LRC - TME n°5

Méthode des tableaux pour $\mathcal{ALC}$ avec Lotrec

Le but de ce TME est de programmer une nouvelle logique dans Lotrec pour pouvoir faire de la vérification de cohérence de Abox selon une Tbox en $\mathcal{ALC}$.

La façon dont nous proposons de réaliser cette implémentation est inspirée des logiques prédéfinies `Classical-Propositional-Logic` et `S5-explicit-edges`.

Après avoir défini les connexteurs, on procède par étape, en programmant d'abord le cas d'une Abox sous forme normale négative (NNF) sans Tbox [Exercice 2], puis on ajoute progressivement des règles et stratégies pour permettre une mise sous forme NNF dynamique [Exercice 3], avant de permettre la prise en compte d'une Tbox simple [Exercice 4], puis d'une Tbox générale [Exercice 5]. On suppose que les Tbox utilisées sont acycliques et on ne traite pas le cas des Tbox définitoires : les Tbox ne pourront contenir que des axiomes d'inclusion simple ou générale.

Pensez à sauvegarder régulièrement. Vous pouvez utiliser l'interface de Lotrec, ou éditer le fichier xml de votre logique.

Exercice 1 Définition des connecteurs

Au lancement de Lotrec, choisir dans la section `Create your own` l'option `New Logic File` : les onglets `Connectors`, `Rules` et `Strategy` sont donc vides et l'objectif du TME est de les remplir progressivement.

Définir d'abord les connecteurs, qui seront communs à tous les exercices, en les ajoutant par `Add` à l'onglet `Connectors` et en utilisant les noms suivants

Expressions de concepts	
$\neg C$	`not C`
$C \sqcap D$	`inter C D`
$C \sqcup D$	`union C D`
$\forall R.C$	`forall R C`
$\exists R.C$	`exists R C`

Assertions	
$a \sqsubseteq C$	`inst A C`
$a, b \sqsubseteq C$	`instR A B C`

Axiomes ontologiques	
$C \sqsubseteq D$	`incl C D`

Remarque : pour la partie `Display`, il est possible de faire un copier-coller des symboles depuis le pdf, en utilisant _ pour les variables, par exemple _ ⊓ _ pour la conjonction.

Exercice 2 Vérification de cohérence d'une Abox sous NNF avec Tbox vide

Le but de cet exercice est de définir une première logique simple ne prenant en compte qu'une Abox déjà mise sous forme normale négative (NNF), sans Tbox associée. Il s'agit donc juste de déployer les règles des différents opérateurs et à les ordonner dans une stratégie.

1.1 – Règles d'initialisations (multiples)

On appelle règle d'initialisation l'équivalent de la règle `ExampleOfModelAndFormula` vue dans un précédent TME. Cette règle doit donc ajouter toutes les assertions de la ABox considérée, par des instructions du type `add w inst Eon not Homme` où `w` désigne le nœud courant et où on écrit ensuite les formules considérées selon les connecteurs définis dans l'exercice précédent.

De plus, cette règle doit commencer par marquer le nœud comme ABox, par l'instruction `mark w ABox`.

Ecrire deux règles de ce type, pour représenter les deux ABox suivantes : la première tirée de l'exemple 2 du TD permet de tester les règles simples, tandis que la deuxième est un nouvel exemple abstrait destiné à tester plus précisément les difficultés liées à la règle $R_\exists$.

$\mathcal{A}_1$:

elisabeth : Femme $\sqcap \forall$amant.Homme
eon : $\neg$Homme $\sqcup$ Travesti
elisabeth, eon : amant

$\mathcal{A}_2$:

a : C $\sqcap \forall$R1.(C $\sqcap$ D) $\sqcap \exists$R1.D
b : $\neg$C $\sqcup$ F
a, b : R1 c : $\exists$R2.$\exists$R1.$\exists$R2.(C $\sqcap \neg$C)

NB Afin de ne pas avoir besoin de modifier la stratégie quand on veut changer l'exemple, et donc la règle d'initialisation choisie, on peut mettre une condition à chaque règle d'initialisation, du type `hasElement w EX1` (respectivement `EX2`) et mettre toutes les règles d'initialisation dans la stratégie. L'utilisateur doit alors dans la partie `Compose your own formula` écrire `EX1` ou `EX2` pour que la stratégie déclenche la règle correspondante.

1.2 – Génération de nouvelles instances

Chaque application de la règle $R_\exists$ demande la création d'une nouvelle instance. On ne peut pas, en Lotrec, utiliser un compteur entier, pour créer successivement `I1`, `I2`, `I3` par exemple. On choisit à la place la notation `sI`, `s(sI)`, `s(s(I))` etc. Il faut de plus savoir quelles sont les instances déjà créées et quelle est la suivante qu'on peut créer.

