

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

CONTRAINTES D'IDENTIFICATION DANS
mathcal{SHOIN}(\mathbf{D})

Thi Dieu Thu NGUYEN, Nhan LE-THANH

Projet MAINLINE

Rapport de recherche
ISRN I3S/RR-2006-34-FR

Novembre 2006

RÉSUMÉ :

Ces dernières années, l'intégration sémantique de données dans l'environnement du Web sémantique a connu un intérêt grandissant. Le but du Web sémantique est d'accéder, de relier et de combiner des connaissances à partir de plusieurs sources. Des besoins d'intégration de sémantique dans cet environnement depuis des sources de données relationnelles apparaissent. Cependant, il y a un aspect important des schémas de base de données que OWL (Ontology Web Language) jusqu'ici n'a pas encore capturé, à savoir la contrainte d'identification. Pour traiter ce problème, nous proposons dans cet article un langage de description appelé $\mathit{SHOINK}(\mathit{D})$ qui étend $\mathit{SHOIN}(\mathit{D})$ (un sous-langage de base de OWL-DL) capturant complètement la sémantique de telles contraintes. Nous traitons également le problème de raisonnement dans ce langage en construisant un algorithme du Tableau, prouvant que la satisfiabilité d'une base de connaissances en présence de ces contraintes peut être réalisée.

MOTS CLÉS :

Logiques de Description, Web sémantique, Base de connaissances,

ABSTRACT:

In recent years, there has been a growing interest in semantic integration in the semantic web environment, whose goal is to access, relate and combine knowledge from multiple sources. The needs of integrating the semantics from relational data sources into this environment therefore is also emerged. However, there is one important aspect of database schemas that OWL (Ontology Web Language) up to now has not captured yet, namely the identification constraint. To address this problem, in this paper we propose a description language called $\mathit{SHOINK}(\mathit{D})$ which extends $\mathit{SHOIN}(\mathit{D})$ (a sublanguage underlying OWL-DL) fully capturing the semantics of such constraints. We also address the problem of reasoning in this language by building a Tableau algorithm, showing that the knowledge base satisfiability and logical implication in the presence of these constraints can be realized.

KEY WORDS :

Description Logics, Semantic Web, Knowledge base, Databases Schemas,

Contrainte d'identification dans $SHOIN(\mathbf{D})$

Thi Dieu Thu NGUYEN — Nhan LE-THANH

Projet MAINLINE - Laboratoire I3S (CNRS - UNSA)

Les Algorithmes - Bât Euclide B

2000 route des Lucioles – B.P. 121

F-06903 Sophia Antipolis Cedex

tnghuyen@i3s.unice.fr;nhan.-le-thanh@unice.fr

RÉSUMÉ. Ces dernières années, l'intégration sémantique de données dans l'environnement du Web sémantique a connu un intérêt grandissant. Le but du Web sémantique est d'accéder, de relier et de combiner des connaissances à partir de plusieurs sources. Des besoins d'intégration de sémantique dans cet environnement depuis des sources de données relationnelles apparaissent. Cependant, il y a un aspect important des schémas de base de données que OWL¹ jusqu'ici n'a pas encore capturé, à savoir la contrainte d'identification. Pour traiter ce problème, nous proposons dans cet article un langage de description appelé $SHOINK(\mathbf{D})$ qui étend $SHOIN(\mathbf{D})$ (un sous-langage de base de OWL-DL) capturant complètement la sémantique de telles contraintes. Nous traitons également le problème de raisonnement dans ce langage en construisant un algorithme du Tableau, prouvant que la satisfiabilité d'une base de connaissances en présence de ces contraintes peut être réalisée.

ABSTRACT. In recent years, there has been a growing interest in semantic integration in the semantic web environment, whose goal is to access, relate and combine knowledge from multiple sources. The needs of integrating the semantics from relational data sources into this environment therefore is also emerged. However, there is one important aspect of database schemas that OWL² up to now has not captured yet, namely the identification constraint. To address this problem, in this paper we propose a description language called $SHOINK(\mathbf{D})$ which extends $SHOIN(\mathbf{D})$ (a sublanguage underlying OWL-DL) fully capturing the semantics of such constraints. We also address the problem of reasoning in this language by building a

1. OWL (Ontology Web Language) est un langage populaire pour concevoir des ontologies dans l'environnement du Web sémantique, recommandé par le consortium World Wide Web (W3C).
2. OWL (Ontology Web Language) is a popular language for designing the ontologies in the semantic web environment, recommended by World Wide Web Consortium (W3C).

Tableau algorithm, showing that the knowledge base satisfiability and logical implication in the presence of these constraints can be realized.

MOTS-CLÉS : Logiques de Description, Web sémantique, Base de connaissances, Schémas de bases de données, Contrainte d'identification.

KEYWORDS: Description Logics, Semantic Web, Knowledge base, Databases Schemas, Uniqueness constraint.

Table des matières

1	Introduction	3
2	Logique de description $SHOIN(\mathbf{D})$ et influence du domaine concret	4
2.1	Type de données concret	5
2.2	Syntaxe et sémantique	5
2.3	Influences du domaine concret	7
3	Contrainte d'identification et $SHOINK(\mathbf{D})$	8
3.1	Définition	8
3.2	CI et le problème avec le domaine concret	9
3.3	CI et le problème avec la procédure de raisonnement pour $SHOINK(\mathbf{D})$	11
4	L'algorithme du Tableau pour $SHOINK(\mathbf{D})$	13
4.1	Tableau pour $SHOINK(\mathbf{D})$	13
4.2	Construction du Tableau pour $SHOINK(\mathbf{D})$	18
5	Discussion et conclusion	21
6	Bibliographie	25

Liste des tableaux

1	Règles d'extension pour $SHOIN(\mathbf{D})$	22
2	Nouvelles règles d'extension pour $SHOINK(\mathbf{D})$	23

1. Introduction

Ces dernières années, le problème d'interopérabilité et d'intégration sémantique des sources de données hétérogènes sur le web a attiré l'attention particulière des chercheurs de plusieurs domaines, regroupés dans une tendance, dite Web sémantique. Une des fonctionnalités principales du Web sémantique est la capacité de décrire de manière déclarative des sources de données sur le Web. Cette annotation doit être suffisamment expressive pour répondre au problème de l'hétérogénéité des sources de données mais elle doit être également bien formalisée pour pouvoir servir dans un

raisonnement automatique. Conçu dans ce but, OWL-DL (Ontology Web Language - Description Logic) est un langage de description des sources de données proposé par W3C¹. Une base de connaissances représentée dans le langage OWL peut être utilisée dans les services d'inférence grâce à une traduction en langage de logique de description $SHOIN(\mathbf{D})$. La motivation de notre travail est de résoudre l'intégration de sources de données relationnelles dans l'environnement du Web sémantique. Cette intégration sémantique peut être vue comme la capacité de l'OWL de représenter un schéma conceptuel relationnel et les contraintes d'intégrité du modèle relationnel dont la contrainte d'identification. Cette contrainte, bien présentée dans le modèle relationnel avec les concepts de clé primaire et de clé étrangère, n'est pas prise en compte naturellement dans la logique de description (LD). L'introduction de cette contrainte dans une LD expressive est donc un pas décisif pour l'intégration des sources de données relationnelles dans le Web sémantique. Nous abordons ce problème Dans cet article. Nous proposons un langage de description, une extension de $SHOIN(\mathbf{D})$ appelé $SHOINK(\mathbf{D})$, qui est capable de capturer la sémantique de la contrainte d'identification. Nous traitons également le problème de raisonnement dans ce langage en proposant un algorithme de Tableau spécifique. Cet algorithme est une extension de celui proposé par Ian Horrocks et Ulrike Sattler dans (Horrocks *et al.*, 2005). Enfin nous proposons une étude des propriétés (complétude et correction) et de la complexité de l'algorithme.

Le reste de cet article est organisé comme suit. La section 2 présente le langage $SHOIN(\mathbf{D})$ comme la base de notre approche. Nous montrons quelques problèmes découlant de l'intégration du domaine concret sur $SHOIN$. Ainsi une solution est proposé également dans cette section (remarquez que $SHOIN(\mathbf{D})$ n'est pas encore décrit complètement dans la littérature). Dans la section 3, nous introduisons la notion de la contrainte d'identification qui est ajouté à $SHOIN(\mathbf{D})$ et fait créer son extension $SHOINK(\mathbf{D})$. Dans cette section, nous discutons des problèmes surgissants lors de l'ajout et proposons une solution pour construire l'algorithme. La section suivante est consacrée à la construction d'un algorithme du Tableau pour ce nouveau langage étendu. Nous démontrons que notre algorithme est correct, complet et surtout se termine. L'évaluation de la complexité de l'algorithme est également discutée dans la dernière section. Pour la conclusion, nous présentons un panorama des travaux relatifs dans la littérature et quelques perspectives de nos futurs travaux de recherche.

2. Logique de description $SHOIN(\mathbf{D})$ et influence du domaine concret

Nous décrivons le langage $SHOIN(\mathbf{D})$ qui sert de base à notre extension. Remarquez que ce langage jusqu'à aujourd'hui n'est pas encore entièrement décrit dans la littérature. Bien qu'il ait la même syntaxe et sémantique pour les mêmes constructeurs pour $SHOIQ$ et emploie le même domaine concret comme pour $SHOQ(\mathbf{D})$, les restrictions concrètes de cardinalité sont la nouvelle caractéristique qui provoque quelques problèmes qui sont présentés et résolus dans cette section.

1. World Wide Web Consortium, <http://www.w3.org/>

2.1. Type de données concret

Le domaine concret dans $SHOIN(\mathbf{D})$ est constitué de types concrets de données qui sont définis par les systèmes de type externes (e.g le système de types du schéma XML). Par exemple, dans le système de types du schéma XML le type de données *nonNegativeInteger* est défini par le type *integer* dont les valeurs sont restreintes à plus ou égales zéro. Les types de données constituent un *domaine concret universel* \mathbf{D} dans lequel, les prédicats représentant les types de données sont unaires. Le nom du prédicat est le nom du type de données. Le paramètre du prédicat est une variable spécifiée à l'intérieur du le domaine du prédicat. Par exemple, avec le type de données (*min21*) défini dans \mathbf{D} comme un ensemble des valeurs du type *integer* plus ou égales 21, on peut représenter le concept "personnes qui ont aux moins 21 ans" comme $\text{Personne} \sqcap \exists \text{age} . (\text{min21})$ où *Personne* est un concept et *age* est un rôle concret. Formellement, on peut définir le domaine concret universel comme la définition 2.1.

Définition 2.1 (Domaine concret universel (Pan, 2004)) *Un domaine concret universel* \mathbf{D} est une paire $(\Delta_{\mathbf{D}}, \Phi_{\mathbf{D}})$, où $\Phi_{\mathbf{D}}$ est l'ensemble de noms des types de données (prédicats unaires) et $\Delta_{\mathbf{D}}$ est le domaine de tous les types de données. Chaque nom de type de données $d \in \Phi_{\mathbf{D}}$ associe à un ensemble $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$. Soit \mathbf{V} un ensemble des variables, une conjonction de type de données sous forme

$$c = \bigwedge_{j=1}^k d_j(v^{(j)}) \quad [1]$$

où $d_j \in \Phi_{\mathbf{D}}$ (peut-être négatif) et $v^{(j)} \in \mathbf{V}$, est satisfiable ssi il existe une fonction δ associant des variables dans c aux valeurs dans $\Delta_{\mathbf{D}}$ pour que $\delta(v^{(j)}) \in d_j^{\mathbf{D}}$ pour $1 \leq j \leq k$. Cette fonction δ est appelée une solution pour c . Le négatif $\neg d$ est interprété comme $\Delta_{\mathbf{D}} \setminus d^{\mathbf{D}}$.

Le domaine concret universel est *admissible* ssi le problème de satisfiabilité pour des conjonctions finies de type de données (peut-être négatifs) sur $\Phi_{\mathbf{D}}$ est décidable.

2.2. Syntaxe et sémantique

Supposons que le domaine d'interprétation du domaine concret $\Delta_{\mathbf{D}}$ est disjoint duquel du domaine abstrait $\Delta^{\mathcal{I}}$ et \mathbf{D} est admissible.

Définition 2.2 (L'hierarchie de rôles) Soit $\mathbf{R} = \mathbf{R}_A \cup \mathbf{R}_D$ l'ensemble des noms des rôles abstraits et concrets.

L'hierarchie de rôles (ou role box) \mathcal{R} est un ensemble fini d'axiomes d'inclusion de rôles sous forme $r \sqsubseteq s$ avec $r, s \in \mathbf{R}_A$ ou $r, s \in \mathbf{R}_D$.

$\mathbf{R}_A = \mathbf{R}_{tp} \cup \{r^- \mid r \in \mathbf{R}_{tp}\}$ où \mathbf{R}_{tp} est l'ensemble des noms des rôles transitifs \mathbf{R}_+ et des noms des rôles abstraits primitifs \mathbf{R}_{Ap} , r^- est le rôle inverse de r .

La fonction Inv est défini comme $\text{Inv}(r) = r^-$ et $\text{Inv}(r^-) = r$ où $r \in \mathbf{R}_{tp}$.

La fonction $\text{Trans}(s) = \text{vrai}$ s'il existe r avec $r \equiv_{\mathcal{R}} s$, $r \in \mathbf{R}_+$ ou $\text{Inv}(r) \in \mathbf{R}_+$.
 $\text{Trans}(s) = \text{faux}$ autrement.

r est un rôle simple si $\text{Trans}(s) = \text{faux}$ pour tout $s \sqsubseteq_{\mathcal{R}} r$.

L'équivalence de 2 rôles $r \equiv_{\mathcal{R}} s$ signifie que $r \sqsubseteq_{\mathcal{R}} s$ et $s \sqsubseteq_{\mathcal{R}} r$ où $\sqsubseteq_{\mathcal{R}}$ est une fermeture réflexive-transitive de \sqsubseteq sur \mathcal{R} .

L'interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ se compose du domaine d'interprétation $\Delta^{\mathcal{I}}$ et une fonction $\cdot^{\mathcal{I}}$ qui associe chaque rôle à un sous-ensemble de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ tel que $\forall p \in \mathbf{R}$ et $r \in \mathbf{R}_+$,

$$\langle x, y \rangle \in p^{\mathcal{I}} \text{ ssi } \langle y, x \rangle \in (p^-)^{\mathcal{I}},$$

$$\text{si } \langle x, y \rangle \in r^{\mathcal{I}} \text{ et } \langle y, z \rangle \in r^{\mathcal{I}} \text{ alors } \langle x, z \rangle \in r^{\mathcal{I}}.$$

L'interprétation \mathcal{I} satisfait l'hierarchie de rôles \mathcal{R} ssi $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ pour tout $r \sqsubseteq s \in \mathcal{R}$. Une telle interprétation est appelée un modèle de \mathcal{R} .

