Sujet : Implémentation d'une logique de descriptions probabiliste pour l'évaluation de la qualité de service de processus métiers

M1 Informatique, Parcours : Données, Connaissances et Intelligence

Auteurs :

Ngo Alexandre

Hammaz Massinissa

Encadrants :

Ouziri Mourad

Benbernou Salima

Année universitaire : 2023-2024

Sommaire

I) Introduction et contexte :

Dans une architecture orientée services, les fournisseurs se doivent de respecter la qualité des services qu'ils proposent. Cependant celles-ci s'avèrent difficile à respecter indéfiniment car n'étant pas automatisée. Ces fournisseurs auraient donc besoin d'un système capable de gérer en temps réel la qualité de service (QoS). Notre projet vise à développer une telle solution, où les services sont représentés en utilisant la logique des descriptions. Le client fixe les attentes en termes de qualité et de performance, et notre programme agit comme un raisonneur intelligent, évaluant ces attentes pour sélectionner uniquement les services qui répondent aux critères définis. Ainsi, nous garantissons que les services fournis respectent les exigences du client, assurant une performance optimale tout en s'adaptant aux variations possibles des attributs des services.

Pour cela, nous allons implémenter une approche basée sur la logique des descriptions probabiliste qui est une famille de langages formels utilisés pour représenter les connaissances structurées d'un domaine de manière précise et formelle. Celle-ci comporte des Concepts, des Rôles et des Individus sur lesquelles nous travaillons.

L'objectif principal de ce projet est d'implémenter un raisonneur capable d'effectuer des inférences en temps réel pour garantir que les services respectent les qualités de service attendues par l'utilisateur. Un raisonneur est un système logiciel conçu pour déduire de nouvelles informations à partir d'une base de connaissances en utilisant des règles logiques. Il permet de vérifier la cohérence des données, de classifier les informations, et de répondre à des requêtes complexes en se basant sur les relations et les contraintes définies dans la base de connaissances.

Concrètement, le système que nous développons permettra de sélectionner dynamiquement les services en fonction des exigences spécifiées dans les accords de niveau de services ANS, et de stopper les services ne répondant plus aux attentes, tout en prenant en compte les variations potentielles des exigences en temps réel. Pour mener à bien ce projet, nous commencerons par la gestion des services et des attentes en termes de qualité fournies par le client. Ensuite, dans un second temps, nous utiliserons le langage Java pour implémenter le raisonneur, en nous appuyant sur l'algorithme tableau pour effectuer les inférences nécessaires. Ce processus nous permettra de développer un système robuste capable de sélectionner dynamiquement les services répondant aux attentes spécifiées par le client.

En somme, ce projet vise à proposer une solution pour la gestion dynamique de la QoS dans les architectures orienté services, en combinant la logique des descriptions avec des concepts probabilistes telle que suggère l'approche innovante conçu à cet effet qu'ont présenté Salima Benbernou, Allel Hadjali, Naouel Karam et Mourad Ouziri.

II) Etat de l'art:

Dans cette partie, nous allons voir quelles sont les méthodes existantes afin de traiter ce sujet :

A) Rappelle sur les logiques des descriptions :

Les logiques de description (LD) sont une famille de formalismes de représentation des connaissances, conçues pour représenter et raisonner sur des connaissances terminologiques. Dans les LD, les connaissances conceptuelles d'un domaine d'application sont représentées en termes de concepts (prédicats unaires) désignant des ensembles d'individus et de rôles (prédicats binaires) désignant des relations binaires entre individus.

En partant de l'ensemble NC des noms de concepts et de l'ensemble NR des noms de rôles, des descriptions de concepts complexes sont construites de manière inductive à l'aide de constructeurs de concepts. Les différents langages de logiques de description se distinguent par les types de constructeurs qu'ils autorisent. Dans notre cadre, nous utilisons la logique de description ALU, dans laquelle les descriptions de concepts sont formées selon les règles de syntaxe décrites dans la deuxième colonne du Tableau 1.

Un élément $A \in NC$ désigne un nom de concept, $r \in NR$ un nom de rôle, et C, D des descriptions de concepts. Une description de concept construite à l'aide des constructeurs d'une logique de description L est appelée une description de concept L.

La sémantique d'une description de concept est définie par la notion d'interprétation. Une interprétation $I=(\Delta I, \cdot I)$ consiste en un ensemble non vide $\Delta I \Delta$I, le domaine de l'interprétation, et une fonction d'interprétation $\cdot I$ qui associe à chaque nom de concept $A \in NC$ un sous-ensemble de ΔI et à chaque nom de rôle $r \in NR$ une relation binaire $rI \subseteq \Delta I \times \Delta I$. La fonction d'interprétation peut être étendue à des descriptions de concepts arbitraires comme illustré dans la troisième colonne du Tableau 1.

Les connaissances d'un domaine d'application sont décrites dans une Base de Connaissances (BC) formée de deux composantes : l'une intentionnelle, appelée TBox, et l'autre extensionnelle, appelée ABox.

Une TBox T est un ensemble fini d'axiomes terminologiques de la forme $A \equiv C$ et $A \sqsubseteq C$ tels qu'aucun nom de concept n'apparaît plus d'une fois sur le côté gauche de la définition.

Une interprétation I est un modèle d'une TBox T si elle satisfait :

$AI=CI$ pour tous les axiomes terminologiques $A \equiv C$ dans T,

$AI \subseteq CI$ pour tous les axiomes terminologiques $A \sqsubseteq C$dans T.

