Complexité des logiques de description

Jean-François Baget **Yannic Tognetti**

Février 1998

Introduction

"The Complexity of Concept Languages"
Donini, Lenzerini, Nardi et Nutt
[DLNN95], [DLNN91]

Une étude d'article pro(im)posée par Roland Ducournau dans le cadre du module "Objets" du DEA Informatique de l'université de Montpellier II.

(i) Logiques de descriptions

(ii) Algorithme de résolution

(iii) Analyse de complexité

Logiques de description

(i) ABox et TBox, syntaxe et sémantique

(ii) La famille de langages $\mathcal{AL}$

(iii) Raisonner sur une base terminologique

ABox et TBox, syntaxe et sémantique

Organisation des connaissances en deux parties distinctes :

- La Tbox (boite terminologique)
  Définit les concepts (classes) et les relations entre concepts. Composante intensionnelle.
- La ABox (boite d'assertions)
  Donnée des individus (extensions d'un concept, objets). Composante extensionnelle.

Place prépondérante de la TBox qui détermine les assertions possibles de la ABox. D'autres langages donnent la possibilité de décrire des concepts de la TBox à partir d'assertions de la ABox [Sch94].

Définir des concepts dans la TBox

On se donne un ensemble de concepts et de rôles primitifs.Des constructeurs permettent de construire des concepts (concepts définis) à partir d'autres concepts. Plusieurs types de définitions ou assertions possibles sur les concepts de la TBox.

- Spécification, Définition
- Inclusion, Equation

Nous ne nous intéresserons qu'à la définition de concepts, notée $A \doteq B$, et encore avec les restrictions suivantes :

- Un seul nom par concept
- Pas de cycle dans la définition

Pour ce deuxième cas, voir [Neb91], [DLNS97]

Constructeurs communs

- A est un concept atomique.
- $\top$ est le concept universel
- $\bot$ est le concept vide
- $\neg A$ est la négation atomique de concept
- $C \sqcap D$ est l'intersection de concepts
- $\forall R.C$ est la quantification universelle de rôle
- $\exists R.\top$ est la quantification existentielle restreinte de rôle

L'ensemble de ces constructeurs est celui qui définit le langage $\mathcal{AL}$

Autres constructeurs

- $C \sqcup D$ est l'union de concepts ($\mathcal{U}$)
- $\exists R.C$ est la quantification existentielle complète ($\mathcal{E}$)
- $\neg C$ est le complémentaire d'un concept ($\mathcal{C}$)
- $\geq nR$ et $\leq nR$ sont des restrictions numériques de rôles ($\mathcal{N}$)
- $S \sqcap T$ est l'intersection de rôles ($\mathcal{R}$)

En utilisant un sous-ensemble de ces constructeurs en sus des constructeurs définissant le langage $\mathcal{AL}$, on peut définir les langages $\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{C}][\mathcal{N}][\mathcal{R}]$.

Liste relativement exhaustive d'autres constructeurs pouvant être utilisés dans [Duc98]

Sémantique : interprétation d'un concept

Une interprétation d'un concept est la donnée d'un domaine $\Delta^{\mathcal{I}}$ et d'une fonction d'interprétation qui associe à tout concept un sous ensemble de $\Delta^{\mathcal{I}}$ et à tout rôle un sous ensemble de $R^{\mathcal{I}} \times R^{\mathcal{I}}$.

$\top^{\mathcal{I}}$	$\Delta^{\mathcal{I}}$
$\bot^{\mathcal{I}}$	$\emptyset$
$(C \sqcap D)^{\mathcal{I}}$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$(\neg A)^{\mathcal{I}}$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
$(\forall R.C)^{\mathcal{I}}$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b\ (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\ \}$
$(\exists R.\top =^{\mathcal{I}}$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b\ (a,b) \in R^{\mathcal{I}}\ \}$

Sémantique : interprétation d'un concept (suite)

$(C \sqcup D)^{\mathcal{I}}$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$(\neg C)^{\mathcal{I}}$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$(\exists R.C)^{\mathcal{I}}$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b\ (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\ \}$
$(\geq nR)^{\mathcal{I}}$	$\{a \in \Delta^{\mathcal{I}} \mid card\{b \mid (a,b) \in R^{\mathcal{I}}\ \} \geq n\}$
$(\leq nR)^{\mathcal{I}}$	$\{a \in \Delta^{\mathcal{I}} \mid card\{b \mid (a,b) \in R^{\mathcal{I}}\ \} \leq n\}$
$(R \sqcap Q)^{\mathcal{I}}$	$R^{\mathcal{I}} \cap Q^{\mathcal{I}}$

Rapport avec la logique du premier ordre

Identification des concepts des logiques terminologiques avec fragments de la logique du premier ordre. Concepts et rôles s'interprètent comme des prédicats unaires et binaires.