Les constructeurs de rôle (cf. le tableau) :

Nom de constructeur	Syntaxe	Sémantique
rôle abstrait atomique	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
rôle concret	u	$u^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}^{\mathcal{I}}$
rôle transitif	$r \in \mathbf{R}_+$	$r^{\mathcal{I}} = r^{\mathcal{I}+}$
rôle inverse	r^-	$\{\langle x, y \rangle \mid \langle y, x \rangle \in r^{\mathcal{I}}\}$
hiérarchie de rôles	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

Définition 2.3 (TBox) Une TBox \mathcal{T} de $\mathcal{SHOIN}(\mathbf{D})$ se compose de concepts primitifs et non-primitifs. Soient N_C l'ensemble des noms de concept primitif et N_I l'ensemble des noms de concept nominal alors $N_I \subseteq N_C$. Des concepts non-primitifs (ou simplement concepts) sont sous forme d'axiome d'inclusion (ou encore appelé GCI- general concept inclusion) $C_1 \sqsubseteq C_2$, ou introduit par la définition de concept $C_1 \equiv C_2$ où C_1 et C_2 sont des concepts (peut-être complexes). La description $C_1 \equiv C_2$ est en effet l'abréviation de 2 axiomes d'inclusion $C_1 \sqsubseteq C_2$ et $C_2 \sqsubseteq C_1$.

Des constructeurs de concept (cf. le tableau) :

Nom de constructeur	Syntaxe	Sémantique
concept atomique	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
type de données	D	$D_{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}^{\mathcal{I}}$
valeur concrète	v	$v^{\mathcal{I}} = v^{\mathbf{D}}$
négation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjonction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjonction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
quantification existentielle	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restriction universelle	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
quantification concrète existentielle	$\exists R.d$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\}$
restriction concrète universelle	$\forall R.d$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in d^{\mathbf{D}}\}$
restriction de cardinalité inférieure	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \geq n\}$
restriction de cardinalité supérieure	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y. \langle x, y \rangle \in R^{\mathcal{I}}\} \leq n\}$
nominal	o	$\#\{o^{\mathcal{I}}\} = 1$

Une interprétation \mathcal{I} satisfait une GCI $C_1 \sqsubseteq C_2$ si $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. \mathcal{I} satisfait une TBox \mathcal{T} si elle satisfait toutes les GCIs dans \mathcal{T} . Une telle interprétation est appelée un modèle de \mathcal{T} .

Un concept C est satisfiable pour une hiérarchie de rôles \mathcal{R} et un TBox \mathcal{T} ssi il existe une interprétation \mathcal{I} de \mathcal{R} et de \mathcal{T} telle que $C^{\mathcal{I}} \neq \emptyset$. Une telle interprétation est appelée un modèle de C pour une \mathcal{R} et \mathcal{T} .

Un concept D subsume un concept C pour \mathcal{R} et \mathcal{T} ($C \sqsubseteq_{\mathcal{R}, \mathcal{T}} D$) si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ dans tout modèle \mathcal{I} de \mathcal{R} et de \mathcal{T} . Deux concepts C et D sont équivalents pour une \mathcal{R} et \mathcal{T} s'ils subsument mutuellement pour \mathcal{R} et \mathcal{T} .

Comme la subsumption peut être réduite à la satisfiabilité, on considère seulement la satisfiabilité de concept pour \mathcal{R} .

2.3. Influences du domaine concret

Etant un membre de la famille de LDs, une des propriétés de $\mathcal{SHOIN}(\mathbf{D})$ est d'avoir les modèles sous forme d'arbre. Plus précisément, l'algorithme du Tableau décide la cohérence d'un concept/Tbox de $\mathcal{SHOIN}(\mathbf{D})$ en essayant de construire une abstraction de modèle appelée un graphe d'accomplissement qui est sous forme d'un graphe arboré. Dans ce graphe, les noeuds représentant les valeurs concrètes sont appelés les *noeuds concrets*, ceux représentant les individus sont appelés les *noeuds abstraits* qui comprennent les *noeuds nominaux* (représentant les nominaux) et les *noeuds blocables*. L'ensemble de noeuds nominaux est disjoint de celui de blocables. Nous appelons deux noeuds *égaux* ssi ils représentent le même élément (c.à.d un individu d'un concept ou une valeur concrète d'un type de données).

Influence sur la restriction de cardinalité. Comme pour $\mathcal{SHOQ}(\mathbf{D})$, le domaine concret \mathbf{D} dans $\mathcal{SHOIN}(\mathbf{D})$ est présumé admissible. Par contre, cette supposition n'est pas suffisante avec la présence de la restriction concrète de cardinalité. Considérons l'exemple suivant : soit le domaine concret $N = (\mathbb{N}, \{<_2\})$, nous vérifions

la satisfiabilité du concept $\geq 3U \sqcap \forall U. <_2$. Construisons le graphe pour ce concept. Nous voyons qu'un noeud x doit avoir la relation U avec au moins 3 noeuds concrets. Spécifiés par la contrainte de cardinalité, ces trois noeuds représentent trois variables concrètes différentes. Définies par le domaine concret, les variables sont différentes si et seulement si la solution δ les associe aux valeurs différentes (remarquez que dans notre approche, l'espace lexicale des valeurs n'est pas considérée, c'est pourquoi nous supposons que deux valeurs de 2 types de données différents sont différentes). Avec la contrainte de restriction universelle pour le type de données $<_2$, ces noeuds ont pour valeur 0 ou 1 (le prédicat $<_2$ définit un ensemble de valeurs de type *integer* à inférieure à 2). Alors il y a au moins 2 noeuds qui ont la même valeur. Ceci est en conflit avec la contrainte de cardinalité précédente. Le raisonnement concret doit alors vérifier la satisfiabilité des variables restreintes par la contrainte de cardinalité concrète. Précisément, soit \neq le symbole représentant l'inégalité entre des noeuds dans le graph, la conjonction suivante doit être vérifiée :

$$c' = c \wedge \bigwedge_{v \neq v'} \neq(v, v') \quad [2]$$

Influence sur la procédure de raisonnement. Comme présenté ci-dessus, la restriction concrète de cardinalité peut causer l'insatisfiabilité d'un concept. En construisant un graphe G , chaque fois que la restriction de cardinalité génère des nouveaux noeuds concrets, un conflit se produit éventuellement. Le raisonnement concret ne peut pas alors être exécuté une fois que toutes les règles d'extension ont été appliquées exhaustivement contrairement à $SHOQ(\mathbf{D})$. L'algorithme de vérification de la satisfiabilité de concept doit être changé en conséquence comme suit :

```

Function Sat( $G$ )
  If  $G$  contient clash then return Insatisfait
  elseif il y a des nouveaux noeuds concrets then Verifier  $c'$ 
  If  $G$  contient clash then return Insatisfait
   $G'$  = application d'une règle d'extension sur  $G$ 
  Return Sat( $G'$ )

```

3. Contrainte d'identification et $SHOINK(\mathbf{D})$

Dans cette section, nous introduisons la formalisation de la contrainte d'identification sur $SHOIN(\mathbf{D})$. Le langage résultant est appelé $SHOINK(\mathbf{D})$. Ajoutant cette contrainte à $SHOIN(\mathbf{D})$ surgit quelques problèmes que nous montrerons au-dessous. Une solution est ainsi proposée dans cette section.

3.1. Définition

Définition 3.1 (Contrainte d'identification) *La contrainte d'identification (CI) est définie dans la formule suivante. Le LD étendu, appelé $SHOINK(\mathbf{D})$, permet*

d'exprimer cette contrainte par un nouveau constructeur **Idfor**. L'ensemble des contraintes crée une nouvelle boîte dans $\mathcal{SHOIN}\mathcal{K}(\mathbf{D})$ appelée KBox \mathcal{K} . La contrainte en utilisant le nouveau constructeur s'écrit :

$$(R_1, \dots, R_n \mathbf{Idfor} C)$$

où C est un concept, R_i est un rôle simple (abstrait ou concret) $\forall 1 \leq i \leq n$.

La sémantique de cette définition est formellement représentée par la définition de l'interprétation suivante : Une interprétation \mathcal{I} satisfait la définition $(R_1, \dots, R_n \mathbf{Idfor} C)$ ssi $\forall s, s' \in C^{\mathcal{I}}$ et $\forall \langle s, t_i \rangle, \langle s', t'_i \rangle \in R_i^{\mathcal{I}}, \forall 1 \leq i \leq n$, on a $t_i = t'_i \forall 1 \leq i \leq n$ alors $s = s'$.

Les rôles dans la définition doivent être simples pour assurer la décidabilité de l'algorithme du Tableau. Intuitivement, cette définition indique que deux instances de C n'ont jamais la même participation dans ces n rôles.

Dans le graphe construit par l'algorithme du Tableau, les noeuds qui identifient un élément d'un concept satisfiant la CI représentent soit des individus soit des valeurs concrètes. On appelle ces noeuds les *noeuds identifiants*. Le noeud représentant l'élément du concept satisfiant la contrainte d'unicité est appelé le *noeud identifié*.

La satisfiabilité de concept doit considérer aussi la présence éventuelle de Kbox.

3.2. CI et le problème avec le domaine concret

L'ajout de CI au $\mathcal{SHOIN}(\mathbf{D})$ fait surgir le même problème avec le domaine concret comme ce qui est montré par C.Lutz et al. pour $\mathcal{ALCOK}(\mathcal{D})$ dans (Lutz et al., 2003). Précisément, le problème est illustré par l'exemple suivant :

$$\exists R.A \sqcap \exists R.(\neg A \sqcap B) \sqcap \exists R.(\neg A \sqcap \neg B) \sqcap \forall R.\exists U. <_2$$

Construisant le graphe pour ce concept, on a trois feuilles qui ont des valeurs satisfiant le prédicat $<_2$. Parce que les valeurs possibles pour les trois feuilles sont 0 et 1 alors pour toute solution δ il y a au moins 2 feuilles qui ont la même valeur. Dans ce cas là, ces deux noeuds concrets sont égaux.

Si ce concept est vérifié pour une Kbox qui contient une définition $(U \mathbf{Idfor} \top)$, alors cette CI demande qu'au moins deux pères de ces feuilles doivent représenter le même individu. Par conséquent, le concept devient insatisfait.

Si le raisonnement concret ne fournit pas des égalités des variables (c.à.d les informations sur les variables qui sont associées à la même valeur dans les solutions), le moteur d'inférence de LD peut ne pas décider éventuellement la satisfiabilité de concept. Pour cette raison, nous avons besoin que le raisonnement concret retourne

ces informations. Remarquez que dans l'exemple au-dessus, aucune solution δ ne peut être trouvée tel que l'inégalité des variables soit satisfaite. Ceci avec le besoin de vérification 2 montré dans la section 2, nous construisons alors un algorithme qui cherche de plusieurs solutions la meilleure δ dans le sens où les variables dans c sont associées différemment autant que possible. Les égalités des variables dans cette solution seront ensuite passées au moteur d'inférence de LD. Nous réalisons cette idée en utilisant la conjonction suivante :

$$c_{\mathcal{K}} = c \wedge \bigwedge_{\forall (v, v') \in V_{ie}} \neq (v, v') \quad [3]$$

où V_{ie} est l'ensemble des variables qui ne sont pas *égales* dans V , l'ensemble des variables dans c . Deux variables v et v' sont appelées *égales* ssi $\delta(v) = \delta(v')$. Nous cherchons le plus grand ensemble V_{ie} tel qu'il existe une solution δ satisfaisant $c_{\mathcal{K}}$.

Avec l'idée de $c_{\mathcal{K}}$, nous construisons alors une fonction *EqualTest* qui devine le plus grand V_{ie} , cherche la solution δ pour $c_{\mathcal{K}}$ et retourne l'ensemble des variables égales. Si δ est trouvé, V_{ie} deviné est vraiment le plus grand ensemble des variables différentes dans c . En conséquence, le complément V_e de V_{ie} par rapport à V est l'ensemble des variables égales que nous avons besoins. Soit \doteq le symbole représentant l'égalité entre les noeuds v et v' , $v \doteq v'$ si et seulement si (v, v') est dans l'ensemble V_e retourné par *EqualTest*. En introduisant $c_{\mathcal{K}}$, nous n'avons plus besoin de vérifier c' dans la formule (2) et nous pouvons décider la cohérence du concept.

Fonction EqualTest(c)

$$V_c = \{(v, v') | \forall v, v' \in c\}$$

$$V_e = \{\emptyset\}$$

$$V_{ie} = V_c$$

Do

$$c_{\mathcal{K}} = c \wedge \bigwedge_{\forall (v, v') \in V_{ie}} \neq (v, v')$$

Chercher la solution δ pour $c_{\mathcal{K}}$

Si δ trouvé **alors Retourne** V_e

sinon

$$\text{chercher le maximal } V_{ie} = \{(v, v') | v, v' \in c\}$$

$$V_e = V_c - V_{ie}$$

endif

While $V_{ie} \neq \{\emptyset\}$

Retourne V_e

Ainsi au lieu de vérifier c' , la procédure de raisonnement pour $SHOIN\mathcal{K}(\mathbf{D})$ appelle la fonction $EqualTest(c)$.

A cause de la similarité des noeuds nominaux et de noeuds concrets dans le graphe, deux noeuds concrets qui représentent la même valeur doivent être *fusionnés* par un mécanisme appelé *fusionnement*. C'est une technique qui permet de remplacer deux noeuds représentant un même élément par un noeud dans le graphe. De plus, les noeuds concrets ne peuvent qu'être les feuilles des arbres. La fusion de ces noeuds alors ne casse pas la structure d'arbre des noeuds abstraits. Pour ces raisons, sans perdre la propriété de structure d'arbre du graphe, nous introduisons la v -règle. Deux noeuds concrets sont fusionnés ssi ils ne sont pas distingués et la solution δ retourne la même valeur pour ces noeuds.