Pour cela, on crée d'abord un nouvel opérateur unaire `nv` qui, appliqué à une variable `I`, s'affiche sous la forme `sI` (suivant de `I`).

On crée ensuite un nœud, qu'on nommera `v`, lié au nœud `w` qui contient les assertions étudiées, dont l'objectif est de permettre de voir les instances déjà créées. La règle d'initialisation `varBox` de ce "nœud des variables" `v` a pour condition que le nœud `w` soit marqué comme étant une Abox, et pour action de créér un nouveau nœud `v`, de le relier à `w` par une relation `Var` et d'ajouter dans ce nouveau nœud la formule `nv I`.

Quand une règle devra utiliser une nouvelle instance (typiquement la règle $R_\exists$), elle pourra récupérer la dernière instance libre `nv` I en vérifiant si la boîte de variable contient `nv` I sans contenir `nv nv` I. La règle pourra alors utiliser `nv` I comme nouvelle variable, et générera la variable suivante (`nv nv` I) dans la boite de variables.

1.3 – Déroulement du tableau

Définir toutes les règles du tableau en s'inspirant des règles de `Classical-Propositional-Logic`, en pensant à inclure des règles pour s'arrêter en cas de conflit ($a : \bot$ ou bien $a : C$ et $a : \neg C$).

On rappelle ci-dessous les règles de l'algorithme tableau pour $\mathcal{ALC}$.

Règles conjonctives

Règle	**condition**	**action**
$R_\sqcap$	$a : C \sqcap D$	ajout de $a : C$ et de $a : D$
$R_\forall$	$a : \forall R.C$ et $a, e : R$	ajout de $e : C$
$R_\exists$	$a : \exists R.C$ et aucun e	ajout de $a, b : R$ et de $b : C$
	tel que $e : C$ et $a, e : R$	où b nouvelle constante

Règle disjonctive

Règle	**condition**	**action**
$R_\sqcup$	$a : C \sqcup D$	Duplication en $\mathcal{A}_g$, $\mathcal{A}_d$
		ajout de $a : C$ à $\mathcal{A}_g$ et de $a : D$ à $\mathcal{A}_d$

Pour gérer $R_\exists$, il y a deux difficultés :

- Générer un identifiant pour la nouvelle instance (voir question précédente).
- La condition "aucun e tel que $e : C$ et $a, e : R$" n'est pas directement expressible en Lotrec. Elle sert à éviter de générer un individu s'il y en a déjà un qui convient. C'est le même principe que pour la règle `Pos` de la logique `S5-explicit-edges` et on va donc utiliser la même technique : avoir une règle qui marque l'assertion $a : \exists R.C$ comme `[Done]` si c'est le cas, et qui est toujours déclenchée avant la règle $R_\exists$).

  Note, on se contentera de vérifier s'il y a une instance e telle que $e : C$ et $a, e : R$ sont explicitement dans la Abox, sans chercher à vérifier si $e : C$ serait entrainé logiquement par la Abox (si par exemple on avait $C = C1 \sqcap C2$ avec $e : C1$ et $e : C2$).

1.4 – Stratégies

Pour finir, écrire une stratégie pour la vérification de cohérence et tester que cela fonctionne. Il peut être pertinent de créer des sous-stratégies pour obtenir l'arbre le plus simple possible. En particulier, il est bon de vérifier dès que possible les conditions d'arrêt et d'appliquer autant que possible les règles $R_\sqcap$ et $R_\forall$ avant de gérer la règle $R_\exists$ et de ne déclencher la règle $R_\sqcup$ qu'une fois toutes les autres règles déclenchées.

Exercice 3 Mise sous NNF dynamique

Etendre la logique définie précédemment avec un nouveau lot de règles pour gérer des Abox non mises sous forme normale.

On définit pour cela une nouvelle stratégie incluant un nouveau jeu de règles traduisant les principes de réécriture, que l'on rappelle ici :

$$\begin{array}{rcl} \neg(\neg C) & \leftrightarrow & C \\ \neg(C \sqcap D) & \leftrightarrow & \neg C \sqcup \neg D \\ \neg(C \sqcup D) & \leftrightarrow & \neg C \sqcap \neg D \end{array} \qquad \begin{array}{rcl} \neg(\forall R.C) & \leftrightarrow & \exists R.\neg C \\ \neg(\exists R.C) & \leftrightarrow & \forall R.\neg C \end{array}$$

On note qu'il est difficile en Lotrec de mettre la Abox complètement sous forme normale dès le départ car chaque règle s'applique sur la Abox complète, dans laquelle les formules sont imbriquées dans des assertions. La solution est de faire cela dynamiquement en ne traitant les négations que lorsqu'elles apparaissent tête d'une assertion (c'est-à-dire quand la boite comprend une assertion $a : \neg(E)$ où E est un concept complexe).