$\Phi_{\top}(x)$	true
$\Phi_{\bot}(x)$	false
$\Phi_{C \sqcap D}(x)$	$\Phi_C(x) \wedge \Phi_D(x)$
$\Phi_{\neg A}(x)$	$\neg \Phi_A(x)$
$\Phi_{\forall R.C}(x)$	$\forall y \; R(x,y) \rightarrow \Phi_C(y)$
$\Phi_{\exists R.\top}(x)$	$\exists y \; R(x,y)$

Rapport avec la logique du premier ordre (suite)

Pour certaines formules, on a besoin de rajouter l'inégalité à la logique du premier ordre : c'est tout de même moins compliqué que l'égalité.

$\Phi_{C \sqcup D}(x)$	$\Phi_C(x) \vee \Phi_D(x)$
$\Phi_{\neg C}(x)$	$\neg \Phi_C(x)$
$\Phi_{\exists R.C}(x)$	$\exists y\ R(x,y) \wedge \Phi_C(y)$
$\Phi_{\geq n R}(x)$	$\exists y_1, \ldots, \exists y_n\ (\bigwedge_{i \neq j} y_i \neq y_j) \wedge R(x,y_1) \wedge \ldots \wedge R(x,y_n)$
$\Phi_{\leq n R}(x)$	$\forall y_1, \ldots, \forall y_{n+1}\ R(x,y_1) \wedge \ldots \wedge R(x,y_{n+1} \rightarrow \bigvee_{i \neq j} y_i \doteq y_j$
$\Phi_{R \sqcap Q}(x,y)$	$R(x,y) \wedge Q(x,y)$

Question : Identifier les sous-ensemble de la logique du premier ordre que l'on traite avec ces langages.

La famille de langages $\mathcal{AL}$

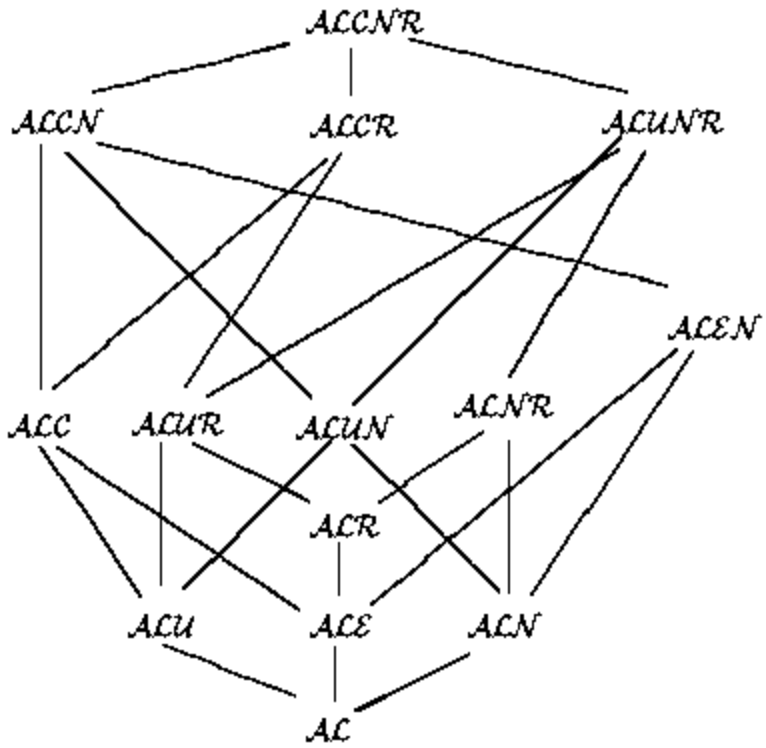
Possibilité de construire à partir de $\mathcal{AL}$, enrichi par les 5 constructeurs possibles : 32 langages différents.

Certains langages sont inclus dans d'autres, par exemple :

- $\mathcal{ALCUE} \equiv \mathcal{ALC}$
- $\mathcal{ALE} \subseteq \mathcal{ALR}$ A ce propos, Napoli [Nap97] affirme que la preuve se trouve dans cet article [DLNN91]. Mais il n'en est rien ! Nous avons donc dû en faire la preuve. Voir que l'on peut simuler $\exists R.C$ avec $(\exists(R \sqcap T).\top) \sqcap (\forall T.C)$ où T est une nouvelle relation.

Possibilité de construire un treillis ordonné par inclusion des 14 langages différents possibles (et non pas 16, grâce à la propriété "fantôme"). D'après R. Ducournau, nous avons "presque raison".

Treillis d'inclusion des langages $\mathcal{AL}$



Questions : Peut-on trouver d'autres inclusions, pour supprimer d'autres langages ? Et quelle est l'expressivité relative de ces langages ? [Bor96]

Raisonner sur une base terminologique

Avoir une base de connaissances, c'est bien. Pouvoir raisonner dessus, c'est mieux.

Quelques questions que l'on peut se poser :

- Satisfiabilité
- Subsomption
- Equivalence
- Incompatibilité
- Classification et instanciation [Nap97], [Duc98]

Seules les 4 premières formes de raisonnement sont traitées dans cet article.On ne raisonne que sur la TBox.