3.3. CI et le problème avec la procédure de raisonnement pour $SHOIN\mathcal{K}(\mathbf{D})$

Une autre difficulté à l'ajout de CI à $SHOIN(\mathbf{D})$ est que l'interaction entre les restrictions de cardinalité, les rôles inverses et cette contrainte peut causer l'indécidabilité. L'essentiel du problème est que l'algorithme du Tableau pour ce langage expressif utilise une technique appelée *blocage* pour détecter des cycles afin d'assurer la terminaison. Cette technique est appliquée sur seulement des noeuds blocables dont deux noeuds sont les *ancêtres* d'autre. Au cas où les cycles existeraient, l'algorithme du Tableau génère un graphe fini qui peut être étendu par *effilochage* (*unravelled* en anglais) pour représenter un modèle infini. Dans ce graphe, il y a des noeuds qui représentent un nombre infini d'éléments du modèle. Par conséquent, des problèmes surgissent quand la CI est appliquée sur ces noeuds. Pour voir ces problèmes, supposons que dans le graphe on a 2 noeuds s, s' représentant des instances de C qui satisfont la contrainte $(R_1, R_2 \text{ Idfor } C)$ et les noeuds t_1, t'_1, t_2, t'_2 tels que $\langle s, t_1 \rangle, \langle s', t'_1 \rangle \in R_1^{\mathcal{I}}$ et $\langle s, t_2 \rangle, \langle s', t'_2 \rangle \in R_2^{\mathcal{I}}$. Par la contrainte on peut dire que (t_1, t_2) identifie s et (t'_1, t'_2) identifie s' . Si t_1 est égal à t'_1 et t_2 est égal à t'_2 , alors par l'interprétation, s doit être égal à s' . Dans le graphe, le label d'un noeud x est appelé $\mathcal{L}(x)$, le label d'une arête de noeud x à y est appelée $\mathcal{L}(\langle x, y \rangle)$. Par conséquent, on a $\mathcal{L}(s) = \mathcal{L}(s')$, $\mathcal{L}(\langle s, t_1 \rangle) = \mathcal{L}(\langle s', t'_1 \rangle)$ et $\mathcal{L}(t_1) = \mathcal{L}(t'_1)$.

- Si s, t_1 et s' sont les ancêtres de t'_1 alors la satisfaction de la CI entraîne celle de la condition de blocage. En conséquence, s' et s sont considérés distingués en effilochage. Il provoque alors un conflit.

- Si les noeuds égaux sont fusionnés, alors la structure d'arbre du graphe peut être cassée. Cela peut causer l'indécidabilité.

- Si s est dans le cycle de blocage et R_1 et R_2 sont des rôles concrets, alors l'action d'effilochage construit un nombre infini d'individus de C qui ont les mêmes relations que s . Supposons que le moteur d'inférence de LD est fournit l'égalité de deux noeuds concrets v_1 et v'_1 et l'égalité de noeuds concrets v_2 et v'_2 tels que $\langle x, v_1 \rangle, \langle y, v'_1 \rangle \in R_1^{\mathcal{I}}$, $\langle x, v_2 \rangle, \langle y, v'_2 \rangle \in R_2^{\mathcal{I}}$ où $x, y \in C^{\mathcal{I}}$ alors x et y en conformité avec la CI doivent être égaux. Cependant, l'effilochage peut causer un nombre infini des noeuds concrets.

Le raisonnement concret en conséquence ne peut pas contrôler l'égalité des valeurs concrètes en effilochage (les individus sont toujours supposés différents dans ce processus). Cela alors peut entraîner l'indécidabilité.

L'algorithme du Tableau doit donc avoir à la fois la capacité d'assurer la satisfaction de toutes les contraintes, dont la CI, et de traiter l'interaction entre les structures relationnelles complexes et les structures d'arbre finies représentant le modèle infini et ces contraintes, en garantissant la terminaison.

Pour répondre à ces besoins, nous nous basons sur les propriétés décisives comme suit :

Lemme 3.1 *Dans le graphe construit par l'algorithme du Tableau pour un $SHOINK(\mathbf{D})$ -concept, un noeud est l'arrivé de plus d'une arête seulement s'il est un noeud nominal ou concret.*

Preuve. Sans perdre en généralité, nous supposons qu'il existe un noeud blocable s qui a deux arêtes venant de x et y . Une arête va à un noeud existant ssi le fusionnement est appliqué. Supposons que s est le résultat du fusionnement de deux noeuds blocables s et s' (qui sont l'enfant de x et y respectivement). Basé sur l'algorithme du Tableau pour $SHOIQ$, les règles pour développer le graphe qui provoquent la fusion incluent v -, o - et \leq -règles. v -règle comme expliqué au-dessus s'applique seulement aux noeuds concrets. o -règle s'applique seulement aux noeuds nominaux. Ces deux règles alors ne s'appliquent pas à s et s' . Par la définition de la \leq -règle, s et s' doivent avoir le même parent. Alors il n'y a qu'une seule arête venant s . Cela est contradictoire à l'hypothèse.

■

De plus, une propriété de la CI est que à chaque élément identifié correspond un ensemble unique d'éléments identifiants par n relations où n est le nombre d'éléments identifiants. Donc s'il y a dans le graphe un noeud identifié par n autre noeuds, aucun autre noeud dans le graphe a les mêmes connections. S'il existe un tel noeud, ces deux noeuds doivent être égaux (c.à.d ces deux noeuds sont la représentation de même individu). Ces noeuds doivent alors être fusionnés pour conserver l'unicité d'individu dans le graphe et la correction de l'algorithme. Ainsi la condition de blocage ne peut pas être appliquée.

D'ailleurs, représenter les noeuds identifiés par les nominaux empêche la fusion de casser la structure d'arbre. Donc si on a deux noeuds identifiés blocables pour une fusion tels qu'ils n'ont pas le même père et que tous leurs noeuds identifiants sont leurs enfants, alors un nouveau nominal est ajouté aux labels de ces deux noeuds. Sinon un noeud doit être l'ancêtre de l'autre. Dans ce cas, le descendant est fusionné à l'ancêtre. Cet idée est effectuée par la \mathcal{K} -règle. Nous montrons que cette fusion ne casse pas la structure d'arbre :

Lemme 3.2 Soient s et s' les noeuds blocables satisfaisant une CI dans un graphe. La fusion de s et s' ne casse pas la structure d'arbre avec la condition de fusionnement comme suit :

- Si tous les noeuds identifiants de s et s' sont leurs enfants et s et s' n'ont pas le même père alors ajouter un nouveau nominal $o_{\mathcal{K}}$ à $\mathcal{L}(s)$ et à $\mathcal{L}(s')$.
- Sinon faire la fusion du descendant à son ancêtre.

Preuve. Au cas où un nouveau nominal serait créé, s et s' seront fusionnés par la o -règle et s et s' deviendront un noeud nominal. Si s et s' sont fusionnés sans créer un nominal, par la condition de fusionnement alors, soit s et s' ont le même père, soit au moins un des noeuds identifiants x n'est pas un enfant de s ou de s' . Si s et s' ont le même père alors la fusion de s et s' ne conserve pas les deux pères. Si s et s' n'ont pas le même père, alors sans perdre en généralité, nous supposons que x est le père de s' . S'il existe un autre père de s' , par le lemme 3.1 s' doit être un noeud nominal. Ceci est contraire à l'hypothèse. Ainsi s et x doivent être les ancêtres de s' . Ceci nous ramène à la deuxième condition de fusion. Par cette fusion, s' et s ont le même père. Parce que le noeud blocable s a seulement un père (par le lemme 3.1), la fusion de s et s' ne crée pas un noeud blocable à deux pères. ■

En outre, le nombre d'éléments d'un concept est naturellement borné par le domaine du concept. Donc pour des noeuds identifiés dans un cycle de blocage, nous supposons que leurs noeuds identifiants concrets ont des valeurs différentes. Ainsi le cycle est terminé quand les domaines concrets pour les noeuds identifiants sont exhaustivement exploités. Remarquez que pour un domaine concret infini d'un type de données, δ trouve toujours des valeurs différentes pour des variables dans le graphe. En conséquence avec de tels domaines il n'existe jamais deux noeuds identifiés par le même ensemble de noeuds concrets. Cette hypothèse aide alors à éviter l'indécidabilité comme présentée au-dessus. Elle est aussi importante afin d'assurer la terminaison de l'algorithme.

4. L'algorithme du Tableau pour $SHOINK(\mathbf{D})$

4.1. Tableau pour $SHOINK(\mathbf{D})$

Supposons que tous les concepts dans Tbox et Kbox sont sous forme négation normale (NNF) (Horrocks *et al.*, 2001). $\neg C$ est la forme NNF du $\neg C$. Pour un concept D dans $SHOINK(\mathbf{D})$, $\text{sub}(D)$ est l'ensemble de tous les sous-concepts de D (D y compris) et est défini formellement comme suit :

$$\begin{aligned} \text{sub}(A) &= \{A\} \text{ avec } A \text{ est un concept primitif,} \\ \text{sub}(C \sqcap D) &= \{C \sqcap D\} \cup \text{sub}(C) \cup \text{sub}(D), \end{aligned}$$

$$\begin{aligned}
\text{sub}(C \sqcup D) &= \{C \sqcup D\} \cup \text{sub}(C) \cup \text{sub}(D), \\
\text{sub}(\forall R.C) &= \{\forall R.C\} \cup \text{sub}(C), \\
\text{sub}(\exists R.C) &= \{\exists R.C\} \cup \text{sub}(C), \\
\text{sub}(\leq nR) &= \{\leq nR\}, \\
\text{sub}(\geq nR) &= \{\geq nR\}, \\
\text{sub}(\leq nU) &= \{\leq nU\}, \\
\text{sub}(\geq nU) &= \{\geq nU\}, \\
\text{sub}(\forall U.d) &= \{\forall U.d\}, \\
\text{sub}(\exists U.d) &= \{\exists U.d\}
\end{aligned}$$

Soit $\text{Con}(\mathcal{K})$ l'ensemble des noms de concept dans KBox. Nous définissons $cl(D, \mathcal{K}, \mathcal{R})$ comme suit :

$$cl(D, \mathcal{K}, \mathcal{R}) = cl(D, \mathcal{K}) \cup \{\forall R.C \mid R \boxeq S, \forall S.C \in cl(D, \mathcal{K}) \text{ and } R \in \mathcal{R}\}$$

où

$$cl(D, \mathcal{K}) = \text{sub}(D) \cup \text{sub}(\text{Con}(\mathcal{K})) \cup \{\neg C \mid C \in \{\text{sub}(D) \cup \text{sub}(\text{Con}(\mathcal{K}))\}\}$$

et

$$\text{sub}(\text{Con}(\mathcal{K})) = \bigcup_{C \in \text{Con}(\mathcal{K})} \text{sub}(C).$$

Nous utilisons $cl(D)$ au lieu de $cl(D, \mathcal{K}, \mathcal{R})$ pour faire plus court.

Définition 4.1 (Tableau) Soient D un concept de $\text{SHOIN}(\mathbf{D})$ sous forme NNF, \mathcal{R} l'hierarchie de rôle, \mathcal{K} Kbox dans NNF, $\mathbf{R}_A^D, \mathbf{R}_D^D$ les ensembles de rôles abstraits et concrets dans D, \mathcal{K} ou \mathcal{R} . Un tableau T pour D pour une \mathcal{R} et une \mathcal{K} est défini comme un tuple $(\mathbf{S}_A, \mathbf{S}_D, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_D)$ tel que :

- \mathbf{S}_A est l'ensemble d'individus ;
 - \mathbf{S}_D est l'ensemble de valeurs concrètes ;
 - $\mathcal{L} : \mathbf{S}_A \rightarrow 2^{cl(D)}$ associe chaque individu à un ensemble de concepts qui est un sous-ensemble de $cl(D)$;
 - $\mathcal{E}_A : \mathbf{R}_A^D \rightarrow 2^{\mathbf{S}_A \times \mathbf{S}_A}$ associe chaque rôle abstrait à un ensemble de paires des individus ;
 - $\mathcal{E}_D : \mathbf{R}_D^D \rightarrow 2^{\mathbf{S}_A \times \mathbf{S}_D}$ associe chaque rôle concret dans \mathbf{R}_D^D à un ensemble de paires d'individu et de valeur concrète.
 - Il existe un individu $s_0 \in \mathbf{S}_A$ tel que $D \in \mathcal{L}(s_0)$ et
- $$\forall C, C_1, C_2 \in cl(D), R, S \in \mathbf{R}_A^D, U, U' \in \mathbf{R}_D^D, s, t \in \mathbf{S}_A \text{ ou } \mathbf{S}_D, \text{ on a}$$

(P1) si $C \in \mathcal{L}(s)$, alors $\neg C \notin \mathcal{L}(s)$,

(P2) si $C_1 \sqcap C_2 \in \mathcal{L}(s)$, alors $C_1 \in \mathcal{L}(s)$ et $C_2 \in \mathcal{L}(s)$,

(P3) si $C_1 \sqcup C_2 \in \mathcal{L}(s)$, alors $C_1 \in \mathcal{L}(s)$ ou $C_2 \in \mathcal{L}(s)$,

