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).

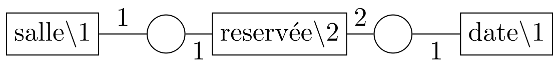

FIGURE 2.1 – Exemple de représentation d'une conjonction d'atomes

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 :

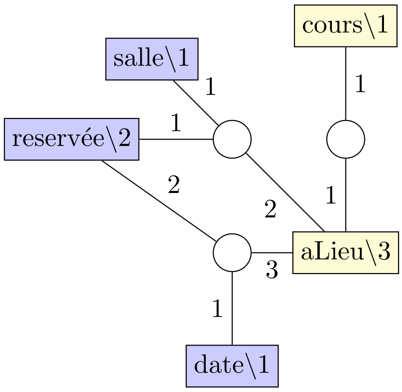

FIGURE 2.2 – Exemple de représentation d'une règle

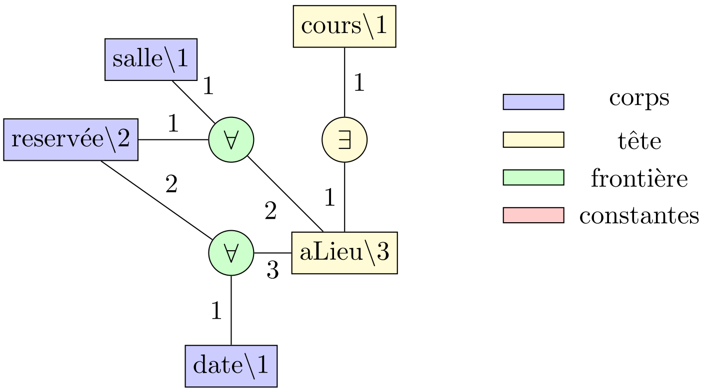

FIGURE 2.3 – Les différents éléments d'une règle

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 :

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

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
$H_0^A \leftarrow H$
$C_0^A \leftarrow p_R(\{x_i \in var(R)\})$
$R^A \leftarrow \{(H_0^A, C_0^A)\}$
for all atome $c_i \in C$ **do**
    $R_i^A \leftarrow (C_0^A, c_i)$
    $R^A \leftarrow R^A \cup \{R_i^A\}$
end for
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 $\exists H \subseteq H_1, \mu\ \mathit{une\ substitution} : \mu(H_1) = \mu(C_2)$, échec sinon

▷ Précoloration
for all *sommet atome* $a \in H_1$ **do**
    if $UnificationLocale(a, R) \neq$ échec **then**
        $couleurLogique[a] \leftarrow noir$
    else
        $couleurLogique[a] \leftarrow blanc$
    end if
end for
▷ Initialisation du tableau contenant les positions des variables existentielles de C_2
$E \leftarrow \{i : c_i\ est\ une\ \exists - var\ de\ C_2\}$
▷ Extension des ensembles
for all *sommet atome* $a \in H_1 : couleurLogique[a] = noir$ **do**
    if $Q \leftarrow Extension(H_1, a, couleurLogique, E) \neq$ échec **then**
        if $UnificationLocale(Q, R) \neq$ échec **then**
            return succès
        end if
    end if
    $couleurLogique[a] \leftarrow blanc$
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.

```
▷ Initialisation du parcours
for all sommet atome a ∈ H1 do
    if couleurLogique[a] = noir then
        for all sommet terme t ∈ voisins(a) do
            couleur[t] = blanc
        end for
        couleur[a] = blanc
    end if
end for
couleur[a] ← noir
Q ← {a} ▷ conjonction d'atomes à traiter
attente ← {a} ▷ file d'attente du parcours
while attente ≠ ∅ do
    u ← haut(attente)
    if u est un atome then
        for all i ∈ E do
            v ← voisin(u, i)
            if v est une constante then
                return échec
            else if couleur[v] = blanc then
                ▷ v est une ∀ − var non marquée par le parcours
                couleur[v] ← noir
                attente ← attente ∪ {v}
            end if
        end for
    else
        ▷ u est un terme
        for all v ∈ voisins(u) do
            if couleurLogique[v] = blanc then
                return échec
            else
                if couleur[v] = blanc then
                    couleur[v] ← noir
                    attente ← attente ∪ {v}
                    Q ← Q ∪ {v}
                end if
            end if
        end for
    end if
end while ▷ Fin du parcours
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 $\bigcirc(\textit{nombre d'atomes} \times \textit{arite max de } H_1)$. 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 ∈ H1 do
    if prédicat(a) ≠ prédicat(C2) then
        return échec
    end if
end for
u ← ∅   ▷ substitution
for all terme ti ∈ C2 do
    ▷ def : ai = terme de a en position i
    E ← {ai : ∀ atome a ∈ H1}
    if ti est une constante then
        if ∃ v ∈ E : v est une constante et v ≠ ti, ou v est une ∃ − variable then
            return échec
        else
            u ← {(v, ti) : v ∈ E et v ≠ ti}
        end if
    else if ti est une ∃ − variable then
        if ∃ v ∈ E : v est une ∃ − variable et v ≠ ti, ou v est une constante then
            return échec
        else
            u ← {(v, ti) : v ∈ E et v ≠ ti}
        end if
    else
        if ∃ v1, v2 ∈ E : v1 ≠ v2 et v1, v2 ne sont pas des ∀ − variables then
            return échec
        else if ∃ c ∈ E : c est une constante then
            u ← {(v, c) : v ∈ E ∪ {ti} et v ≠ c}
        else
            u ← {(v, ti) : v ∈ E et v ≠ ti}
        end if
    end if
    H1 ← u(H1)
    C2 ← u(C2)
end for
return succès
```