Définitions

- Un concept C est satisfiable si il existe une interprétation $\mathcal{I}$ (un modèle de C) telle que $C^{\mathcal{I}}$ soit non vide.
- Un concept C est subsumé par un concept D si pour toute interprétation $\mathcal{I}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- Deux concepts C et D sont équivalents si pour toute interprétation $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$
- Deux concepts C et D sont incompatibles si pour toute interprétation $\mathcal{I}$, $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$

Propriétés d'équivalence 1

- C est subsumé par D ssi $C \sqcap \neg D$ est insatisfiable
- C et D sont équivalents ssi $C \sqcap \neg D$ et $\neg C \sqcap D$ sont insatisfiables
- C et D sont incompatibles ssi $C \sqcap D$ est insatisfiable

Conclusion : Satisfiabilité suffit à résoudre les autres problèmes de décision pour les langages disposants du constructeur $\mathcal{C}$ (complément généralisé).

Propriétés d'équivalence 2

- C est insatisfiable ssi C est subsumé par $\bot$
- C et D sont équivalents ssi C et D se subsument l'un l'autre
- C et D sont incompatibles ssi $(C \sqcap D)$ est subsumé par $\bot$

Conclusion : Une borne superieure pour la complexité de la subsomption donne une borne superieure pour la complexité des autres problèmes.

Propriétés d'équivalence 3

Les assertions suivantes sont équivalentes :

(i) C est insatisfiable

(ii) C est subsumé par $\bot$

(iii) C et $\bot$ sont équivalents

(iv) C et $\top$ sont incompatibles

Conclusion : Une borne inferieure pour la complexité de l'insatisfiabilité donne une borne inferieure pour la complexité des autres problèmes.

Algorithme de résolution

(i) Présentation de l'algorithme

(ii) Exemples

(iii) Décidabilité, consistance et complétude

Présentation de l'algorithme

Algorithme générique vérifiant la satisfiabilité de concept pour tous les langages de la famille $\mathcal{AL}$

- Basé sur la méthode des tableaux [Smu68]
- Expansion des concepts en contraintes [HN90]
- Vérification de "clash" entre contraintes
- Ordre des instanciations successives donné par un arbre

Des concepts aux contraintes

Un système de contraintes S est un ensemble fini, non vide de contraintes de la forme :

- $x : C$ qui signifie que x est un individu appartenant à l'extension du concept C
- xPy qui signifie que x et y sont deux individus en relation par la relation primitive P
- $x \neq y$ qui signifie que x et y sont deux individus distincts

Interprétation de contraintes

Une fonction d'interprétation ($\mathcal{I}$-assignement) α associe à chaque variable un élément de $\Delta^{\mathcal{I}}$. On dit que α satisfait une contrainte :

- $x : C$ si $\alpha(x) \in C^{\mathcal{I}}$
- xRy si $(\alpha(x), \alpha(y)) \in R^{\mathcal{I}}$
- $x \neq y$ si $\alpha(x) \neq \alpha(y)$

Interprétation de contraintes (suite)

Quelques définitions utiles :

- Une contrainte c est satisfiable si il existe une interprétation $\mathcal{I}$ pour laquelle une fonction d'interprétation α satisfait c
- Une fonction d'interprétation α satisfait un système de contraintes S si α satisfait chaque contrainte de S.
- Un système de contraintes S est satisfiable si il existe une interprétation $\mathcal{I}$ pour laquelle une fonction d'interprétation α satisfait S.

"Clash" dans un système de contraintes

Un système de contraintes contient une contradiction ou "clash" si il contient un sous-ensemble de contraintes de l'une des formes suivantes :

- $\{x : \bot\}$ instanciation du concept vide
- $\{x : A, x : \neg A\}$ avec A concept primitif
- $\{x : (\leq nR), xRy_1, \ldots xRy_{n+1}\} \cup \{y_i \neq y_j \mid i, j \in 1 \ldots n+1, i \neq j\}$ signifie que x est en relation avec $n+1$ successeurs distincts alors qu'il en faut au plus n. Notons que R peut être un concept primitif ou défini.

La notion de successeur

Notion essentielle : c'est elle qui nous assure l'arrêt du calcul.

- Soit S un système de contraintes et R un rôle primitif ou défini. Alors y est un R-successeur de x si xRy est une contrainte de S
- On dit que y est un successeur de x si il existe un rôle R pour lequel y est un R-successeur de x.
- $S[y/z]$ est le système de contraintes obtenu en remplaçant toutes les occurences de y dans S par z
- Deux individus x et y sont séparés dans un système de contraintes S si S contient la contrainte $x \neq y$

Algorithme $SAT(S, x)$

Données: Un système de contraintes S sous forme normale négative, une variable x

Résultat: Un booléen valant **VRAI** si il existe une interprétation telle que x satisfait le système de contraintes S, i.e. si le concept associé à S est satisfiable, FAUX sinon

Remarques :