- (P4) si $\langle s, t \rangle \in \mathcal{E}_A(R)$ et $R \boxplus S$ alors $\langle s, t \rangle \in \mathcal{E}_A(S)$,
- (P5) si $\forall R.C \in \mathcal{L}(s)$ et $\langle s, t \rangle \in \mathcal{E}_A(R)$ alors $C \in \mathcal{L}(t)$,
- (P6) si $\exists R.C \in \mathcal{L}(s)$ alors il y a un certain $t \in \mathbf{S}_A$ tel que $\langle s, t \rangle \in \mathcal{E}_A(R)$ et $C \in \mathcal{L}(t)$,
- (P7) si $\forall S.C \in \mathcal{L}(s)$ et $\langle s, t \rangle \in \mathcal{E}_A(R)$ pour $R \boxplus S$ avec $\text{Trans}(R)$ alors $\forall R.C \in \mathcal{L}(t)$,
- (P8) $\langle s, t \rangle \in \mathcal{E}_A(R)$ ssi $\langle t, s \rangle \in \mathcal{E}_A(\text{Inv}(R))$
- (P9) si $\geq nS \in \mathcal{L}(s)$ alors $\#S^T(s) \geq n$,
- (P10) si $\leq nS \in \mathcal{L}(s)$ alors $\#S^T(s) \leq n$,
- (P11) si $o \in \mathcal{L}(s) \cap \mathcal{L}(t)$ alors $s = t$,
- (Pd12) si $\geq nU \in \mathcal{L}(s)$ alors $\#U^T(s) \geq n$,
- (Pd13) si $\leq nU \in \mathcal{L}(s)$ alors $\#U^T(s) \leq n$,
- (Pd14) si $\exists U.d \in \mathcal{L}(s)$ alors il y a un certain $t \in \mathbf{S}_D$ tel que $\langle s, t \rangle \in \mathcal{E}_D(U)$ et $t \in d^{\mathbf{D}}$,
- (Pd15) si $\forall U.d \in \mathcal{L}(s)$ et $\langle s, t \rangle \in \mathcal{E}_D(U)$ alors $t \in d^{\mathbf{D}}$,
- (Pd16) si $\langle s, t \rangle \in \mathcal{E}_D(U)$ et $U \boxplus U'$ alors $\langle s, t \rangle \in \mathcal{E}_D(U')$,
- (Pu17) Si $(R_1, \dots, R_n \text{ Ifor } C) \in \mathcal{K}$ et $\langle s, t_i \rangle \in \mathcal{E}_A(R_i)$ ou $\in \mathcal{E}_D(R_i) \forall 1 \leq i \leq n$ alors $\{C, \neg C\} \cap \mathcal{L}(s) \neq \emptyset$
- (Pu18) Si $(R_1, \dots, R_n \text{ Idfor } C) \in \mathcal{K}$, $C \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$, $\langle s, t_i \rangle$ et $\langle s', t'_i \rangle \in \mathcal{E}_A(R_i)$ ou $\mathcal{E}_D(R_i)$ et $t_i = t'_i \forall 1 \leq i \leq n$ alors $s = s'$ et $(\leq 1R_i) \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$

avec

$$S^T(s) := \{t \in \mathbf{S}_A \mid \langle s, t \rangle \in \mathcal{E}_A(S)\},$$

$$U^T(s) := \{t \in \mathbf{S}_D \mid \langle s, t \rangle \in \mathcal{E}_D(U)\}.$$

Lemme 4.1 *Un concept D dans $\mathcal{SHOINK}(\mathbf{D})$ sous forme NNF est satisfiable pour une hiérarchie de rôles \mathcal{R} et Kbox \mathcal{K} ssi D a un tableau pour une \mathcal{R} et une \mathcal{K} .*

Preuve. Nous nous concentrons sur les nouvelles caractéristiques par rapport à $\mathcal{SHOQ}(\mathbf{D})$, i.e les restrictions de cardinalité pour les types de données et la contrainte d'identification. Le reste est comparable à [(Horrocks *et al.*, 2001), (Horrocks *et al.*, 2005), (Horrocks *et al.*, 1999)]. Généralement, le modèle est construit à partir du tableau donné en prenant \mathbf{S}_A comme domaine d'interprétation de D et en ajoutant des relations de rôle-successeurs pour des rôles transitifs. Ensuite, par l'induction sur la structure de formule, nous prouvons que si $D \in \mathcal{L}(s)$ alors $s \in D^{\mathcal{I}}$. La propriété 11 assure que le nominal est interprété comme un singleton. Les propriétés 12 et 13 assurent l'interprétation correcte pour les restrictions concrètes de cardinalité. La propriétés 17 et 18 assurent que les CIs sont interprétées correctement. Réciproquement, chaque modèle est un tableau par définition de la sémantique.

Pour la direction "si", soit $T = (\mathbf{S}_A, \mathbf{S}_D, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_D)$ le tableau pour D en vue de (e.v.d) \mathcal{R} et \mathcal{K} . Nous construisons le modèle \mathcal{I} pour D e.v.d \mathcal{R} et \mathcal{K} comme suit :

$$\begin{aligned}
\Delta^{\mathcal{I}} &:= \mathbf{S}_A, \\
A^{\mathcal{I}} &:= \{s \in \mathbf{S}_A \mid A \in \mathcal{L}(s)\} \text{ pour tout nom de concept } A \text{ dans } cl(D), \\
R^{\mathcal{I}} &:= \begin{cases} \mathcal{E}_A(R)^+ & \text{si } \text{Trans}(R) \\ \mathcal{E}_A(R) \cup \bigcup_{S \boxplus R, S \neq R} S^{\mathcal{I}} & \text{autrement} \end{cases} \\
U^{\mathcal{I}} &:= \mathcal{E}_{\mathbf{D}}(U) \\
&\text{où } \mathcal{E}_A(R)^+ \text{ est la fermeture transitive de } \mathcal{E}_A(R)
\end{aligned}$$

Pour montrer que \mathcal{I} est le modèle de D e.v.d \mathcal{R} et \mathcal{K} , on doit prouver que (1) $\mathcal{I} \models \mathcal{R}$, (2) $D^{\mathcal{I}} \neq \emptyset$ et (3) $\mathcal{I} \models \mathcal{K}$.

La définition de $R^{\mathcal{I}}$ montre que si $(x, y) \in R^{\mathcal{I}}$ alors soit $(x, y) \in \mathcal{E}_A(R)^+$ au cas où R est transitif, soit $(x, y) \in \mathcal{E}_A(R) \cup \bigcup_{S \boxplus R, S \neq R} S^{\mathcal{I}}$ pour interpréter correctement des rôles non-transitifs qui peuvent avoir des sous-rôles transitifs. La propriété 4 et 16 du tableau assurent que l'hierarchie de rôles est interprétée correctement. L'interprétation des rôles inverses est satisfaite par la propriété 8 du tableau. Par la définition de \mathcal{E}_A et $\mathcal{E}_{\mathbf{D}}$, $\mathcal{I} \models \mathcal{R}$.

$D^{\mathcal{I}} \neq \emptyset$ est montré par l'induction sur la structure de D . Autrement dit, on va montrer que pour tout concept $E \in cl(D)$, $E \in \mathcal{L}(s)$ implique $s \in E^{\mathcal{I}}$.

- 1) Si $E = A$ alors par la définition $s \in E^{\mathcal{I}}$.
- 2) Si $E = \neg A$ alors $A \notin \mathcal{L}(s)$ par la propriété 1 de la définition 4.1. Donc $s \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} = E^{\mathcal{I}}$.
- 3) Si $E = (C_1 \sqcap C_2)$ alors par la propriété 2 $C_1 \in \mathcal{L}(s)$ et $C_2 \in \mathcal{L}(s)$. Par l'induction on a $s \in C_1^{\mathcal{I}}$ et $s \in C_2^{\mathcal{I}}$ qui implique $s \in (C_1 \sqcap C_2)^{\mathcal{I}}$.
- 4) $E = (C_1 \sqcup C_2)$ est prouvé comme le dernier cas avec la propriété 3.
- 5) $E = \exists R.C$. $E \in \mathcal{L}(s)$ implique par la propriété 6 l'existence d'un individu $t \in \mathbf{S}_A$ tel que $\langle s, t \rangle \in \mathcal{E}_A(S)$ et $C \in \mathcal{L}(t)$. Par l'induction on a $t \in C^{\mathcal{I}}$, par la définition de $R^{\mathcal{I}}$ on a $\langle s, t \rangle \in R^{\mathcal{I}}$. Par conséquence, $s \in (\exists R.C)^{\mathcal{I}}$.
- 6) $E = \forall R.C$. Soit $t \in \mathbf{S}_A$ un individu arbitraire tel que $\langle s, t \rangle \in R^{\mathcal{I}}$. Par la définition,

- soit $\langle s, t \rangle \in \mathcal{E}_A(R)$. Alors par la propriété 5 $C \in \mathcal{L}(t)$.

- soit $\langle s, t \rangle \notin \mathcal{E}_A(R)$. Alors il existe un chemin $\langle s, s_1 \rangle, \langle s_1, s_2 \rangle, \dots, \langle s_n, t \rangle \in \mathcal{E}_A(S)$ avec $\text{Trans}(S)$ et $S \boxplus R$. Par la propriété 7, $\forall S.C \in \mathcal{L}(s_i) \forall 1 \leq i \leq n$. Par propriété 5, $C \in \mathcal{L}(t)$.

En deux cas, par induction on a $t \in C^{\mathcal{I}}$ alors $s \in E^{\mathcal{I}}$.

7) $E = \geq nS$. Par la propriété 9, on a $\sharp S^{\mathcal{I}}(s) \geq n$. Alors il y a n individus t_1, t_2, \dots, t_n tels que $t_i \neq t_j$ pour $i \neq j$, $\langle s, t_i \rangle \in \mathcal{E}_A(S) \forall 1 \leq i \leq n$. Parce que $\mathcal{E}_A(S) \subseteq S^{\mathcal{I}}$ alors $s \in E^{\mathcal{I}}$.

8) $E = \leq nS$. Par la propriété 10, on a $\sharp S^{\mathcal{I}}(s) \leq n$. Parce que S est le rôle simple, par la définition $S^{\mathcal{I}} = \mathcal{E}_A(S)$. Par conséquent, $\sharp \{t. \langle s, t \rangle \in S^{\mathcal{I}}\} \leq n$. Alors $s \in E^{\mathcal{I}}$.

9) $E = o$. Soient $E \in \mathcal{L}(s)$ et $E \in \mathcal{L}(t)$. Par la propriété 11, on a $s = t$ alors E est interprété correctement comme un nominal et $s \in E^{\mathcal{I}}$.

10) $E = \exists U.d$. La propriété 14 implique l'existence d'une valeur $t \in \mathbf{S}_D$ telle que $\langle s, t \rangle \in \mathcal{E}_D(U)$ et $t \in d^D$. Par la définition de $U^{\mathcal{I}}$ on a $\langle s, t \rangle \in U^{\mathcal{I}}$. Alors $s \in (\exists U.d)^{\mathcal{I}}$.

11) $E = \forall U.d$. Soit $t \in \mathbf{S}_D$ une valeur arbitraire telle que $\langle s, t \rangle \in U^{\mathcal{I}}$. Par la définition de $U^{\mathcal{I}}$, $\langle s, t \rangle \in \mathcal{E}_D(U)$. Alors par la propriété 15 $t \in d^D$ et $s \in E^{\mathcal{I}}$.

12) $E \geq nU$. Par la propriété 12, on a $\#U^T(s) \geq n$. Par la définition, $U^{\mathcal{I}} = \mathcal{E}_D(U)$ alors on a $\#\{t.\langle s, t \rangle \in U^{\mathcal{I}}\} \geq n$ et $s \in E^{\mathcal{I}}$.

13) $E \leq nU$. Par la propriété 13, on a $\#U^T(s) \leq n$. Par la définition de $U^T(s)$ on a $\#\{t.\langle s, t \rangle \in \mathcal{E}_D(U)\} \leq n$. Par la définition $U^{\mathcal{I}} = \mathcal{E}_D(S)$ alors $\#\{t.\langle s, t \rangle \in U^{\mathcal{I}}\} \leq n$. Alors $s \in E^{\mathcal{I}}$.

Maintenant, il nous reste de montrer que \mathcal{I} est le modèle de \mathcal{K} . Soit $(R_1, \dots, R_n \text{ Idfor } C) \in \mathcal{K}$.

Si $s, s' \in C^{\mathcal{I}}$ tels que $\langle s, t_i \rangle$, et $\langle s', t'_i \rangle \in R_i^{\mathcal{I}}$ et $t_i = t'_i$ pour tout $1 \leq i \leq n$. Par la propriété 17 de la définition 4.1 on a $\{C, \neg C\} \cap \mathcal{L}(s) \neq \emptyset$ et $\{C, \neg C\} \cap \mathcal{L}(s') \neq \emptyset$. Si $\neg C \in \mathcal{L}(s)$ alors $s \in (\neg C)^{\mathcal{I}}$, contradictoire avec l'hypothèse. Donc on a $C \in \mathcal{L}(s)$ et $C \in \mathcal{L}(s')$ analogiquement. Alors $C \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$. Par la propriété 18, alors $s = s'$.

Si $s \in C^{\mathcal{I}} \forall \langle s, t_i \rangle$, et $\langle s, t'_i \rangle \in R_i^{\mathcal{I}}$ pour tout $1 \leq i \leq n$. Par la propriété 18 de la définition 4.1, $(\leq 1R_i) \in \mathcal{L}(s)$ alors $\#\{t.\langle s, t \rangle \in R_i^{\mathcal{I}}\} \leq 1$. Par l'hypothèse alors $t_i = t'_i$.

Donc \mathcal{I} satisfait \mathcal{K} .

Pour la direction "seulement si", nous construisons un tableau de D si D a un modèle \mathcal{I} e.v.d \mathcal{R} et \mathcal{K} . Soit $\mathcal{I} = (\Delta^{\mathcal{I}}, \mathcal{I})$ le modèle de D avec $\mathcal{I} \models \mathcal{R}$ et $\mathcal{I} \models \mathcal{K}$. Nous construisons un tableau pour D $T = (\mathbf{S}_A, \mathbf{S}_D, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_D)$ comme suit :

$$\begin{aligned} \mathbf{S}_A &:= \Delta^{\mathcal{I}} \\ \mathbf{S}_D &:= \Delta_D \\ \mathcal{E}_A(R) &:= R^{\mathcal{I}} \\ \mathcal{E}_D(U) &:= U^{\mathcal{I}} \\ \mathcal{L}(s) &:= \{C \in cl(D) \mid s \in C^{\mathcal{I}}\} \end{aligned}$$

Maintenant, nous montrons que T est le tableau pour D e.v.d \mathcal{R} et \mathcal{K} , i.e. T satisfait des propriétés de tableau.

– Des propriétés 1-3, 5-6, 8-15 sont satisfaites comme le résultat direct de la définition de l'interprétation \mathcal{I} et de \mathcal{L} .

– Propriétés 4 et 16 sont satisfaites parce que $\mathcal{I} \models \mathcal{R}$

– Si $s \in (\forall S.C)^{\mathcal{I}}$ et $\langle s, t \rangle \in R^{\mathcal{I}}$ avec $\text{Trans}(R)$ et $R \boxtimes S$. Supposons que $t \notin (\forall R.C)^{\mathcal{I}}$ alors $\forall u$ tel que $\langle t, u \rangle \in R^{\mathcal{I}}$ (parce que R est transitif) $u \notin C^{\mathcal{I}}$. Parce que $\langle s, t \rangle \in R^{\mathcal{I}}$, $\langle t, u \rangle \in R^{\mathcal{I}}$ et $\text{Trans}(R)$ alors $\langle s, u \rangle \in R^{\mathcal{I}}$. On a $R \boxtimes S$ alors $\langle s, u \rangle \in S^{\mathcal{I}}$ et $s \notin \forall S.C$ - contradictoire avec l'hypothèse. Alors T satisfait la propriété 7.