3.2.3 Correction

3.2.4 Complexites

Soit R une règle, et G_R son graphe associé. On note :

- k : nombre d'atomes dans R
- p : arité maximum des prédicats de R

- $t \leq n \times p$: nombre de termes de R

et :

- $n = k + t$: nombre de sommets dans G_R
- $m \leq n \times p$: nombre d'arêtes dans G_R

Avec notre représentation nous avons donc les complexités suivantes :

- Parcourir les sommets prédicats : $\bigcirc(k)$.
- Parcourir tous les sommets : $\bigcirc(n)$.
- Accéder au i^{eme} voisin d'un sommet : $\bigcirc(1)$.

- 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

TODO REFS

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 Faiblement acyclique

Cette classe est un peu particulière puisqu'elle demande la génération d'une autre structure de graphes et qu'elle s'applique directement sur un ensemble de règles et pas uniquement sur chacune des règles indépendamment des autres.

On crée donc un graphe de dépendances des positions dont les sommets sont les positions des prédicats et dont la construction des arcs est la suivante : pour chaque variable x d'une règle R apparaissant dans l'hypothèse en position p_i, si x appartient à la frontière de R alors il existe un arc de p_i vers chacune des positions r_j de la conclusion de R dans laquelle apparaît x, de plus pour chacune des variables existentielles apparaîssant en position q_k il existe un arc *spécial* de p_i vers q_k.

Si le graphe de dépendances des positions associés à un ensemble de règle ne contient aucun circuit passant par un arc *spécial* alors il est dit *faiblement acyclique* et appartient à *FES* .

4.2.3 Domaine restreint

Une règle satisfait cette contrainte si tous les atomes de sa 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* .

4.2.4 Guardée

Une règle gardée est définie comme étant une règle dont un atome de son hypothèse contient toutes les variables de celle-ci. Si toutes les règles de l'ensemble contiennent un garde, alors l'ensemble appartient à *GBTS* .

4.2.5 Frontière gardée

On dit qu'une règle a une frontière gardée si un atome de son hypothèse possède toutes les variables de la frontière. On peut remarquer que cette classe est une généralisation de la précédente. Et dans le cas où toutes les règles de l'ensemble possède cette propriété, celui-ci appartient alors à *GBTS* .

4.2.6 Frontière-1

Cette classe contient les règles dont la frontière est de taille 1, elle est donc une spécialisation des règles à frontière gardée, Ainsi un ensemble de règles dont toutes les frontières sont de taille 1 appartient à *GBTS* .

4.2.7 Hypothèse atomique

Les règles ne contenant que des hypothèses atomiques s'assurent que la règle est *gardée (4.2.4)*, et de plus assurent que les algorithmes basés sur le chaînage arrière se terminent en temps fini. Donc si toutes les règles d'un ensemble sont à hypothèse atomique alors celui-ci est *GBTS* et *FUS* .

4.2.8 Règle déconnectée

Une règle est dite déconnectée si sa frontière est vide, cette classe est donc une spécialisation des règles à *domaine restreint (4.2.3)*, à *frontière gardée (4.2.5)* et *faiblement acyclique (4.2.2)*. Ce type de règle n'est pas très utilisé étant donné que seules des constantes sont partagées entre l'hypothèse et la conclusion ce qui limite leur usage, mais elles ont l'avantage d'être à la fois *FES* , *GBTS* et *FUS* .