- Toute contrainte de la forme xRy où R est une relation définie par $R \doteq R_1 \sqcap \ldots \sqcap R_k$ sera immédiatement décomposée en un ensemble de contraintes $xR_1y, \ldots, xR_ky$
- L'ordre d'expansion des contraintes est d'une importance capitale. Pas simplement pour l'efficacité mais pour consistance.

début

cas où S *contient un "clash"* **retourner** $FAUX$;

cas où $S = S' \cup \{x : (C \sqcap D)\}$ **retourner** $SAT((S' \cup \{x : C, x : D\}), x)$;

cas où $S = S' \cup \{x : (C \sqcup D)\}$
| **retourner** $SAT((S' \cup \{x : C\}), x)$ **ou** $SAT((S' \cup \{x : D\}), x)$

cas où $S = S' \cup \{x : (\exists R.C)\}$ **et** x *n'a pas de R-succ. de type* C *dans* S'
| **retourner** $SAT((S' \cup \{xRy, y : C\}), x)$

cas où $S = S' \cup \{x : (\forall R.C)\}$ **et** y *est un R-succ. de* x *dans* S' *non typé par* C
| **retourner** $SAT((S \cup \{y : C\}), x)$

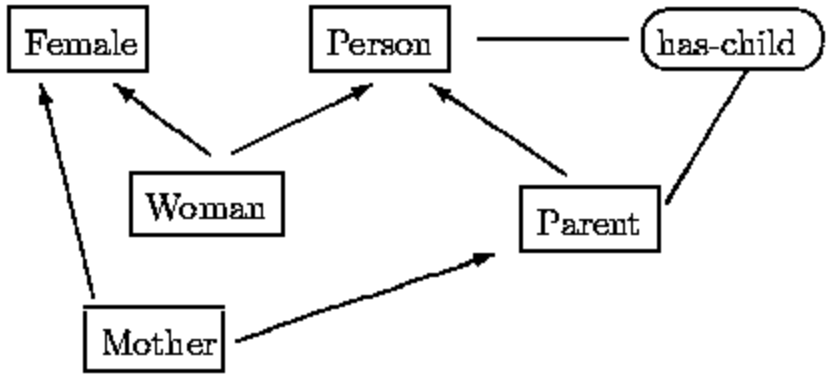
cas où $S = S' \cup \{x : (\geq nR)\}$ **et** $|\{R\text{-}succ_{S'}(x)\}| \leq n$ **et** y_i *nouvelles variables*
| **retourner** $SAT((S \cup \{xRy_1, \ldots, xRy_n\} \cup \{y_i \neq y_j / i, j \in [1, n], i \neq j\}), x)$

cas où $S = S' \cup \{x : (\leq nR)\}$ **et** $|\{R\text{-}succ_{S'}(x)\}| > n$
| **retourner** $SAT((S[y/z]), x)$ où y et z R-succ. de x non séparés dans S'

retourner $\wedge_{y_k \in succ(x)} SAT(S, y_k)$

fin

Exemples



Concepts et rôles primitifs : Female, Person, has-child

Concepts définis :

- Woman $\doteq$ Female $\sqcap$ Person
- Parent $\doteq$ Person $\sqcap \exists$ has-child.Person $\sqcap \forall$ has-child.Person
- Mother $\doteq$ Female $\sqcap$ Parent

Exemple tiré de : [DLNN95]

Un problème de subsomption

Intuitivement, on voit que Mother est plus spécifique que Woman. Nous allons en formaliser la preuve.

- $Woman$ subsume $Mother$
- $Mother \sqcap (\neg Woman)$ insatisfiable
- Expansion en concepts primitifs (possible car pas de cycle)

  $Female \sqcap Person \sqcap (\exists has\text{-}child.Person) \sqcap (\forall has\text{-}child.Person) \sqcap \neg(Female \sqcap Person)$

  insatisfiable
- Mise sous forme normale négative

  $Female \sqcap Person \sqcap (\exists has\text{-}child.Person) \sqcap (\forall has\text{-}child.Person) \sqcap (\neg Female \sqcup \neg Person)$

  insatisfiable
- Voir que le système de contraintes associé produit un “clash”

Expansion d'un système de contraintes

$\{x : (Female \sqcap Person \sqcap (\exists has\text{-}child.Person) \sqcap (\forall has\text{-}child.Person) \sqcap (\neg Female \sqcup \neg Person))\}$

Donne par utilisations répétées de la règle $\sqcap$

$\{x : Female, x : Person, x : (\exists has\text{-}child.Person), x : (\forall has\text{-}child.Person),$

$x : (\neg Female \sqcup \neg Person)\}$

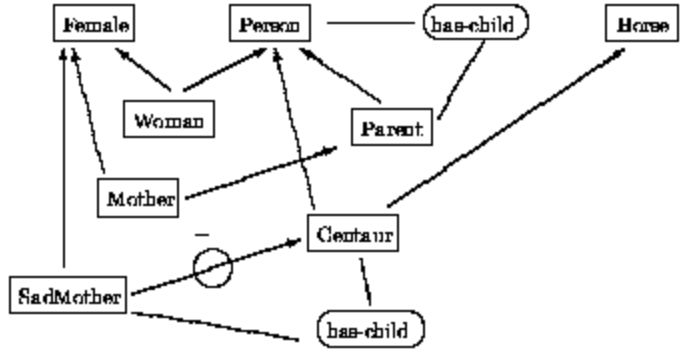
Maintenant, pas le choix, utilisation de la règle $\sqcup$