– La propriété 17,18 sont satisfaites par la définition de l'interprétation de \mathcal{I} , de \mathcal{E} et de \mathcal{L} dans T .

■

4.2. Construction du Tableau pour $\mathcal{SHOLNK}(\mathbf{D})$

Par le lemme 4.1 prouvé ci-dessus, on peut vérifier la satisfiabilité du concept D en construisant un tableau pour D . Si on réussit à construire un tel tableau, le concept D est satisfiable. Cette section représente l'algorithme pour construire ce tableau.

Définition 4.2 (Graphe d'accomplissement) Soit D un concept dans $\mathcal{SHOLNK}(\mathbf{D})$ sous forme NNF, \mathcal{R} l'hérarchie de rôles, \mathcal{K} la Kbox sous forme NNF, \mathbf{R}_A^D l'ensemble des rôles abstraits dans $cl(D)$, \mathcal{K} ou \mathcal{R} .

Un graphe d'accomplissement pour un concept D de $\mathcal{SHOLNK}(\mathbf{D})$ e.v.d \mathcal{K} et \mathcal{R} est un graphe orienté $\mathbf{G} = (V_A, V_D, E, \mathcal{L}, \neq, \doteq)$ dont

– chaque noeud abstrait $x \in V_A$ est étiqueté par un ensemble

$$\mathcal{L}(x) \subseteq cl(D) \cup N_I \cup \{(\leq mR) | (\leq nR) \in cl(D) \text{ et } m \leq n\} \cup \{(\leq 1R) | R \in Kbox\},$$

où N_I est l'ensemble des nominaux.

– chaque noeud concret $x \in V_D$ est étiqueté par un ensemble $\mathcal{L}(x) \subseteq cl_D(D)$ où $cl_D(D)$ est l'ensemble de tous les types de données (peut-être négatifs) dans $cl(D)$,

– chaque arête $\langle x, y \rangle \in E$ est étiquetée par un ensemble $\mathcal{L}(\langle x, y \rangle)$ de rôles (peut être l'inverse) dans $cl(D)$, \mathcal{K} ou \mathcal{R} ,

– \neq est une relation binaire et symétrique entre des noeuds dans le graphe pour distinguer des individus et des valeurs inégaux.

– \doteq est une relation binaire et symétrique entre des noeuds égaux dans le graphe pour indiquer leur égalité.

Si $\langle x, y \rangle \in E$ alors y est appelé successeur de x et x est appelé prédécésseur de y . Ancêtre est la fermeture transitive de prédécésseur et descendant est la fermeture transitive de successeur. Un noeud y est appelé un R -successeur d'un noeud x si y est un successeur de x et $S \in \mathcal{L}(\langle x, y \rangle)$ avec $S \sqsubseteq R$. Un noeud y est appelé un R -voisin d'un noeud x si y est un R -successeur de x ou si x est un $\text{Inv}(R)$ -successeur de y .

Pour un rôle R et un noeud x dans le graphe \mathbf{G} , on définit l'ensemble des R -voisins de x , $R^{\mathbf{G}}(x)$, comme suit :

$$R^{\mathbf{G}}(x) = \{y | y \text{ est un } R\text{-voisin de } x\}$$

Définition 4.3 (Clash) Pour un noeud x , $\mathcal{L}(x)$ contient un clash si une des conditions suivantes est satisfaite :

- 1) Pour un concept primitif A , $\{A, \neg A\} \subseteq \mathcal{L}(x)$,
- 2) Pour un rôle (peut-être concret) S , $(\leq nS) \in \mathcal{L}(x)$ et il y a $n + 1$ S -voisins (S -successeurs en cas de rôle concret) y_0, \dots, y_n de x tels que $y_i \neq y_j \forall 0 \leq i < j \leq n$,
- 3) Pour deux noeuds $x \neq y$, le raisonnement concret retourne $x \doteq y$,
- 4) Pour un $o \in N_I$ il y a des noeuds $x \neq y$ avec $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$,
- 5) Il existe un noeud concret dont label contient n types de données (peut-être négatifs) d_1, \dots, d_n et $d_1^{\mathbf{D}} \cap \dots \cap d_n^{\mathbf{D}} = \emptyset$.
- 6) Pour 2 noeuds $s \neq s'$ il y a un concept C tel que $C \in \mathcal{L}s \cap \mathcal{L}s'$ et R_1, \dots, R_n **Idfor** $C \in \mathcal{K}$, $R_i \in L(\langle s, t_i \rangle) \cap L(\langle s', t_i \rangle) \forall 1 \leq i \leq n$,

Un graphe est *clash-libre* ssi aucun de ses noeuds contient un *clash*. Un graphe est *complet* si aucune règle dans les tableaux 1 et 2 est applicable ou pour un noeud x dans le graphe, $\mathcal{L}(x)$ contient un *clash*.

Si o_1, \dots, o_l sont des nominaux dans D , pour vérifier la satisfiabilité d'un concept D , l'algorithme initialise un graphe d'accomplissement

$$\mathbf{G} = (\{r_0, r_1, \dots, r_l\}, \emptyset, \emptyset, \{\mathcal{L}(r_0) = \{D\}, \mathcal{L}(r_i) = \{o_i\}\}, \emptyset, \emptyset) \text{ pour } 1 \leq i \leq l$$

Le graphe \mathbf{G} est développé en appliquant des règles des tableaux 1 et 2 jusqu'à qu'aucune règle ne puisse être appliquée ou s'il y a un *clash*. Si le graphe est complet et *clash-libre* alors D est satisfiable e.v.d \mathcal{R} et \mathcal{K} .

\mathbf{G} contient deux types de noeuds qui sont les abstraits et les concrets. Les noeuds concrets en effet ne peuvent qu'être les feuilles dans les sous-arbres. Les noeuds abstraits contiennent les *nominaux* et les *blocables* (comme définis dans (Horrocks *et al.*, 2005)). Ceux qui représentent nominaux peuvent être reliés arbitrairement. Au contraire, les blocables n'existent que dans les sous-arbres enracinés sur les noeuds nominaux ou sur r_0 . Une branche est terminée par un noeud bloqué, ou par un noeud nominal, ou par une feuille représentant une valeur concrète. Alors le blocage n'applique jamais aux noeuds nominaux et aux noeuds concrets. Basant sur un langage expressif avec l'hierarchie de rôles et les rôles inverses, nous utilisons aussi la condition de blocage pour \mathcal{SHOIQ} .

Définition 4.4 (Blocage par paire) *Un noeud est bloqué ssi il est bloqué directement ou indirectement. Un noeud x est bloqué directement ssi aucun de ses ancêtres est bloqué et il a 3 ancêtres x', y, y' tels que*

- 1) x est un successeur de x' , y est un successeur de y'
- 2) y, x et tous les noeuds sur le chemin de y à x sont blocables,
- 3) $\mathcal{L}(x) = \mathcal{L}(y)$ et $\mathcal{L}(x') = \mathcal{L}(y')$
- 4) $\mathcal{L}(\langle x', x \rangle) = \mathcal{L}(\langle y', y \rangle)$

et on l'appelle que y bloque x .

Un noeud y est bloqué indirectement ssi il est blocable et un de ses ancêtres est bloqué.

Comme expliquée ci-dessus, notre solution d'ajout de CI ne provoque pas de nouvelles conditions de fusionnement. Alors nous profitons du mécanisme de fusion qui inclut les fonctions Merge et Prune définies dans (Horrocks *et al.*, 2005). Par contre, nous modifions la fonction Prune en accord avec les domaines concrets appliqués pour CIs.

Fusionnement. Le fusionnement définit la fonction $Merge(y, x)$ qui fait la fusion de y à x dans un graphe $\mathbf{G} = (V_A, V_D, E, \mathcal{L}, \dot{=}, \dot{=})$ et crée un graphe obtenu du \mathbf{G} comme suit :

- 1) pour tout noeud z tel que $\langle z, y \rangle \in E$
 - a) si $\{\langle x, z \rangle, \langle z, x \rangle\} \cap E = \emptyset$ alors ajouter $\langle z, x \rangle$ à E , et mettre $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, y \rangle)$,
 - b) si $\langle z, x \rangle \in E$ alors mettre $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, x \rangle) \cup \mathcal{L}(\langle z, y \rangle)$,
 - c) si $\langle x, z \rangle \in E$ alors mettre $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle x, z \rangle) \cup \{\text{Inv}(S) \mid S \in \mathcal{L}(\langle z, y \rangle)\}$,
 - d) enlever $\langle z, y \rangle$ de E ;
- 2) pour tout z le noeud nominal tel que $\langle y, z \rangle \in E$
 - a) si $\{\langle x, z \rangle, \langle z, x \rangle\} \cap E = \emptyset$ alors ajouter $\langle x, z \rangle$ à E , et mettre $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle y, z \rangle)$,
 - b) si $\langle x, z \rangle \in E$ alors mettre $\mathcal{L}(\langle x, z \rangle) = \mathcal{L}(\langle x, z \rangle) \cup \mathcal{L}(\langle y, z \rangle)$,
 - c) si $\langle z, x \rangle \in E$ alors mettre $\mathcal{L}(\langle z, x \rangle) = \mathcal{L}(\langle z, x \rangle) \cup \{\text{Inv}(S) \mid S \in \mathcal{L}(\langle y, z \rangle)\}$,
 - d) enlever $\langle y, z \rangle$ de E
- 3) $\mathcal{L}(x) = \mathcal{L}(x) \cup \mathcal{L}(y)$
- 4) ajouter $x \dot{=} t$ pour tout t tel que $y \dot{=} t$
- 5) Prune(y).

$Prune(y)$ est une fonction pour enlever le noeud y et les sous-arbres de y dans \mathbf{G} . On peut décrire cette fonction en détail comme suit :

- 1) pour tout successeur z de y , enlever $\langle y, z \rangle$, si z n'est pas soit un nominal, soit un successeur d'un autre noeud alors Prune(z);
- 2) enlever y

A part les nominaux existants dans D , dans N_I il y a aussi des nouveaux nominaux créés par les contraintes de cardinalité et par CIs. On va prouver qu'il est possible de fixer la borne supérieure du nombre des nouveaux noeuds nominaux. C'est nécessaire pour montrer que l'algorithme se termine.

Comme dans (Horrocks *et al.*, 2005), on utilise des R-voisins sûrs pour assurer que le nombre de R-voisins d'un noeud nominal est assez produit. Un R-voisin y d'un

noeud x est appelé *sûr* si x est blocable ou x est un noeud nominal et y n'est pas bloqué.

Le niveau des noeuds nominaux. Puisqu'il est possible que les noeuds identifiés n'aient pas de voisins nominaux, nous mettons le niveau 0 sur ces noeuds. Le niveau de nominal est utilisé pour fixer la priorité d'application des règles d'extension sur le graphe. Quand un noeud au niveau plus bas est fusionné à un autre, le niveau du dernier noeud peut-être diminué parce que le fusionnement sauvegarde toutes les connections entre des noeuds nominaux. Les règles sont appliquées aux noeuds du niveau le plus bas jusqu'au niveau le plus haut.

Soient o_1, \dots, o_l les nominaux dans le concept d'entrée D .

- chaque noeud x avec $o_i \in \mathcal{L}(x)$, $1 \leq i \leq l$ est de niveau zero,
- un noeud nominal x est de niveau i si x n'est pas de niveau j avec $j < i$ et x a un voisin de niveau $i - 1$.
- un noeud nominal x est de niveau zero si il n'a pas de voisin nominal.

La priorité d'application des règles. Pour les règles identiques aux celles pour $SHOIQ$, nous conservons la priorité d'application comme pour $SHOIQ$ (Horrocks et al., 2005). Les nouvelles règles pour le domaine concret et la CI sont appliquées avec la priorité la plus basse. Les règles pour la CI sont appliquées après la v -règle. Précisément,

- 1) o -règle est appliquée avec la plus haute priorité,
- 2) suivant, applique la \leq - et la NN-règle. Elles sont appliquées d'abord pour des nominaux du niveau le plus bas auquel le plus haut. Au cas où ces deux règles sont appliquées avec le même noeud, la NN-règle est appliquée avant,
- 3) les autres règles pour $SHOIN(\mathbf{D})$ sont appliquées après,
- 4) v -règle est appliquée juste quand deux noeuds concrets sont égaux.
- 5) les autres règles pour la CI sont appliquées avec la priorité la plus basse (après la v -règle).

Cette stratégie de priorité est cruciale pour la terminaison de l'algorithme.

Dans le tableau 2, on voit encore les nouvelles règles sauf les règles v -règle et \mathcal{K} -règle qui sont abordées ci-dessus. Les deux nouvelles \mathcal{K}_+ - et $\leq_{\mathcal{K}}$ -règles sont construites afin d'assurer que la sémantique de la CI est capturée complètement.