4.3 Combinaisons

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.

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.

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 5

Implémentation

5.1 Structures de donnée

Nous avons donc choisi d'utiliser la représentation graphique des règles telle que définie dans la section 2.6. Ainsi une structure de graphe biparti non orienté a été mise en place, l'ensemble des arêtes est codé par une liste de voisinage pour chaque sommet. En effet l'opération de parcours des voisins doit être la plus efficace possible, étant donné que les atomes ne sont pas stockés directement dans le graphe.

De plus, le graphe de dépendances des règles est quant à lui un graphe orienté dont les arcs sont également codés par listes de voisinage. Il est évident que les arcs d'un graphe de dépendances ont plus intérêt à être implémentés de cette manière puisque l'objectif de construire un tel graphe est de connaître rapidement de quel sommet dépend quel autre.

5.2 GRDAnalyser

Les règles (les données) sont donc stockées directement dans le graphe de dépendances des règles dont une instance est encapsulée dans le *GRDAnalyser*. De plus le *GRDAnalyser*est en fait constitué de deux parties distinctes supplémentaires : la première est en charge de la détermination des classes concrètes, tandis que la seconde vérifie si la base est décidable et combine les classes abstraites de manière à savoir quels algorithmes utiliser.

5.3 Détermination d'une classe concrète

En section 4.2 nous avons défini de nombreuses classes de règles qu'il faut donc pouvoir déterminer. Pour cela, nous avons déclaré une interface de fonction *DecidableClassCheck* fournissant une méthode renvoyant une étiquette à partir d'un ensemble de règles. L'analyseur de classes concrètes contient une liste des contraintes à tester, et lors de son exécution, il vérifie tout d'abord l'ensemble complet des règles sur chacune de celles-ci, puis ensuite sur chaque composante fortement connexe du graphe de dépendances.

5.4 Combinaison des classes abstraites

Comme expliqué plus en détails dans la section 4.3 il est ensuite nécessaire de vérifier si l'ensemble est bel et bien décidable. L'analyseur de classes abstraites regarde donc tout d'abord si l'ensemble des règles est étiqueté par une classe concrète, si tel est le cas, une des approches pour répondre à une requête est donc d'exécuter l'algorithme correspondant. Si ce n'est pas le cas, il effectue un parcours en largeur du graphe des composantes fortement connexes à partir de l'ensemble des sources de celui-ci, attribuant à chaque sommet découvert (qu'il soit déjà traité ou non) la plus petite étiquette fournie par ses classes concrètes et supérieure à celle de son prédécesseur ou 0 si ce n'est pas possible. Une fois cette opération effectuée, si tous les sommets sont étiquetés par des valeurs strictement positives, l'ensemble de règles est décidable.

5.5 Formats de fichiers

Le graphe de dépendances des règles est capable de charger une base à partir d'un fichier, celui-ci devant être écrit dans un format spécifique : chaque ligne doit être une règle de la forme suivante : $atome_1; atome_2; ...; atome_n - - > atome_c$ avec n le nombre d'atomes dans l'hypothèse et $atome_c$ l'unique atome de la conclusion et où chaque atome i est écrit : $p_i(t_{i1}, t_{i2}, ..., t_{ik})$. Les termes sont considérés des constantes s'ils sont encadrés par des simple quotes.
Par exempe la règle $\forall x(\rightarrow (\exists z))$ doit être écrite :

En plus du format interne ci-dessus, il est également possible de fournir un fichier Datalog (.dtg) qui ne contient que des règles à hypothèse atomique. Chaque ligne est soit une règle, soit un commentaire auquel cas elle doit débuté par //. Ici les règles sont sous le format suivant : $[!]atome_c : -atome_h$. Le point d'explamation est utilisé pour signaler la négation d'une conclusion, celle-ci sera convertie en une règle contenant ses deux atomes actuels dans son hypothèse et ayant une conclusion au prédicat spécial *ABSURD*. De plus, les termes sont maintenant considérés comme des variables s'ils commencent par un point d'interrogation.
Ainsi la ligne du fichier correspondant à la règle $\forall x(\rightarrow (\exists z))$ doit être :
.

Chapitre 6

Perspectives

Réalisé avec $\LaTeX$