graphe orienté $GRD = (V_{GRD}, E_{GRD})$ avec :

- $V_{GRD} = \{R_i \in B_R\}$,
- $E_{GRD} = \{(R_i, R_j) : \exists$ un bon unificateur $\mu : \mu(C_i) = \mu(H_j)\}$.

Intuitivement, on crée un sommet par règle et on relie R_i à R_j si R_i "peut amener à déclencher" R_j (R_j dépend de R_i). La notion d'unificateur est abordée dans la section suivante.

3.2 Unification de règles

Afin de pouvoir construire ce graphe, il faut donc pouvoir déterminer si une règle peut en déclencher une autre, c'est à dire s'il existe un unificateur entre la conclusion de la première et l'hypothèse de la seconde. Tout d'abord, l'unification est définie, s'en suit un algorithme permettant de vérifier si un tel unificateur existe, puis la correction de celui-ci ainsi que ses complexités.

3.2.1 Définitions

Substitution

Une substitution de taille n d'un ensemble de symboles X dans un ensemble de symboles Y est une fonction de X vers Y représentée par l'ensemble de couples suivants (avec $n \leq |X|$) :
- $s = \{(x_i, y_i) \ : \ \forall i \in [1, n] \ x_i \in X, \ y_i \in Y, \forall j \neq i \ x_i \neq x_j\}$
- $s(x_i) = y_i \ \forall i \in [1, n]$
- $s(x_i) = x_i \ \forall i \in [n + 1, |X|] \ x_i \in X$

Unificateur logique

Un unificateur logique entre deux atomes a_1 et a_2 est une substitution μ telle que :
- $\mu : var(a_1) \cup var(a_2) \rightarrow dom(a_1) \cup dom(a_2)$
- $\mu(a_1) = \mu(a_2)$

Cette définition s'étend aux conjonctions d'atomes.

Unificateur de conclusion atomique

Un unificateur de conclusion atomique est un unificateur logique $\mu = \{(x_i, t_i) : \forall i \in [1, n]\}$ entre l'hypothèse d'une règle $R_1 = (H_1, C_1)$ et la conclusion atomique d'une règle $R_2 = (H_2, C_2)$ et est défini de la manière suivante :
- $\mu : fr(R_2) \cup var(H_1) \rightarrow dom(C_2) \cup cst(H_1)$
- $\forall (x_i, t_i) \in \mu \ si \ x_i \in fr(R_2) \ alors \ t_i \in cutp(R_2) \cup cst(H_1)$

Bonne unification atomique

Un bon unificateur de conclusion atomique est un unificateur de conclusion atomique $\mu = \{x_i, t_i) : \forall i \in [1, n]\}$ entre un sous ensemble Q de l'hypothèse d'une règle $R_1 = (H_1, C_1)$ et la conclusion d'une règle atomique $R_2 = (H_2, C_2)$ tel que :
$\forall (x_i, t_i) \in \mu : si \ x_i \in H_1 \setminus Q \ alors \ t_i \ n'est \ pas \ une \ \exists - var$

Un tel ensemble Q est appelé un *bon ensemble d'unification atomique* de l'hypothèse de R_1 par la conclusion de R_2. On note que Q est donc défini comme suit :
- $Q \subseteq H_1$
- $\forall$ *position* i *de* $\exists - var$ *dans* $C_2, \forall$ *atome* $a \ \in Q, si \ a_i \in var(H_1) \ alors \ \forall$ *atome* $b \in H_1 : si \ \exists \ b_j \in b : a_i = b_j, alors \ b \in Q$

Bon ensemble d'unification atomique minimal

Un *bon ensemble d'unification atomique minimal* Q de H_1 par C_2 enraciné en a est défini tel que :
- Q est un bon ensemble d'unification atomique de H_1 par C_2
- $a \in Q$
- $|Q| = min(|Q_i| : Q_i \ est \ un \ bon \ ensemble \ d'unification \ atomique \ de \ H_1 \ par \ C_2 \ et \ a \in Q_i)$

3.2.2 Algorithmes

Vérifier qu'une règle atomique R_i peut déclencher R_j consiste donc à trouver un bon unificateur atomique entre R_i et R_j. Dans cette section, un algorithme permettant de répondre à ce problème est détaillé.

Dans la suite, les règles sont supposées représentées par des graphes (tels que définis en 2.6) et à conclusion atomique.