5. Discussion et conclusion

Un concept analogue à celui de contrainte d'identification et souvent utilisé dans la conception de la base de données est le concept de "clé". Essentiellement, ces deux concepts impliquent une relation 1-1 entre une partie déterminante et la partie déterminée. En considérant la famille des logiques de description comme un candidat dans la conception de base de données grâce à son ingénieux compromis entre l'expressivité

\sqcap -règle :	si $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, et $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ alors $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
\sqcup -règle :	si $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, et $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ alors $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ où $C \in \{C_1, C_2\}$
\exists -règle :	si $\exists R.C \in \mathcal{L}(x)$, x n'est pas bloqué, et aucun R -voisin sûr y de x avec $C \in \mathcal{L}(y)$ alors produire un nouveau noeud y avec $\mathcal{L}(\langle x, y \rangle) = \{R\}$, $\mathcal{L}(y) = \{C\}$
\exists_D -règle :	si $\exists U.d \in \mathcal{L}(x)$, x n'est pas bloqué, et aucun U -successeur y de x avec $d \in \mathcal{L}(y)$ alors produire une nouvelle feuille y avec $\mathcal{L}(\langle x, y \rangle) = \{U\}$, $\mathcal{L}(y) = \{d\}$
\forall -règle :	si $\forall R.C \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, et il y a R -voisin y de x avec $C \notin \mathcal{L}(y)$ alors $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
\forall_D -règle :	si $\forall U.d \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, et il y a U -successeur y de x avec $d \notin \mathcal{L}(y)$ alors $\mathcal{L}(y) = \mathcal{L}(y) \cup \{d\}$
\forall_+ -règle :	si $\forall S.C \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, et il y a R avec $\text{Trans}(R)$ et $R \boxtimes S$, il y a R -voisin y de x avec $\forall R.C \notin \mathcal{L}(y)$ alors $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$
\geq -règle :	si $\geq nR \in \mathcal{L}(x)$, x n'est pas bloqué, et il n'y a pas n R -voisins sûrs y_1, \dots, y_n de x avec $y_i \dot{\neq} y_j$ pour tout $1 \leq i < j \leq n$ alors produire n nouveaux noeuds y_1, \dots, y_n avec $\mathcal{L}(\langle x, y_i \rangle) = \{R\}$ et $y_i \dot{\neq} y_j$ pour tout $1 \leq i < j \leq n$.
\geq_D -règle :	si $\geq nU \in \mathcal{L}(x)$, x n'est pas bloqué, et il n'y a pas n U -successeurs y_1, \dots, y_n de x avec $y_i \dot{\neq} y_j$ pour tout $1 \leq i < j \leq n$ alors produire n nouvelles feuilles y_1, \dots, y_n avec $\mathcal{L}(\langle x, y_i \rangle) = \{U\}$ et $y_i \dot{\neq} y_j$ pour tout $1 \leq i < j \leq n$.
\leq -règle :	si $\leq nR \in \mathcal{L}(z)$, z n'est pas indirectement bloqué, $\#R^G(z) > n$ et il y a deux R -voisins x, y de z sans $x \dot{\neq} y$ alors si x est un noeud nominal alors $\text{Merge}(y, x)$ sinon si y est un noeud nominal ou l'ancêtre de x alors $\text{Merge}(x, y)$ sinon $\text{Merge}(y, x)$;
\leq_D -règle :	si $\leq nU \in \mathcal{L}(x)$, x n'est pas indirectement bloqué, $\#U^T(x) > n$ et il y a deux U -successeurs y, z de x sans $y \dot{\neq} z$ alors $\text{Merge}(y, z)$
NN-règle :	si $\leq nS \in \mathcal{L}(x)$, x est un noeud nominal et il y a un noeud blockable y , S -voisin de x , x est un successeur de y et, il n'y a pas de m tel que $1 \leq m \leq n$, $(\leq mS) \in \mathcal{L}(x)$ et m S -voisins nominaux z_1, \dots, z_m de x avec $z_i \dot{\neq} z_j$ pour tout $1 \leq i < j \leq m$ alors "deviner" m avec $1 \leq m \leq n$ et mettre $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leq mS)\}$ produire m nouveaux noeuds y_1, \dots, y_m avec $\mathcal{L}(\langle x, y_i \rangle) = \{S\}$, $L(y_i) = \{o_i\}$ pour tout $o_i \in N_I$ nouveau dans \mathbf{G} , $y_i \dot{\neq} y_j$ pour $1 \leq i < j \leq m$.
o -règle :	si pour $o \in N_I$, il y a deux noeuds x, y avec $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ sans $x \dot{\neq} y$ alors $\text{Merge}(x, y)$

Tableau 1. Règles d'extension pour $SHOIN(\mathbf{D})$

v -règle :	si	il y a deux noeuds x, y sans $x \neq y$, le raisonnement concret retourne $x \doteq y$
	alors	$\text{Merge}(x, y)$
\mathcal{K}_+ -règle :	si	s a R_i -voisin t_i pour tout $1 \leq i \leq n$ et $R_1, \dots, R_n \mathbf{Idfor} C \in \mathcal{K}$, s n'est pas bloqué et $\{C, \neg C\} \cap \mathcal{L}(s) = \emptyset$,
	alors	$\mathcal{L}(s) = \mathcal{L}(s) \cup \{E\}$ pour $E \in \{C, \neg C\}$
$\leq_{\mathcal{K}}$ -règle :	si	il existe $C \in \mathcal{L}(s)$, $R_1, \dots, R_n \mathbf{Idfor} C \in \mathcal{K}$ et $\leq 1R_i \notin \mathcal{L}(s)$ pour $1 \leq i \leq n$
	alors	$\mathcal{L}(s) = \mathcal{L}(s) \cup \{\leq 1R_i\}$ pour tout R_i -voisin de s , $1 \leq i \leq n$
\mathcal{K} -règle :	si	il existe $C \in \mathcal{L}(s) \cap \mathcal{L}(s')$, $R_1, \dots, R_n \mathbf{Idfor} C \in \mathcal{K}$, et $R_i \in \mathcal{L}(\langle s, t_i \rangle) \cap \mathcal{L}(\langle s', t_i \rangle)$, $\forall 1 \leq i \leq n$ sans $s \neq s'$
	alors	si s est un noeud nominal alors $\text{Merge}(s', s)$ sinon si s' est un noeud nominal alors $\text{Merge}(s, s')$ sinon si t_i est un R_i -successeur de s et de $s' \forall 1 \leq i \leq n$ et prédécesseur de s n'est pas un prédécesseur de s' alors $\mathcal{L}(s) = \mathcal{L}(s) \cup \{o_{\mathcal{K}}\}$ et $\mathcal{L}(s') = \mathcal{L}(s') \cup \{o_{\mathcal{K}}\}$ pour $o_{\mathcal{K}} \in N_I$ nouveau dans \mathbf{G} sinon si s' est un descendant de s alors $\text{Merge}(s', s)$ sinon $\text{Merge}(s, s')$

Tableau 2. Nouvelles règles d'extension pour $\mathcal{SHOINK}(\mathbf{D})$

de connaissances et la complexité de raisonnement [(Calvanese *et al.*, 1998), (Borgida *et al.*, 1997)], plusieurs chercheurs ont essayé d'ajouter la notion de "clé" aux LDs, ce qui n'est pas une propriété naturelle de ces dernières. Une piste consiste à introduire la clé comme un nouveau concept dans le langage [(Borgida *et al.*, 1997), (Khizder *et al.*, 2000), (Calvanese *et al.*, 1995)], mais l'approche est limitée. Premièrement, dans la plupart des travaux, la contrainte "clé" ne s'applique que sur un sous-ensemble d'instances de concept devant satisfaire la contrainte. Ainsi la sémantique de la contrainte n'est pas complètement capturée. Deuxièmement, l'interprétation de ce concept n'assure pas la relation 1-1 entre la clé et le concept identifié. Cela peut créer des situations contradictoires dans lesquelles la contrainte n'est plus satisfaite sur toutes les instances. En outre, dans ces approches, l'utilisation de la notion de concept pour représenter la notion de clé établit un cloisonnement entre les concepts clé. Par exemple, soient K et K' deux concepts clé générés par mettre la contrainte "clé" sur le concept C . La seule différence dans ces deux définitions de concept clé est que l'ensemble "clé" de K est subsumé par celui de K' . Cependant on ne peut pas conclure qu'il existe une relation entre K et K' (qui doit être $K \sqsubseteq K'$). De plus, les langages LDs appliqués dans ces approches sont limités soit en expressivité, soit en décidabilité.

Diego Calvanese *et al.* a surmonté ces problèmes par une autre approche, ainsi la contrainte est définie comme une assertion de clé (Calvanese *et al.*, 2001). Dans leur approche, l'assertion de clé a la même syntaxe que celle dans les dernières approches mais est formalisée comme un nouveau type dans TBox. L'assertion définit directe-

ment la clé pour un concept défini. Ainsi le problème de poser la contrainte sur le sous-ensemble d'un concept comme montré ci-dessus ne se produit pas. Cependant, une des limites de cette approche est que dans certains cas (par exemple la dépendance fonctionnelle unaire) cette représentation étendue entraîne l'indécidabilité. De plus, il n'y a pas une procédure pratique de décision connue pour résoudre le problème de décidabilité dans \mathcal{DLR} , le langage proposé dans (Calvanese *et al.*, 2001). Même si une telle procédure existe, elle cause quelques problèmes de *solubilité* (Horrocks *et al.*, 2000). Cette approche est également suivie par C. Lutz et al. (Lutz *et al.*, 2003). Ils ne maintiennent pas les assertions de clé dans Tbox mais dans une nouvelle boîte appelée KBox, et introduisent les contraintes aux LDs étendus avec le domaine concret $\mathcal{ALCO}(\mathcal{D})$ et $\mathcal{SHOQ}(\mathcal{D})$ (c.à.d $\mathcal{ALCOK}(\mathcal{D})$ et $\mathcal{SHOQK}(\mathcal{D})$ correspondants). Dans cette approche, les auteurs ont proposé des algorithmes de raisonnement avec la complexité en NEXPTIME pour ces deux langages. La relation 1-1 de la notion de clé est aussi bien respectée dans ces langages. Cependant pour cela, les auteurs emploient les rôles fonctionnels concrets qui restreignent la clé aux seulement valeurs. Ceci par conséquent ne permet pas de décrire toutes les possibilités de la contrainte. Notre travail, dans cette approche également, prend en compte cela en permettant d'identifier une entité (c.à.d un concept) par un ensemble d'objets qui sont soit des entités soit des valeurs. En outre, quoique Lutz et al. ajoutent les contraintes "clé" à LDs avec un domaine concret plus expressif (c.à.d prédicats n -aire), il y a plusieurs restrictions sur KBox pour s'assurer de la décidabilité des langages. En outre, limité par le rôle fonctionnel concret, $\mathcal{ALCOK}(\mathcal{D})$ et $\mathcal{SHOQK}(\mathcal{D})$ ne permettent pas certains constructeurs concrets (c.à.d $\forall U.d, \leq nU, \geq nU$) qui sont permis dans $\mathcal{SHOINK}(\mathbf{D})$. Même si le domaine concret dans $\mathcal{SHOINK}(\mathbf{D})$ (c.à.d. datatypes concrets) est moins expressif, il joue un rôle important dans la construction des ontologies. Nous augmentons donc la puissance expressive de la LD par l'extension avec la contrainte CI en garantissant les possibilités de la représentation des connaissances pour des ontologies.

En ce qui concerne les algorithmes de décision, $\mathcal{SHOIN}(\mathbf{D})$ est déjà connu avec la complexité de NExpTime-complet (Tobies, 2000). Alors pour $\mathcal{SHOINK}(\mathbf{D})$, la procédure de décision, dans le pire des cas en pratique, ne serait pas terminée. Pour éviter ces problèmes, nous avons construit notre algorithme comme des composants disloquables. En absence de contrainte d'unicité, l'algorithme se comporte comme celui pour $\mathcal{SHOIN}(\mathbf{D})$. Au cas où aucune condition de la CI n'est satisfaite alors les règles d'extension ne sont pas appliquées. En conséquence, la performance de $\mathcal{SHOINK}(\mathbf{D})$ est comparable à celle de $\mathcal{SHOIN}(\mathbf{D})$. Remarquez que la solution proposée dans cet article est aussi applicable pour la langage $\mathcal{SHOIQ}(\mathbf{D})$.

Dans ce papier, nous présentons une solution afin d'exprimer la contrainte d'identification dans un langage de logique de description spécifique. Ce travail s'inscrit dans la perspective d'un projet global visant la construction d'une méthodologie d'intégration des sources de données relationnelles dans le Web sémantique. Cette intégration devrait être réalisée par l'intermédiaire de modèle ORM (Object Role Modeling) (Halpin, Novembre 2001). Le développement de notre projet sera effectué en deux étapes. La première étape consiste en une étude de transformation sans perte de la sémantique

, d'un schéma ORM dans un langage de description de type $\mathcal{SHOINK}(\mathbf{D})$. Dans une seconde étape nous orienterons vers la recherche des d'une traduction automatique d'une requête DL dans une requête SQL en utilisation les mappings établis entre le schéma ORM et le schéma DL.

6. Bibliographie

- Borgida A., Weddell G. E., « Adding Uniqueness Constraints to Description Logics (Preliminary Report) », *DOOD '97 : Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases*, Springer-Verlag, London, UK, p. 85-102, 1997.
- Calvanese D., De Giacomo G., Lenzerini M., « Structured Objects : Modeling and Reasoning », *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*, number 1013 in *Lecture Notes in Computer Science*, Springer, p. 229-246, 1995.
- Calvanese D., De Giacomo G., Lenzerini M., « Identification Constraints and Functional Dependencies in Description Logics », *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, p. 155-160, 2001.
- Calvanese D., Lenzerini M., Nardi D., « Description logics for conceptual data modeling », *Logics for databases and information systems*, p. 229-263, 1998.
- Halpin T., « Object Role Modeling : An overview », Novembre 2001.
- Horrocks I., Sattler U., « Ontology Reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ Description Logic », *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, Morgan Kaufmann, Los Altos, p. 199-204, 2001.
- Horrocks I., Sattler U., « A Tableaux Decision Procedure for \mathcal{SHOIQ} », *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, p. 448-453, 2005.
- Horrocks I., Sattler U., Tessaris S., Tobies S., « How to Decide Query Containment Under Constraints Using a Description Logic », *7th International Workshop on Knowledge Representation meets Databases (KRDB2000)*, 2000.
- Horrocks I., Sattler U., Tobies S., A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restrictions, LTCS-Report n° 99-08, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.
- Khizder V. L., Toman D., Weddell G. E., « On Decidability and Complexity of Description Logics with Uniqueness Constraints. », *Description Logics*, p. 193-202, 2000.
- Lutz C., Areces C., Horrocks I., Sattler U., « Keys, Nominals, and Concrete Domains », *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence IJCAI-03*, Morgan-Kaufmann Publishers, Acapulco, Mexico, 2003.
- Pan J. Z., Description Logics : Reasoning Support for the Semantic Web, PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK, 2004.
- Tobies S., « The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics », *Journal of Artificial Intelligence Research*, vol. 12, p. 199-217, 2000.

Annexe : Démonstration de la terminaison et de la correction de l’algorithme du Tableau

Lemme 6.1 (Terminaison) *Pour chaque concept D sous forme NNF dans $\mathcal{SHOIQ}(D)$, \mathcal{R} est l’hierarchie de rôles et \mathcal{K} est la Kbox, l’algorithme du Tableau se termine.*

Preuve. Soient $m = |cl(D)|$, k le nombre de rôles abstraits et de ses inverses dans D et \mathcal{R} , (n_{\geq}) le nombre maximal dans les restrictions de cardinalité supérieur, (n_{\leq}) le nombre minimal dans les restrictions de cardinalité inférieur, o_1, \dots, o_l tous les nominaux dans D , $\lambda = 2^{2m+k}$ et n_{\bowtie} le maximum de n_{\leq} et n_{\geq} .