$\{x : Female, x : Person,$	$\{x : Female, x : Person,$
$x : (\exists has\text{-}child.Person),$	$x : (\exists has\text{-}child.Person),$
$x : (\forall has\text{-}child.Person), x : \neg Female\}$	$x : (\forall has\text{-}child.Person), x : \neg Person\}$
CLASH	CLASH

Création de deux systèmes de contraintes contenant tous deux un "clash", donc le concept est insatisfiable.

Un autre exemple

L'exemple précédent était un peu trop simple, on peut complexifier la TBox en y rajoutant quelques concepts. Les sources d'inspiration de cet exemple sont à rechercher du côté des cours de R. Ducournau.



Concept primitif : Horse

Concepts définis :

- Centaur $\doteq$ Person $\sqcap$ Horse
- SadMother $\doteq$ Female $\sqcap \neg$ Centaur $\sqcap (\exists$ has-child . Centaur)

Question : Est-ce-qu'une SadMother peut-être une Mother ?

Décidabilité, consistance et complétude

- Le calcul de $SAT(S, x)$ est consistant. Il faudra prouver que si le calcul s'arrête sur un ensemble de contraintes sans "clash", alors le concept associé au système de contraintes C est satisfiable.
- Le calcul de $SAT(S, x)$ est complet. Il faudra prouver que si le concept associé au système de contraintes S est satisfiable, alors le calcul s'arrêtera sur un ensemble de contraintes sans "clash"
- Pour prouver que les langages de la famille $\mathcal{AL}$ sont décidables, il faudra prouver que $SAT(S, x)$ s'arrête, que le concept représenté par le système de contraintes S soit satisfiable ou pas.

Consistance

Proposition 3.2 *(partie 1)* Si un système de contraintes $\{x : C\}$ est satisfiable, alors le concept C est satisfiable.

Démonstration par simple réécriture

Théorème 3.4 (Invariance) *(partie 1)* Si S' est un système de contraintes satisfiable généré par l'appel de $SAT(S, x)$, alors S est un système de contraintes satisfiable.

Preuve entièrement calquée sur celle de la consistance de la méthode des tableaux [HN95]

Proposition 3.6 Si S est un système de contraintes complètement expansé ne contenant pas de "clash", alors S est satisfiable.

Preuve en construisant une fonction d'interprétation pour une interprétation $\mathcal{I}$ sur un tel système de contraintes.

Complétude

Proposition 3.2 *(partie 2)* Si un concept C est satisfiable, alors le système de contraintes $\{x : C\}$ est satisfiable.

Démonstration par simple réécriture

Théorème 3.4 (Invariance) *(partie 2)* Soit S un système de contraintes satisfiable. Alors il existe un système de contraintes S', obtenu à partir de S en appliquant la règle d'expansion qui s'impose, tel que S' est satisfiable.

Preuve entièrement calquée sur celle de la consistance de la méthode des tableaux [HN95]

Théorème 3.5 Si un système de contraintes est satisfiable, alors il ne contient pas de "clash".

Contraposée, puis réécriture

Décidabilité

Soit $\mathcal{A}$ l'arbre représentant les appels récursifs issus de $SAT(S, x)$.

Proposition 3.9 (Terminaison) Aucune branche de $\mathcal{A}$ n'est de profondeur infinie.

Voir qu'on passe progressivement de contraintes portant sur des concepts définis à des contraintes portant sur des concepts primitifs, et à ce moment, il n'y a plus d'expansion possible [HN95]

Théorème 3.11 (Décidabilité) La satisfiabilité des concepts de $\mathcal{ALCNR}$ est un problème décidable.

Il faudra vérifier que l'arbre $\mathcal{A}$ ne possède qu'un nombre fini de branches. En effet, ce nombre de branches est borné par une fonction dépendant du nombre de règles non déterministes applicables et de la taille de l'arbre des successeurs, qui est fini.

Analyse de complexité

(i) Sources de complexité

(ii) Langages PSPACE-complets

(iii) Langages NP-complets ou co-NP-complets

Sources de complexité

Pour étudier la complexité de cet algorithme pour un concept appartenant à un langage de la famille $\mathcal{AL}$, il faut étudier les facteurs suivants :

- Complexité de la sélection et de l'application d'une règle à un système de contraintes.
- Complexité de la vérification de la présence de "clash" dans un système de contraintes.
- Nombre de systèmes de contraintes différents pouvant être générés ($\approx$ nombre de branches de l'arbre $\mathcal{A}$)
- Nombre de règles menant à l'expansion complète d'un système de contraintes ($\approx$ taille de la plus longue branche de l'arbre $\mathcal{A}$)

Application d'une règle

"Polynomial dans la taille d'un système de contraintes S"

Mais $x : (\geq nR)$ s'expanse en $\{xRy_1, \ldots, xRy_n\} \sqcup \{y_i \neq y_j\}_{i,j \in [1,n+1], i \neq j}$

Et ce n'est polynomial que si on a une représentation unaire des entiers. Et ce problème va se répercuter dans toutes les démonstrations.