Le premier algorithme fait appel aux deux suivants de manière à déterminer si il existe au moins un unificateur entre les deux conjonctions d'atomes. En première phase, il vérifie l'exitence d'unificateurs avec chaque atome de manière indépendante. S'ensuit une extension à partir des atomes préselectionnés, et dès qu'un bon ensemble d'unification est entièrement unifié, l'algorithme s'arrête en répondant avec succès.

Algorithm 2 Unification

Require: H_1 : conjonction d'atomes, $R = (H_2, C_2)$: règle à conclusion atomique
Ensure: succès si C_2 peut s'unifier avec H_1, i.e. si $\exists H \subseteq H_1, \mu$ *une substitution* : $\mu(H_1) = \mu(C_2)$, échec sinon

```
▷ Précoloration
for all sommet atome a ∈ H1 do
    if UnificationLocale(a, R) ≠ échec then
        couleurLogique[a] ← noir
    else
        couleurLogique[a] ← blanc
    end if
end for
▷ Initialisation du tableau contenant les positions des variables existentielles de C2
E ← {i : ci est une ∃ − var de C2}
▷ Extension des ensembles
for all sommet atome a ∈ H1 : couleurLogique[a] = noir do
    if Q ← Extension(H1, a, couleurLogique, E) ≠ échec then
        if UnificationLocale(Q, R) ≠ échec then
            return succès
        end if
    end if
    couleurLogique[a] ← blanc
end for
return échec
```

Le deuxième algorithme est utilisé pour le calcul des bons ensembles d'unification à partir d'un atome racine. Tant qu'aucune erreur n'est détectée il *avale* les atomes voisins aux termes en positions existentielles. Les positions existentielles sont les indices des variables existentielles dans l'atome de conclusion.

Algorithm 3 Extension

Require: H_1 : conjonction d'atomes, $a \in H_1$: sommet atome racine, *couleurLogique* : tableau de taille égal au nombre d'atomes dans H_1 tel que $couleurLogique[a] = noir$ ssi $UnificationLocale(a, R) =$ succès, E : ensemble des positions des variables existentielles

Ensure: Q : bon ensemble d'unification minimal des atomes de H_1 construit à partir de a s'il existe, échec sinon.

```
1  ▷ Initialisation du parcours
2  for all sommet atome a ∈ H1 do
3      if couleurLogique[a] = noir then
4          for all sommet terme t ∈ voisins(a) do
5              couleur[t] = blanc
6          end for
7          couleur[a] = blanc
8      end if
9  end for
10 couleur[a] ← noir
11 Q ← {a} ▷ conjonction d'atomes à traiter
12 attente ← {a} ▷ file d'attente du parcours
13 while attente ≠ ∅ do
14     u ← haut(attente)
15     if u est un atome then
16         for all i ∈ E do
17             v ← voisin(u, i)
18             if v est une constante then
19                 return échec
20             else if couleur[v] = blanc then
21                 ▷ v est une ∀ − var non marquée par le parcours
22                 couleur[v] ← noir
23                 attente ← attente ∪ {v}
24             end if
25         end for
26     else
27         ▷ u est un terme
28         for all v ∈ voisins(u) do
29             if couleurLogique[v] = blanc then
30                 return échec
31             else
32                 if couleur[v] = blanc then
33                     couleur[v] ← noir
34                     attente ← attente ∪ {v}
35                     Q ← Q ∪ {v}
36                 end if
37             end if
38         end for
39     end if
40 end while   ▷ Fin du parcours
41 return Q
```

Remarque :
La phase d'initialisation du parcours pourrait simplement parcourir tous les sommets de H_1 et mettre leur couleur à blanc. En pratique, cette solution serait sans doute plus efficace, mais dépendrait donc du nombre de sommets total dans H_1. Ce qui en théorie amènerait la complexité de cette boucle en $\bigcirc(nombre\ d'atomes\ \times arite\ max\ de\ H_1)$. Or ici, la complexité ne dépend pas de cette arité max, mais uniquement de l'arité du prédicat de la conclusion C_2.

Le dernier algorithme est celui qui teste réellement si il existe un unificateur entre une conclusion atomique, et un bon ensemble d'unification atomique minimal. Il est appelé une première fois pour tester les atomes de la conjonction séparement, et permettre une préselection des atomes (qui vont servir de racine). Durant la dernière phase (lorsqu'il est appelé sur les ensembles étendus), s'il trouve un unificateur, celui-ci assure que la règle peut amener à déclencher la conjonction.