L’algorithme construit un graphe qui se compose d’un ensemble de noeuds nominaux connectés arbitrairement, un ensemble de noeuds concrets et de “arbres” des noeuds blocables. Chaque “arbre” s’enracine dans r_0 ou dans un noeud nominal. Chaque branche peut se terminer à un noeud nominal ou à un noeud concret.

Selon la terminologie proposé par Ian Horrocks et al. (Horrocks *et al.*, 2005), nous distinguons les règles en deux catégories : les *règles de génération* qui génèrent les nouveaux noeuds consistent en $\exists, \exists_{\mathbf{D}}, \geq, \geq_{\mathbf{D}}, \text{NN}$ et \mathcal{K} -règles ; les *règles de réduction* (*shrinking rules* en anglais) qui réduisent le nombre des noeuds comprennent $\leq, \leq_{\mathbf{D}}, o, v$ et \mathcal{K} -règles. Remarquez que \mathcal{K} -règle, soit réduit le nombre des noeuds (en cas où un des noeuds appliqués est nominal ou les noeuds blocables appliqués ont la relation d’ancêtre/descendant ou le même prédécesseur), soit augmente le nombre des nouveaux noeuds nominaux en étendant les labels des deux noeuds blocables avec un nouveau nominal.

Précisément, la terminaison est grâce aux propriétés des règles d’extension au-dessous. Les quatres premières sont comparables auxquelles utilisées pour prouver la terminaison pour \mathcal{SHOIQ} (Horrocks *et al.*, 2005). Les dernières démontrent la borne supérieure sur le nombre des nouveaux noeuds nominaux générés par les NN et \mathcal{K} -règles.

1) Par les définitions des règles, toutes les règles sauf lesquelles de réduction développent le graphe en générant des nouveaux noeuds (et des arêtes correspondantes) ou en étendant le label d’un noeud.

2) Des nouveaux noeuds sont seulement générées par les règles de génération. Et ces règles sont appliquées au plus one fois pour chaque concept dans le label d’un noeud. Cette propriété est évident quand il n’y a pas de fusion. Au cas où il faille faire la fusion, elle arrive dans un des trois cas suivants. Remarquez que si y est fusionné à z et y est un S -voisin de x alors $\mathcal{L}(y)$ est ajouté à $\mathcal{L}(z)$, z hérite toutes les inégalités de y , et soit z est S -voisin de x (au cas où x soit un noeud nominal ou y soit un successeur de x), soit x est enlevé en appliquant $Prune(x)$ (au cas où x soit un noeud blocable et un successeur de y).

a) Pour \exists -règle ($\exists_{\mathbf{D}}$ -règle), si $\exists S.C \in \mathcal{L}(x)$ ($\exists U.d \in \mathcal{L}(x)$) alors un nouveau noeud y est créée avec $\mathcal{L}(\langle x, y \rangle) = \{S\}$ ($\mathcal{L}(\langle x, y \rangle) = \{U\}$) et $\mathcal{L}(y) = \{C\}$ ($\mathcal{L}(y) =$

$\{d\}$). Alors quand on fait fusion de y à un noeud z , soit x est enlevé en appliquant $Prune(x)$, soit x a z comme un S-voisin (U-successeur) avec $C \in \mathcal{L}(z)$ ($d \in \mathcal{L}(z)$). Par conséquent, \exists -règle ($\exists_{\mathbf{D}}$ -règle) ne s'applique plus sur $\exists S.C \in \mathcal{L}(x)$ ($\exists U.d \in \mathcal{L}(x)$).

b) Pour \geq -règle ($\geq_{\mathbf{D}}$ -règle), si $\geq nS \in \mathcal{L}(x)$ ($\geq nU \in \mathcal{L}(x)$) alors n nouveaux noeuds y_1, \dots, y_n sont créés avec $\mathcal{L}(\langle x, y_i \rangle) = \{S\}$ ($\mathcal{L}(\langle x, y_i \rangle) = \{U\}$) et $y_i \neq y_j$ pour $1 \leq i < j \leq n$. S'il faut faire fusion de y_i à z_i alors soit x est enlevé, soit x a z_1, \dots, z_n comme des S-voisins (U-successeurs) avec $z_i \neq z_j$ pour $1 \leq i < j \leq n$. Par conséquent, la \geq -règle ($\geq_{\mathbf{D}}$ -règle) n'est plus applicable sur $\geq nS \in \mathcal{L}(x)$ ($\geq nU \in \mathcal{L}(x)$).

c) Pour NN-règle, si $\leq nS \in \mathcal{L}(x)$ alors m nouveaux noeuds nominaux y_1, \dots, y_m sont créés avec $\mathcal{L}(\langle x, y_i \rangle) = \{S\}$, $y_i \neq y_j$ pour $1 \leq i < j \leq m$, $1 \leq m \leq n$ et $\leq mS \in \mathcal{L}(x)$. S'il faut faire la fusion de y_i à z_i alors $\leq mS$ encore dans $\mathcal{L}(x)$ et x a z_1, \dots, z_m noeuds nominaux comme des S-voisins avec $z_i \neq z_j$ pour $1 \leq i < j \leq m$. Par conséquent, la NN-règle n'est plus applicable sur $\leq nS \in \mathcal{L}(x)$.

d) Comme une règle de génération, la \mathcal{K} -règle est appliqué au plus une fois sur une paire de noeuds blocables et aussi bien sur ces noeuds. En ajoutant un nouveau concept nominal aux deux labels des noeuds blocables, ces noeuds ne sont plus blocables. En plus, la o -règle qui est de la plus haute priorité a déjà fait la fusion ces deux noeuds. Alors la \mathcal{K} -règle n'est plus applicable sur cette paire de noeuds.

Comme pour \mathcal{SHOIQ} , l'application d'une règle de génération sur un concept dans le label d'un noeud x peut créer au plus (n_{\geq}) successeurs blocables de x . Parce qu'il y a au plus m concepts dans $L(x)$ alors le nombre maximal des successeurs blocables d'un noeud est $m \times n_{\geq}$.

3) Avec la condition de blocage par paire, la longueur d'un chemin qui comprend entièrement des noeuds blocables est limité par λ comme prouvé pour \mathcal{SHOIQ} (Horrocks *et al.*, 2005).

4) \mathcal{K} -règle est une règle spéciale qui s'applique au plus une fois pour une paire de noeuds. La raison est qu'en tout cas, les deux noeuds appliqués sont fusionnés (soit par \mathcal{K} -règle elle-même, soit par o -règle qui est de la plus haute priorité). Alors cette "paire" n'existent plus. En outre, cette règle appliquée, comme montré ci-dessus, au plus one fois sur un noeud blocable. En conséquence, le nombre de nouveaux noeuds nominaux créés par \mathcal{K} -règle est borné par la moitié du nombre de noeuds blocables qui sont appliqués par cette règle.

5) Chaque noeud abstrait peut avoir au plus $m \times n_{\geq U}$ successeurs concrets où $n_{\geq U}$ est le nombre maximum dans la restriction concrète de cardinalité supérieure. Le nombre de noeuds concrets est donc borné linéairement par le nombre de noeuds abstraits.

6) Soit $n_{\mathbf{D}}$ le nombre maximum des valeurs dans les domaines finis des types de données dans \mathbf{D} . Un nouveau noeud nominal créé par \mathcal{K} -règle peut prolonger un chemin du racine à un noeud bloqué au plus jusqu'à $n_{\mathbf{D}}$ fois le cycle de blocage (au cas où un nominal serait ajouté à un noeud blocable dans le cycle et casse le cycle). Par la propriété 3, ce cycle de blocage contient inférieur à λ noeuds blocables

(puisqu'au moins un noeud devient nominal). Par conséquent le nombre maximum des noeuds blocables générés le long d'un chemin est inférieur à $n_{\mathcal{D} >} \times \lambda$. Cependant, pour la même raison un nouveau noeud nominal généré par NN-règle ne peut prolonger un chemin qu'au plus n_{\leq} fois le cycle de blocage. Ainsi le nombre maximum des noeuds blocables produits le long de ce chemin est inférieur à $n_{\leq >} \times \lambda$. Soit $N_{\leq} = \text{Max}(n_{\leq}, n_{\mathcal{D} >})$. Le nombre de noeuds blocables le long d'un chemin est alors borné par $N_{\leq} \times \lambda$. Par la propriété 2, le nombre des chemins pour un "arbre" est borné par $(m \times n_{\geq})^\lambda$. Au début nous avons $(l + 1)$ "arbres". Par conséquent, le nombre de noeuds blocables dans \mathbf{G} est borné par $(l + 1) \times N_{\leq} \times \lambda \times (m \times n_{\geq})^\lambda$. Un nouveau noeud nominal est créé par \mathcal{K} -règle en ajoutant un nouveau nominal aux noeuds blocables, prédécesseurs de noeuds nominaux ou de noeuds concrets 3.1. Par la propriété 4 alors le nombre de noeuds nominaux produits par \mathcal{K} -règle est borné par $N_K = ((l + 1)/2) \times N_{\leq} \times \lambda \times (m \times n_{\geq})^\lambda$.

7) Le nombre total de nominaux est borné par $\mathcal{O}(l \times \lambda \times N_{\leq} \times (m \times n_{\text{v}})^{2\lambda})$.

À la différence de la situation pour \mathcal{SHOIQ} , les nouveaux nominaux sont maintenant générés non seulement par la NN-règle mais également par la \mathcal{K} -règle. D'ailleurs, il y a une interaction entre ces deux sources générant les nominaux. D'une part, les nouveaux nominaux générés par \mathcal{K} -règle peuvent être appliqués par NN-règle pour créer de nouveaux nominaux. D'autre part, les nouveaux nominaux générés par NN-règle peuvent être les noeuds identifiants qui font que la \mathcal{K} -règle s'applique sur deux noeuds blocables et peuvent créer un nouveau nominal. Puisque le nombre de noeuds nominaux générés par \mathcal{K} -règle est borné comme montré dans la propriété 6, nous devons encore seulement considérer le nombre de noeuds nominaux générés par NN-règle.

Remarquez que la NN-règle et la \mathcal{K} -règle sont seulement appliquées après ou autant qu'un nominal est ajoutés au label d'un noeud blocable x . En outre, ajoutant un nominal au label d'un noeud blocable peut entraîner à l'addition d'un nominal au label de son prédécesseur (par la \leq ou la \mathcal{K} -règle). Reprenant cet argument, il est possible que tous les ancêtres de x deviennent les noeuds nominaux. Au début, il n'y a que des nominaux de niveau 0. Par la définition du niveau de noeuds nominaux, l'application de la NN- ou \mathcal{K} -règle peut seulement créer les noeuds nominaux de niveau 0 ou 1. Puisque la longueur maximum d'un chemin contenant totalement des noeuds blocables est λ , les nouveaux noeuds nominaux créés peuvent seulement être de niveau inférieur ou égal à λ . En conséquence, nous pouvons calculer la borne supérieure du nombre de noeuds nominaux dans \mathbf{G} générés par NN-règle. le processus de comptage est terminé quand la \mathcal{K} -règle et la NN-règle ont généré tous les noeuds nominaux du niveau λ .

Au début, il y a l noeuds nominaux de niveau 0 dans le graphe. Par la propriété 6, le nombre de noeuds nominaux de niveau 0 créés par la \mathcal{K} -règle est borné par N_K . Le nombre total de noeuds nominaux de niveau 0 dans \mathbf{G} est donc borné par $N_K + l$. La NN-règle par conséquent s'applique sur les noeuds nominaux de niveau 0 et crée au plus $(m \times n_{\leq})(N_K + l)$ noeuds nominaux de niveau 1. La \mathcal{K} -règle génère des noeuds de niveau 1 à partir de prédécesseurs blocables des noeuds nominaux de niveau 0. Par la propriété 6, le nombre de noeuds nominaux de niveau 1 créés par la \mathcal{K} -règle

est également borné par N_K . Alors le nombre total de noeuds nominaux de niveau 1 dans \mathbf{G} est donc borné par $(m \times n_{\leq})(N_K + l) + N_K$. Analogiquement, les noeuds nominaux de niveau i consistent en ceux qui sont créés par l'application de NN-règle sur les noeuds de niveau $i - 1$ dans le graphe et ceux de niveau i créés par la \mathcal{K} -règle. En conséquence, le nombre total de noeuds nominaux de niveau i dans le graphe est borné par

$$l(m \times n_{\leq})^i + N_K \sum_{k=0}^i (m \times n_{\leq})^k.$$

Le processus de génération de noeuds nominaux est terminé quand la \mathcal{K} - et la NN-règle ont généré tous les noeuds nominaux de niveau λ . Le nombre de noeuds nominaux produits par NN-règle dans \mathbf{G} est donc borné par

$$l \sum_{i=1}^{\lambda} (m \times n_{\leq})^i + N_K \sum_{i=0}^{\lambda} \sum_{k=0}^i (m \times n_{\leq})^k.$$

Par la propriété 6, le nombre de noeuds nominaux produits par NN-règle dans \mathbf{G} est donc borné par $\mathcal{O}(l \times \lambda \times N_{\leq} \times (m \times n_{\leq})^{2\lambda})$. C'est également la borne supérieure du nombre total de nominaux dans \mathbf{G} .

■

Lemme 6.2 (Correction) *Si les règles d'extension peuvent être appliquées au concept D , \mathcal{R} et \mathcal{K} telles qu'elles produisent un graphe d'accomplissement complet et clash-libre alors D a un tableau pour une \mathcal{R} et \mathcal{K} .*

Preuve. La manière de démontrer est semblable à celle pour \mathcal{SHOIQ} . Un ensemble de chemins de \mathbf{G} est défini afin de manipuler la production des individus. Le tableau est défini à partir de graphe \mathbf{G} en prenant l'ensemble de nominaux et de chemins comme ensemble d'individus, l'ensemble de valeurs retourné en appliquant δ aux noeuds concrets de \mathbf{G} comme l'ensemble de valeurs concrètes, un label d'un noeud comme une fonction associant un individu à un ensemble de concepts. D'autres éléments dans le tuple de tableau sont alors définis basé sur les définitions données. En vérifiant les chemins en effilochant les parties d'"arbre" de \mathbf{G} , toutes les propriétés du tableau sont satisfaites.