Plutôt que d'accepter une telle représentation, nous proposons :

- Soit de lire *pseudo-polynomial* à la place de *polynomial* pour chaque résultat concernant un langage comprenant $\mathcal{N}$.
- Soit d'imposer à n d'être petit, ou même d'être exponentiellement petit devant la taille du système de contraintes qui le contient.

Vérification de "clash"

Trivialement polynomial pour les deux premiers cas de "clash". Mais pour le cas

$$\{x : (\leq nR), xRy_1, \ldots, xRy_{n+1}\} \cup \{y_i \neq y_j \mid i, j \in 1 \ldots n+1, i \neq j\}$$

Voir que c'est en général un problème NP-complet (réduction à recherche de clique de taille n dans un graphe).

Mais ce cas ne peut se produire que si il y a plus de n R-successeurs de x dans S, et ceci suffit à prouver le "clash" lorsque plus aucune expansion n'est possible (sinon, on aurait pu utiliser la dernière règle).

Conclusion : Pour rester polynomial, on ne testera ce troisième cas de "clash" que sur les feuilles de l'arbre $\mathcal{A}$, i.e. après toutes les expansions possibles.

Nombre de branches de l'arbre $\mathcal{A}$

Les seules règles qui induisent l'exploration de plusieurs systèmes de contraintes sont celles associées aux contraintes $x : (C \sqcup D)$ et $x : (\leq nR)$ qui contient implicitement une disjonction.

Ces règles, que l'on appellera règles non déterministes, peuvent donc générer un nombre exponentiel de systèmes de contraintes.

Les deux règles non déterministes sont notre première source de complexité.

Taille des branches de l'arbre $\mathcal{A}$

On peut classer les règles d'expansion suivant deux types différents :

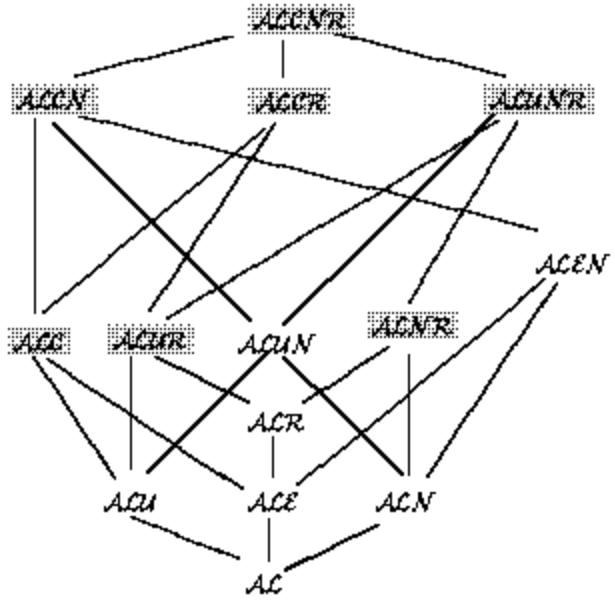
- Règles non génératrices
  Ce sont celles qui n'introduisent pas de nouvelles variables. Elles ne font que créer des sous-concepts, et un concept ne peut avoir qu'un nombre de sous-concepts linéaire dans sa taille.
- Règles génératrices
  Ce sont les règles applicables aux contraintes $x : (\exists R.C)$ ou $x : (\geq nR)$ qui introduisent de nouvelles variables. Voir que ces règles peuvent produire un nombre exponentiel de nouvelles variables. Par exemple la contrainte de taille n : $x :$
  $(\exists R_1.C_{11} \sqcap \exists R_1.C_{12} \sqcap \forall R_1.(\exists R_2.C_{21} \sqcap \exists R_2.C_{22} \sqcap \forall R_2.(\ldots(\exists R_n.C_{n1} \sqcap \exists R_n.C_{n2} \sqcap \forall R_n.D)\ldots)))$
  s'expanse en un système de contraintes contenant 2^n nouvelles variables

Les deux règles génératrices sont notre deuxième source de complexité.

Langages PSPACE-complets

Voir tout d'abord que le calcul de satisfiabilité (ou de subsomption) dans $\mathcal{ALCNR}$ est dans PSPACE (et encore, avec hypothèse de représentation unaire des entiers).

Voir ensuite que l'introduction du constructeur $\mathcal{C}$, celle de $\mathcal{UR}$ ou celle de $\mathcal{NR}$ suffisent à rendre le calcul PSPACE-complet.



$\mathcal{ALCNR}$ est dans PSPACE

Proposition 4.4 Soit C un concept quelconque. Alors le calcul de $SAT(x : C, x)$ a une complexité polynomiale en espace.