Algorithm 4 UnificationLocale

Require: H_1 : conjonction d'atomes, $R = (H_2, C_2)$: règle à conclusion atomique
Ensure: succès si C_2 peut s'unifier avec H_1

▷ Vérification des prédicats
for all *atome* $a \in H_1$ **do**
  if prédicat(a) $\neq$ prédicat(C_2) **then**
    return échec
  end if
end for
$u \leftarrow \emptyset$ ▷ substitution
for all *terme* $t_i \in C_2$ **do**
  ▷ def : a_i = terme de a en position i
  $E \leftarrow \{a_i : \forall$ *atome* $a \in H_1\}$
  if t_i *est une constante* **then**
    if $\exists\ v \in E : v$ *est une constante et* $v \neq t_i$, *ou* v *est une* $\exists -$ *variable* **then**
      return échec
    else
      $u \leftarrow \{(v, t_i) : v \in E$ *et* $v \neq t_i\}$
    end if
  else if t_i *est une* $\exists -$ *variable* **then**
    if $\exists\ v \in E : v$ *est une* $\exists -$ *variable et* $v \neq t_i$, *ou* v *est une constante* **then**
      return échec
    else
      $u \leftarrow \{(v, t_i) : v \in E$ *et* $v \neq t_i\}$
    end if
  else
    if $\exists\ v_1, v_2 \in E : v_1 \neq v_2$ *et* v_1, v_2 *ne sont pas des* $\forall -$ *variables* **then**
      return échec
    else if $\exists\ c \in E : c$ *est une constante* **then**
      $u \leftarrow \{(v, c) : v \in E \cup \{t_i\}$ *et* $v \neq c\}$
    else
      $u \leftarrow \{(v, t_i) : v \in E$ *et* $v \neq t_i\}$
    end if
  end if
  $H_1 \leftarrow u(H_1)$
  $C_2 \leftarrow u(C_2)$
end for
return succès

3.2.3 Correction

3.2.4 Complexites

On note :
– n = nombre d'atomes dans H_1
– p = arité de C_2

– t = nombre de termes "colorables" dans H_1
– m = nombre d'arêtes "suivables" dans H_1
On remarque que dans le pire des cas on a :
– $t = n \times p$
– $m = n \times p$

Temps

– UnificationLocale (pire des cas) = $\bigcirc(np)$
– Extension (pire des cas) = $\bigcirc(np)$
– Unification (pire des cas) = $\bigcirc(n^2p)$

Extension
On effectue simplement un parcours en largeur à partir d'un sommet donné qui peut éventuellement s'arrêter plus tôt qu'un parcours classique. La complexité en temps dans le pire des cas est donc au plus la même, c'est à dire linéaire au nombre de sommets plus le nombre d'arcs. Le graphe représentant la conjonction d'atomes H_1 possèdent $n \times ariteMax(H_1)$ arêtes et $n \times (ariteMax(H_1) + 1)$ sommets. On sait donc que $C^{temps}_{Extension} = \bigcirc(n \times ariteMax(H_1))$.

Deux cas :
i) Découverte d'un sommet atome :
parcours classique =¿ on récupère tous les voisins blancs
parcours extension =¿ on récupére tous les voisins blancs de couleur logique noire (en effet arrêt immédiat si un voisin de couleur logique noire est découvert.
ii) Découverte d'un sommet variable :
pas de différence
On a donc que l'algo ne suit que les sommets atomes pouvant être unifies localement. C'est à dire (entre autres) que leur prédicat est égal à prédicat(C_2).

...

Espace

3.3 Composantes fortement connexes

Les composantes fortement connexes du graphe de dépendances sont utilisées pour le *découper* de façon à attribuer des étiquettes différentes à celles-ci en fonction des classes concrètes auxquelles son ensemble de règles appartient.

De plus une fois chaque sous ensemble étiqueté, il faut encore vérifier que celles-ci sont compatibles entre elles.

Pour cela, nous définissons le graphe orienté des composantes fortement connexes associé tel que son ensemble de sommets est l'ensemble des composantes du graphe, et qu'il existe un arc entre deux composantes C_i et C_j si et seulement s'il existe un arc d'un sommet de C_i vers un sommet de C_j. Par définition des composantes fortement connexes,

ce graphe est évidemment sans circuit, et les arcs de celui-ci influent directement sur la décidabilité de l'ensemble de règles.