Précisément, un *chemin* est une séquence des paires de noeuds blocables de \mathbf{G} sous forme $p := \langle (x_0, x'_0), \dots, (x_n, x'_n) \rangle$. Pour un tel chemin, on définit $\text{Tail}(p) := x_n$ et $\text{Tail}'(p) := x'_n$. $\langle p | (x_{n+1}, x'_{n+1}) \rangle$ est un chemin allongé de p et correspond à $\langle (x_0, x'_0), \dots, (x_n, x'_n), (x_{n+1}, x'_{n+1}) \rangle$. L'ensemble des chemins de \mathbf{G} Paths(\mathbf{G}) est défini inductivement comme suit :

– Pour chaque noeud blocable x qui est un successeur d'un noeud nominal ou une racine dans \mathbf{G} , $\langle (x, x) \rangle \in \text{Paths}(\mathbf{G})$;

– Pour un chemin $p \in \text{Paths}(\mathbf{G})$ et un noeud blocable $y \in \mathbf{G}$:

- si y est un successeur de $\text{Tail}(p)$ et y n'est pas bloqué alors $\langle p|(y, y) \rangle \in \text{Paths}(\mathbf{G})$, et

- si y est un successeur de $\text{Tail}(p)$ et y est bloqué par z , alors $\langle p|(z, y) \rangle \in \text{Paths}(\mathbf{G})$

Avec cette définition, on implique que si $p = \langle p'|(x, x') \rangle$ et x n'est pas bloqué, alors x' est bloqué ssi $x' \neq x$, x' n'est jamais bloqué indirectement. En outre, $\mathcal{L}(x) = \mathcal{L}(x')$ à cause de la définition de blocage.

Supposons qu'on a un graphe complet et clash-libre $\mathbf{G} = (V_A, V_D, E, L, \neq)$. Soit $\text{Nom}(\mathbf{G})$ l'ensemble des nominaux dans \mathbf{G} , on définit le tableau pour le graphe \mathbf{G} $T = (\mathbf{S}_A, \mathbf{S}_D, \mathcal{L}', \mathcal{E}_A, \mathcal{E}_D)$ comme suit :

$$\mathbf{S}_A = \{\text{Nom}(\mathbf{G}) \cup \text{Paths}(\mathbf{G})\}$$

$$\mathbf{S}_D = \{\delta(x) | x \in V_D\}$$

$$\mathcal{L}'(p) = \begin{cases} \mathcal{L}(\text{Tail}(p)) & \text{si } p \in \text{Paths}(\mathbf{G}); \\ \mathcal{L}(p) & \text{si } p \in \text{Nom}(\mathbf{G}). \end{cases}$$

$$\mathcal{E}_A(R) = \{\langle p, q \rangle \in \text{Paths}(\mathbf{G}) \times \text{Paths}(\mathbf{G}) |$$

$$q = \langle p|(x, x') \rangle \text{ et } x' \text{ est un R-successeur de Tail}(p) \text{ ou}$$

$$p = \langle q|(x, x') \rangle \text{ et } x' \text{ est un Inv}(R)\text{-successeur de Tail}(q)\} \cup$$

$$\{\langle p, x \rangle \in \text{Paths}(\mathbf{G}) \times \text{Nom}(\mathbf{G}) | x \text{ est un R-voisin de Tail}(p)\} \cup$$

$$\{\langle x, p \rangle \in \text{Nom}(\mathbf{G}) \times \text{Paths}(\mathbf{G}) | \text{Tail}(p) \text{ est un R-voisin de } x\} \cup$$

$$\{\langle x, y \rangle \in \text{Nom}(\mathbf{G}) \times \text{Nom}(\mathbf{G}) | y \text{ est un R-voisin de } x\}$$

$$\mathcal{E}_D(R) = \{\langle x, y \rangle \in \mathbf{S}_A \times \mathbf{S}_D | y \text{ est R-successeur de soit Tail}(x) \text{ au cas}$$

$$\text{où } x \in \text{Paths}(\mathbf{G}) \text{ soit } x \text{ si } x \in \text{Nom}(\mathbf{G})\}$$

Nous démontrons que T satisfait toutes les propriétés de la définition 4.1.

– $D \in \mathcal{L}(r_0)$ par la définition de l'algorithme. Si r_0 est un noeud nominal alors $D \in \mathcal{L}'(r_0)$. Sinon on a $D \in \mathcal{L}(\text{Tail}(\langle (r_0, r_0) \rangle))$ par la définition de $\text{Tail}(p)$ alors $D \in \mathcal{L}'(\langle (r_0, r_0) \rangle)$.

– **Propriété 1** est satisfait parce que \mathbf{G} est clash-libre. **Propriété 2 et 3** sont satisfaites parce que $\text{Tail}(p)$ n'est pas bloqué par la définition et \mathbf{G} est complet.

– **Propriété 4, 8 et 16** sont satisfaites par la définition de l'hierarchie de rôle, de R -successeur, de R -voisin et l'état complet et clash-libre du \mathbf{G} .

– **Propriété 5** Supposons $\forall R.C \in \mathcal{L}'(p)$ et $\langle p, q \rangle \in \mathcal{E}_A(R)$.

- Si $\langle p, q \rangle \in \text{Paths}(\mathbf{G}) \times \text{Paths}(\mathbf{G})$ alors $\forall R.C \in \mathcal{L}(\text{Tail}(p))$ par la définition de $\mathcal{L}'(p)$ et

- soit $\text{Tail}'(q)$ est un R -successeur de $\text{Tail}(p)$ par la définition de $\mathcal{E}_A(R)$. Dans ce cas là, l'application complète des règles assure que $C \in \mathcal{L}(\text{Tail}'(q))$. Par la définition de chemin alors soit $\text{Tail}'(q) = \text{Tail}(q)$, soit $\mathcal{L}(\text{Tail}'(q)) = \mathcal{L}(\text{Tail}(q))$ à cause de la condition de blocage. Alors $C \in \mathcal{L}(\text{Tail}(q))$. En conséquence $C \in \mathcal{L}'(q)$ par la définition de \mathcal{L}' .

- soit $\text{Tail}'(p)$ est un $\text{Inv}(R)$ – *successeur* de $\text{Tail}(q)$. Si $\text{Tail}'(p) = \text{Tail}(p)$ alors $\text{Tail}(p)$ est un $\text{Inv}(R)$ -successeur de $\text{Tail}(q)$. Alors $C \in \mathcal{L}(\text{Tail}(q))$. Si $\mathcal{L}(\text{Tail}'(p)) = \mathcal{L}(\text{Tail}(p))$ (au cas où la condition de blocage soit effectuée), alors $\forall R.C \in \mathcal{L}(\text{Tail}'(p))$. Par conséquent, $C \in \mathcal{L}(\text{Tail}(q))$. Par la définition, $C \in \mathcal{L}'(q)$.
 - Si $\langle p, q \rangle \in \text{Nom}(\mathbf{G}) \times \text{Nom}(\mathbf{G})$ alors $\forall R.C \in \mathcal{L}(p)$ et q est un R-voisin de p alors $C \in \mathcal{L}(q)$. Par la définition, $C \in \mathcal{L}'(q)$.
 - Si $\langle p, q \rangle \in \text{Nom}(\mathbf{G}) \times \text{Paths}(\mathbf{G})$ alors $\forall R.C \in \mathcal{L}(p)$. $\text{Tail}(q)$ est un R-voisin de p alors $C \in \mathcal{L}(\text{Tail}(q))$. Par la définition, $C \in \mathcal{L}'(q)$.
 - Si $\langle p, q \rangle \in \text{Paths}(\mathbf{G}) \times \text{Nom}(\mathbf{G})$ alors $\forall R.C \in \mathcal{L}(\text{Tail}(p))$. Parce que q est un R-voisin de $\text{Tail}(p)$ alors $C \in \mathcal{L}(q)$. Par conséquent $C \in \mathcal{L}'(q)$
- **Propriété 6** Supposons $\exists R.C \in \mathcal{L}'(x)$ avec $x \in \mathbf{S}_A$.
 - Si $x \in \text{Paths}(\mathbf{G})$ alors $\exists R.C \in \mathcal{L}(\text{Tail}(x))$. Parce que $\text{Tail}(x)$ n'est pas bloqué et le graph \mathbf{G} est complet, il doit exister un R-voisin y de $\text{Tail}(x)$ avec $C \in \mathcal{L}(y)$.
- si y est un noeud nominal alors $y \in S_A$ et $\mathcal{L}'(y) = \mathcal{L}(y)$. Par conséquent, $C \in \mathcal{L}'(y)$ et $\langle x, y \rangle \in \mathcal{E}_A(R)$.
- si y est un noeud blocable et un successeur de $\text{Tail}(x)$ alors $\langle x | (y', y) \rangle \in \mathbf{S}_A$ avec soit $y' = y$ soit y' bloque y et $\langle x, \langle x | (y', y) \rangle \rangle \in \mathcal{E}_A(R)$. $\mathcal{L}'(\langle x | (y', y) \rangle) = \mathcal{L}(\text{Tail}(\langle x | (y', y) \rangle))$ alors $\mathcal{L}'(\langle x | (y', y) \rangle) = \mathcal{L}(y')$. En conséquence, $\mathcal{L}'(\langle x | (y', y) \rangle) = \mathcal{L}(y)$ parce que soit $y' = y$ soit $\mathcal{L}(y') = \mathcal{L}(y)$ par la définition de blocage. Donc, on a $C \in \mathcal{L}'(\langle x | (y', y) \rangle)$.
- si y est un noeud blocable et un prédécesseur de $\text{Tail}(x)$ alors $x = \langle \langle p | (y, y) \rangle | (\text{Tail}(x), \text{Tail}'(x)) \rangle$ par la définition de chemin, où $\langle p | (y, y) \rangle$ est un chemin dans $\text{Paths}(\mathbf{G})$. $C \in \mathcal{L}(y)$ alors $C \in \mathcal{L}(\text{Tail}(\langle p | (y, y) \rangle))$. Par conséquent, $C \in \mathcal{L}'(\langle p | (y, y) \rangle)$. Par la définition de chemin, $\text{Tail}(x)$ est le successeur de y alors $\text{Tail}'(x) = \text{Tail}(x)$. Par conséquent, $\langle x, \langle p | (y, y) \rangle \rangle \in \mathcal{E}_A(R)$.
 - Si $x \in \text{Nom}(\mathbf{G})$ alors $\exists R.C \in \mathcal{L}(x)$. Le graph \mathbf{G} est complet alors il doit exister un R-successeur y de x avec $C \in \mathcal{L}(y)$.
- si y est un noeud nominal alors $\langle x, y \rangle \in \mathcal{E}_A(R)$ et $C \in \mathcal{L}'(y)$.
- si y est un noeud blocable alors $\langle x, \langle p | (y, y) \rangle \rangle \in \mathcal{E}_A(R)$ et $C \in \mathcal{L}(\text{Tail}(\langle p | (y, y) \rangle))$. Alors $C \in \mathcal{L}'(\langle p | (y, y) \rangle)$
- **Propriété 7** est prouvé comme pour la priorité 5.
- **Propriété 9, 10, 12, 13** sont les résultats de la construction du graphe \mathbf{G} .
- **Propriété 11, 18** sont satisfaites par la construction de graphe, par la définition de nominal et par l'état complet et clash-libre du \mathbf{G} .
- **Propriété 14** Supposons $\exists U.d \in \mathcal{L}'(p)$ avec $p \in \mathbf{S}_A$. Le graph \mathbf{G} est complet et clash-libre alors il doit exister un U-successeur x de $\text{Tail}(p)$ ou de p avec $d \in \mathcal{L}(x)$ et $\delta(x) \in \mathbf{S}_D$. Par conséquent, $\delta(x) \in d^D$.

– **Propriété 15** Supposons $\forall U.d \in \mathcal{L}'(p)$ et $\langle p, v \rangle \in \mathcal{E}_{\mathbf{D}}(U)$. Alors $v \in \mathbf{S}_{\mathbf{D}}$. Le graphe \mathbf{G} est complet et clash-libre alors $d \in \mathcal{L}(x)$ avec x est un noeud concret tel que $\delta(x) = v$. Par conséquent, $v \in d^{\mathbf{D}}$.

– **Propriété 17** est le résultat de la construction du graphe et de l'état complet et clash-libre du \mathbf{G} .

■

Lemme 6.3 (Complet) *Si D a un tableau pour une \mathcal{R} et \mathcal{K} alors les règles d'extension peuvent être appliquées sur D, \mathcal{R} et \mathcal{K} de telle manière qu'elles produisent un graphe complet et clash-libre pour D .*

Preuve. Soit $T = (\mathbf{S}_A, \mathbf{S}_{\mathbf{D}}, \mathcal{L}', \mathcal{E}_A, \mathcal{E}_D)$ pour D e.v.d \mathcal{R} et \mathcal{K} .

Pour des règles non-déterministes ($\sqcup-, \leq-, NN-$ et \mathcal{K}_+ -règle), on peut diriger le graphe de telle manière que chaque application de ces règles conserve l'état clash-libre du graphe. Avec la terminaison prouvé dans 6.1, alors on obtient un graphe complet.

Une fonction π qui associe chaque noeud abstrait dans le graphe \mathbf{G} à un élément de \mathbf{S}_A et chaque noeud concret à un élément de $\mathbf{S}_{\mathbf{D}}$ est construit au fur et à mesure de l'application des règles comme suit :

- Pour chaque noeud $x \in \mathbf{S}_A$, $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$
- Pour une paire de noeuds x, y et un rôle R , si y est un R -successeur de x alors $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_A(R)$ ou $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_{\mathbf{D}}(R)$
- $x \neq y$ implique $\pi(x) \neq \pi(y)$.
- $x \doteq y$ implique $\pi(x) = \pi(y)$.

De même, la manière de démontrer est semblable à celle pour *SHOIQ*. Ainsi nous concentrons sur les nouvelles caractéristiques de la contrainte d'identification. Le reste peut être trouvé dans (Horrocks *et al.*, 2001) et (Horrocks *et al.*, 2005). Nous pouvons appliquer les règles non déterministes (c.à.d $\sqcup-, \leq-, NN-$ et \mathcal{K}_+ -règles) de telle manière qu'elles maintiennent le graphe complet et clash-libre en employant une fonction qui associe des noeuds du graphe aux éléments dans le tableau le long du processus de construction de graphe. Grâce à la propriété 12, le clash de la forme (3) ne peut pas se produire. Grâce à la propriété 13, le clash de la forme (2) ne peut pas se produire. Les propriétés 17 et 18 garantissent que le clash de la forme (6) ne peut pas être généré.

■