Pour tester ces nouvelles règles, on crée une nouvelle règle d'initialisation conditionnée par `EX2` :

$\mathcal{A}_3$:

$\mathtt{a} : \mathtt{C} \sqcap \neg(\exists \mathtt{R1}.(\neg \mathtt{C} \sqcup \neg \mathtt{D}) \sqcup \forall \mathtt{R1}.\neg \mathtt{D})$
$\mathtt{b} : \neg \mathtt{C} \sqcup \mathtt{F}$
$\mathtt{a}, \mathtt{b} : \mathtt{R1}\ \mathtt{c} : \neg \forall \mathtt{R2}.\forall \mathtt{R1}.\forall \mathtt{R2}.\top$

Exercice 4 Prise en compte d'une Tbox simple

Etendre la logique précédente de façon à pouvoir prendre en compte une Tbox simple, c'est-à-dire une Tbox ne contenant que des inclusions de la forme $C \sqsubseteq E$ où C est un concept atomique.

On rappelle que pour prendre en compte une telle Tbox, il faut *déployer les axiomes d'inclusion simples* comme suit : Pour chaque triplet (a, C, D) (avec a constante, C concept atomique et D expression de concept) tel que la Abox contienne l'assertion $a : C$ et que la Tbox contienne l'inclusion $C \sqsubseteq D$, on **ajoute** (si non présent) l'assertion $a : D$. On itère tant que cela génère des ajouts.

Ici, il est nécessaire, après le déploiement initial, de refaire un déploiement *chaque fois qu'on ajoute une assertion $a : C$ où C concept est un concept atomique apparaissant à gauche d'une inclusion.*

Pour représenter la Tbox, la méthode est de créer un nouveau nœud lors de l'initialisation, de le marquer `[Tbox]` et de le relier à la Abox par la relation `TboxDe`.

Pour tester l'algorithme on utilisera l'exemple de l'exercice 4 de la feuille de TD, rappelée ci-dessous (utilisant la condition `EX3` pour sa règle d'initialisation) :

$\mathcal{T}_4$:

$\mathtt{Herbivore} \sqsubseteq \exists \mathtt{mange}.\mathtt{Plante}$
$\mathtt{Carnivore} \sqsubseteq \exists \mathtt{mange}.\mathtt{Animal}$
$\mathtt{Sauvage} \sqsubseteq \mathtt{Animal} \sqcup \mathtt{Plante}$

$\mathcal{A}_4$: $e : \exists \mathtt{mange}.\mathtt{Sauvage} \sqcap \neg(\mathtt{Herbivore} \sqcup \mathtt{Carnivore})$

Pour récapituler, afin d'étendre la logique, il faut définir des règles pour l'initialisation et le déploiement et une stratégie qui les incluent correctement.

Exercice 5 Prise en compte d'une Tbox générale

Proposer une dernière extension de la logique précédente de façon à pouvoir prendre en compte une Tbox générale, c'est-à-dire une Tbox pouvant contenir des inclusions de la forme $E_1 \sqsubseteq E_2$ où E_1 est un concept non atomique. Cette logique devra contenir une règle d'initialisation pour l'exemple de l'exercice 5, rappelé ci-dessous :

$\mathcal{T}_5$:

$\forall \mathtt{mange}.\mathtt{Animal} \sqsubseteq \mathtt{CarnivoreStrict}$
$\neg \mathtt{Plante} \sqcap \neg \mathtt{Animal} \sqsubseteq \top$

$\mathcal{A}_5$: $e : \neg \exists \mathtt{mange}.\mathtt{Plante} \sqcap \neg \mathtt{CarnivoreStrict}$

Prise en compte d'une Tbox générale :

- *Réécriture des axiomes généraux.* Chaque axiome d'inclusion générale $C \sqsubseteq D$ est remplacé par sa réécriture $\top \sqsubseteq \neg C \sqcup D$. On met ensuite la Tbox sous NNF.
- *Déploiement des axiomes généraux* :
  Pour chaque constante a apparaissant dans la Abox, on **ajoute** (si non présent) l'assertion $a : \top$, puis on effectue un déploiement des axiomes $\top \subseteq E$ comme dans le cas d'une Tbox simple.
- Ici, il est nécessaire, après le déploiement initial, de refaire un déploiement *chaque fois qu'on ajoute une nouvelle constante (règle $R_\exists$).*