On dit que C_i *précède* C_j s'il n'existe aucun arc de C_j vers C_i, et on note cette relation $C_i \triangleright C_j$.

De plus, on associe à chaque C_i une étiquette qui déterminera la classe abstraite considérée pour cette composante. Une condition (suffisante)pour que l'ensemble de règles soit décidable est la suivante :
$\{C_i : etiquette(C_i) = FES\} \triangleright \{C_i : etiquette(C_i) = GBTS\} \triangleright \{C_i : etiquette(C_i) = FUS\}$
C'est à dire qu'aucune règle FES ne doit dépendre d'une règle FUS ou $GBTS$, et qu'aucune règle $GBTS$ ne doit dépendre d'une règle FUS .

En effet les algorithmes de chaînage arrière par exemple réécrivent la requête jusqu'à ce qu'elle corresponde à la base, tandis que ceux avant ajoutent des faits jusqu'à obtenir la requête. Il est donc évident que si une composante n'accepte que le chaînage arrière, il ne doit exister aucune règle de celle-ci de laquelle dépende une règle de la composante acceptant uniquement le chaînage avant (si tel était le cas, cette règle ne serait jamais déclenchée).

Chapitre 4

Classes de règles

4.1 Classes abstraites

Trois classes abstraites ont été définies, chacune permettant l'usage de certains algorithmes sur l'ensemble de règles considéré. Elles sont dites *abstraites* puisque déterminer si un ensemble de règles appartient à l'une de ces classes est un problème non décidable. De plus elles sont incomparables entre elles, et non exclusives.

4.1.1 Finite Expansion Set

La première classe est définie comme assurant la finition des algorithmes de chaînage avant. Ainsi tout ensemble de règles appartenant à cette classe peut être utilisé pour les dérivations de ces algorithmes. Dans le cas de certaines classes, il est par contre nécessaire d'ajouter des conditions d'arrêt, celles-ci sont détaillées plus loin.

4.1.2 Finite Unification Set

La deuxième classe abstraite quant à elle assure la finition des algorithmes de chaînage arrière. De la même manière que précedemment il est parfois nécessaire de modifier les conditions d'arrêt.

4.1.3 Bounded Treewidth Set

La dernière définit les ensembles de règles où la production de nouvelles règles suit la forme d'un arbre. Cette classe ne permet pas l'utilisation direct d'algorithmes, mais par contre la classe abstraite Greedy Bounded Treewidth qui est une spécialisation de celle-ci, s'assure que le chaînage avant s'exécute en temps fini, et ce dès qu'un algorithme glouton de ... est utilisé.

4.2 Classes concrètes

4.2.1 Acyclicité du graphe de dépendance des règles

Le seul fait que le graphe de dépendance soit sans circuit suffit à certifier que le chaînage avant et arrière s'exécutent en temps fini, impliquant que si cette contrainte est satisfaite, l'ensemble des règles appartient à FES et à FUS .

4.2.2 Domaine restreint

Un ensemble de règles satisfait cette contrainte si tous les atomes de leur conclusion contiennent soit toutes les variables de l'hypothèse, soit aucune. Dans le cas des règles à conclusion atomique, cela revient à s'assurer que la frontière de chaque règle est soit égale à 0 soit au nombre de variables universelles. Cette contrainte est suffisante pour que l'ensemble de règles appartienne à FUS .

4.3 Combinaisons

Chapitre 5

Implémentation

5.1 Conjonction d'atomes

TODO =¿ à adapter à la section

Par la suite on supposera que le graphe G_K associé à la conjonction K est implémenté par deux ensembles de sommets (les atomes et les termes), ainsi que par une liste de voisinage pour chaque sommet. De plus, chaque sommet peut avoir connaissance de son type de contenu parmis : *atome*, *terme*, *constante*, $\forall - var$, $\exists - var$ en temps constant.

On note n (resp. t) le nombre d'atomes (resp. de termes) dans K.

Opérations (et temps d'exécution associé) :
– Parcourir les sommets *atomes* : n.
– Parcourir tous les sommets : $n + t$.
– Accéder au i^{eme} voisin d'un sommet : *constant*.

Chapitre 6

Perspectives