Tous les systèmes de contraintes générés ont une taille bornée polynomialement (toujours avec assertion de la représentation unaire).

Lors des appels récursifs de SAT, on n'a besoin de garder dans la pile qu'un nombre polynomial de ces appels.

Conclusion : Tous les langages de la famille $\mathcal{AL}$ sont dans PSPACE.

$\mathcal{ALC}$ est PSPACE-complet

Théorème 4.7 ($\mathcal{ALC}$-Réduction) La satisfiabilité d'un concept de $\mathcal{ALC}$ est un problème PSPACE-complet.

Preuve par réduction de Quantified Boolean Formulas à $\mathcal{ALC}$-SAT.

Quantified Boolean Formulas

Données: Une formule $\mathcal{F} = Q_1 \dots Q_n(\mathcal{B})$ où les Q_i sont des quantificateurs $\forall$ ou $\exists$ et $\mathcal{B}$ est une expression booléenne.

Résultat: Est-ce-que $\mathcal{B}$ est une tautologie ?

PSPACE-complet [SM73]

Réduction utilisée

Quitte à renommer les variables, on peut écrire une QBF sous la forme :

$\mathcal{F} = Q_1 x_1 \ldots Q_n x_n (C_1 \wedge \ldots \wedge C_m)$

où les C_i sont des clauses de la forme $K_1 x_1 \vee \ldots \vee K_n x_n$ avec K_i symbole logique pouvant être inexistant (la variable x_i est positive), $\neg$ (la variable x_i est négative), ou $\emptyset$ si la variable x_i ne se trouve pas dans la clause.

Construisons le concept $C = \Phi(Q_1 x_1 \ldots Q_n x_n) \sqcap \Psi(C_1) \sqcap \ldots \sqcap \Psi(C_m)$.

Voir que la transformation est polynomiale et que $\mathcal{F}$ est valide si et seulement si C est satisfiable.

Réduction utilisée (suite)

- Transformation des quantificateurs

  $\Phi(\forall x_n) = (\exists R.A) \sqcap (\exists R.\neg A)$

  $\Phi(\exists x_n) = (\exists R.\top)$

  $\Phi(\forall x_i(\mathcal{Q})) = (\exists R.A) \sqcap (\exists R.\neg A) \sqcap (\forall R.\Phi(\mathcal{Q}))$

  $\Phi(\exists x_i(\mathcal{Q})) = (\exists R.\top) \sqcap ((\forall R.\Phi(\mathcal{Q}))$
- Transformation d'une clause $K_1 x_1 \vee \ldots \vee K_n x_n$

  Soit p l'indice tel que $K_p \neq \emptyset$ et $\forall q > p, K_q = \emptyset$

  $\Psi(x_p) = \forall R.A$

  $\Psi(\neg x_p) = \forall R.\neg A$

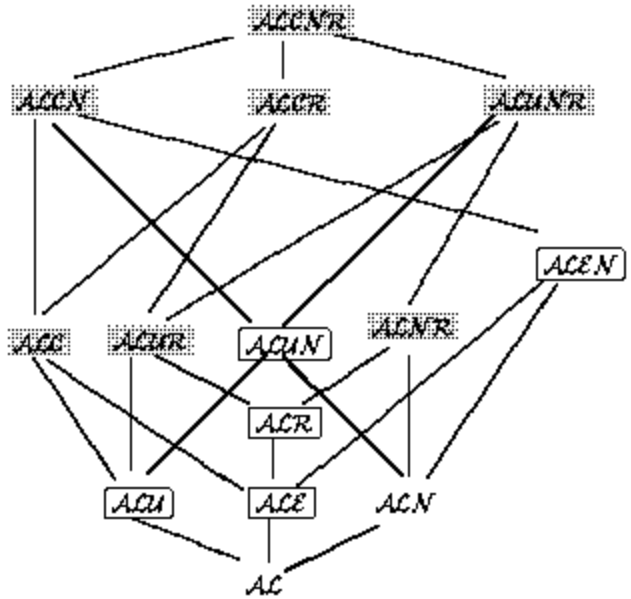
  $\Psi(x_i \vee C_i) = \forall R.(A \sqcup \Psi(C_i))$

  $\Psi(\neg x_i \vee C_i) = \forall R.(\neg A \sqcup \Psi(C_i))$

  $\Psi(\emptyset x_i \vee C_i) = \forall R.(\Psi(C_i))$

Langages NP-complets et co-NP-complets

Nous avons vu que nos deux sources de complexité étaient les règles génératrices, et les règles non déterministes.



Nous verrons quelles sont les règles dont il faut se passer si on veut ne pas être PSPACE-complet pour le problème **Insatisfiabilité**.

Langages NP-complets

Le plus grand langage dans lequel on ne trouve aucune règle non déterministe est $\mathcal{ALER}$ (pour nous $\mathcal{ALR}$).

Intuitivement, il appartient à NP car on peut exhiber un cerificat polynomial. En effet, l'arbre des appels récursifs de SAT ne contient comme seuls noeuds que ceux induits par le constructeur $\forall$. Si on veut prouver qu'un concept est insatisfiable, il suffit d'exhiber une seule de ses branches fermées — et on a déjà vu qu'elles étaient de longueur polynomiale.

Preuve de NP-complétude par réduction dans [DHL+92]

Langages co-NP-complets

Définition : Un problème de décision $\Pi : x \in L?$ est dans co-NP si et seulement si le problème $\Pi' : x \notin L?$ est dans NP.

Voir que $\mathcal{ALU}$ est co-NP-complet.

Intuitivement, cette fois-ci, pour exhiber un certificat polynomial prouvant qu'un concept est insatisfiable, il faut prouver que tous les systèmes de contraintes que l'on peut générer par des $\sqcup$ sont insatisfiables — et il peut y en avoir un nombre exponentiel. Par contre, pour prouver que ce concept est satisfiable, il suffira d'exhiber un seul système de contraintes satisfiables

Preuve par réduction dans [SSS91], à partir de formules de la logique des propositions.

Conclusion

Interêt :

- Approche assez complète sur les parties décidables de la logique du premier ordre.
- Algorithme générique qui est "dans la bonne classe de complexité" pour tous les langages étudiés.
- Compilation, completion et unification sous la vision "système de contraintes" des résultats de complexité sur cette famille de langages.
- Article de référence pour qui désire "étendre" un langage quelconque de représentation des connaissances (Bases de données, Langages orientés objets, graphes conceptuels ...)

Inconvénients :

- Multiplicité des systèmes formels équivalents (clauses, concepts, contraintes) qui obligent à fournir un énorme travail d'assimilation avant de commencer à comprendre de quoi parle l'article.
- Preuves importantes souvent éparpillées dans l'ensemble de l'article, ou renvoyées en biblio. Nous avons essayé de les simplifier.
- Ignorance totale de la ABox. On fait de la représentation des connaissances, mais pourquoi ? On a l'impression que l'article aurait pu s'intituler "Complexité des parties décidables de la logique du premier ordre". La fonction d'interprétation ne va pas voir ce qui se passe du côté des instances, on ne parle ni d'instanciation, ni de classification.
- On reste sur notre faim en ce qui concerne l'expressivité de ces différents langages.

Références

[BDNS95] Martin Buchheit, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. A refined architecture for terminological systems : Terminology = schema + views. Technical report, Deutsches Forchungszentrum für Künstliche Intelligenz, Saarbrücken, 1995.

[BDS93] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1993.

[Bor96] Alex Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 1996.

[DH89] Roland Ducournau and Michel Habib. La multiplicité de l'héritage dans les langages à objets. *Technique et science informatiques*, 1989.

[DHH+95] Roland Ducournau, Michel Habib, Marianne Huchard, Marie-Laure Mugnier, and Amedeo Napoli. Le point sur l'héritage multiple. *Technique et science informatiques*, 1995.

[DHL+92] Francesco M. Donini, Benhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamella, Daniele Nardi, and Werner Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 1992.

[DLNN91] Francesco M. Donini, Morizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proceedings of KR'91*, Cambridge (MA), 1991.

[DLNN95] Francesco M. Donini, Morizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, 1995.

[DLNS97] Francesco M. Donini, Morizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In *A Great*

Collection. U. Gnowho and U. Gnowho-Else, eds., 1997.

[Duc98] Roland Ducournau. Des langages à objets aux logiques terminologiques : les systèmes classificatoires. Technical report, LIRMM, Montpellier, 1998.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability : a guide to the Theory of NP- Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[HN90] Bernhard Hollunder and Werner Nutt. Subsumption algorithms for concept languages. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslauten, 1990.

[HN95] Bernhard Hollunder and Werner Nutt. Algorithms for concept languages. Technical report, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, 1995.

[Joh90] David S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[MC96] Marie-Laure Mugnier and Michel Chein. Représenter des connaissances et raisonner avec des graphes. *Revue d'intelligence artificielle*, 1996.

[Nap97] Amedeo Napoli. Une introduction élémentaire aux logiques de descriptions. Technical report, INRIA, 1997.

[Neb90] Bernhard Nebel. Reasoning and revision in hybrid representation systems. In *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1990.

[Neb91] Benhard Nebel. Terminological cycles : Semantics and computational properties. In *Principles of Semantic Networks : Explorations in the Representation of Knowledge*, San Mateo (CA), 1991. Morgan Kaufmann Publishers.

[Sch91] Klaus Schild. A correspondance theory for terminological logics. Technical report, Technische Universität, Berlin, 1991.

[Sch94] Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 1994.

[SM73] L. J. Stockmeyer and A. L. Meyer. Word problems requiring exponential time. In *Proc. 5th Ann. ACM Symp. on Theory of Computing*, New-York, 1973. Association for Computing Machinery.

[Smu68] Raymond M. Smullyan. *First-Order Logic*. Springer Verlag, Berlin, 1968.

[Sow84] John F. Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addisson-Wesley, Reading (MA), 1984.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 1991.

□