Une interprétation I est un modèle d'une TBox T si elle satisfait chaque axiome terminologique de T.

Une ABox A est un ensemble d'assertions individuelles de la forme $C(a)$ pour les concepts et $r(a,b)$ pour les rôles. Une interprétation I est un modèle d'une ABox A si elle satisfait :

$aI \in CI$ pour toutes les assertions $C(a)$ dans A,

$(aI,bI) \in rI$ pour toutes les assertions $r(a,b)$ dans A.

Une interprétation I est un modèle d'une ABox A si elle satisfait chaque assertion dans A.

Les logiques de description sont assorties de diverses tâches de raisonnement. Deux sont d'intérêt dans notre contexte :

Cohérence : vérifier si une base de connaissances BC=⟨TBox,ABox⟩ est satisfaisable, c'est-à-dire si elle a un modèle, noté BC⊨⟨TBox,ABox⟩.

Vérification d'instance : vérifier si l'assertion $C(a)$ est satisfaite dans chaque modèle de BC, noté BC⊨⟨TBox,ABox⟩$C(a)$.

B) Logiques de Description Probabilistes :

Introduction

La logique des descriptions probabilistes (PLD) combine les caractéristiques des logiques descriptives (DL) avec les capacités de gestion des incertitudes inhérentes aux probabilités. Cela permet de modéliser des connaissances dans des domaines où l'incertitude joue un rôle crucial, comme en intelligence artificielle, en apprentissage automatique et en traitement du langage naturel.

Intégration de la Probabilité dans les Logiques Descriptives

La logique des descriptions probabilistes introduit des mécanismes pour représenter des assertions probabilistes sur les concepts et les rôles. Cela permet de capturer des informations telles que "la probabilité qu'un individu donné appartienne à un certain concept" ou "la probabilité qu'une certaine relation tienne entre deux individus".

Modèles et Syntaxe

Dans les PLD, les concepts et les rôles peuvent être associés à des distributions de probabilités. Par exemple, un concept C peut être associé à une distribution de probabilité $P(C)$, qui représente la probabilité qu'un individu soit un membre de C.

Les PLD utilisent généralement une syntaxe étendue des logiques de descriptions standards avec des annotations probabilistes. Par exemple :

Prob($a \in C$) =p signifie que la probabilité que l'individu aa appartienne au concept C est p.

Prob((a, b)∈R) =p signifie que la probabilité que la paire (a,b) soit dans la relation R est p.

Sémantique

Les sémantiques des PLD sont définies en termes de distributions de probabilité sur des modèles d'interprétation classiques des DLs. Un modèle probabiliste est une paire (I,π), où I est une interprétation standard des DLs, et π est une mesure de probabilité sur l'ensemble des interprétations. [2]

Raisonnement et Inférence

Le raisonnement dans les PLD implique de calculer les probabilités de diverses assertions à partir d'un ensemble de connaissances probabilistes. Les algorithmes de raisonnement probabiliste étendent ceux des DLs classiques pour gérer les annotations probabilistes. [2]

Applications

Les PLD sont particulièrement utiles dans des domaines où les données sont incomplètes ou incertaines. Quelques applications notables incluent :

Systèmes de recommandation : En intégrant les préférences des utilisateurs avec des probabilités, on peut améliorer la précision des recommandations.

Bio-informatique : La modélisation des incertitudes dans les données biologiques, telles que les interactions protéine-protéine, permet une meilleure compréhension des processus biologiques complexes.

Robots autonomes : Les PLD permettent aux robots de raisonner sur des environnements incertains, améliorant leur capacité à prendre des décisions en situation réelle.

C) Algorithmes pour l'inférence :

Les algorithmes d'inférence dans les logiques de description probabilistes sont conçus pour déduire de nouvelles informations à partir des connaissances probabilistes disponibles. Parmi ces algorithmes, on retrouve les plus connus :

1)rétro-propagation (backward chaining) :

L'algorithme de backward chaining, ou rétro-propagation, est une méthode d'inférence utilisée dans la logique des descriptions pour répondre à des questions/déduire de nouvelles choses en remontant à travers des règles logiques. Cet algorithme commence par la conclusion recherchée et via un ensemble de règles prédéfinis, remonte ensuite à travers ces règles logiques en vérifiant pour chacune de ces règles si les prémisses sont satisfiables pour voir si les conditions nécessaires pour arriver à cette conclusion sont remplies. Si ces conditions sont satisfaites, l'algorithme continue de remonter jusqu'à ce qu'il atteigne les faits de base qui justifient la conclusion initiale. L'algorithme s'arrête lorsqu'il n'y a plus aucun fait à déduire ou lorsqu'on est arrivé à la conclusion souhaitée. Par analogie, c'est comme résoudre un puzzle en commençant par la pièce finale et en partant de celle-ci essayer de retrouver ses pièces voisines.

2)l'algorithme du tableau (belief propagation) :

L'algorithme des tableaux est une méthode d'inférence utilisée dans la logique des descriptions pour déduire de nouvelles informations à partir de connaissances existantes. Dans le cas où l'on n'utilise pas de probabilité associée aux concepts, l'algorithme des tableaux fonctionne en propageant des informations le long des arêtes d'un graphe représentant les relations entre les concepts et les attributs. Chaque nœud du graphe stocke des informations sur les valeurs des concepts et des attributs, tandis que les arêtes représentent les relations entre ces concepts et attributs. L'algorithme commence par initialiser les concepts selon les informations de départ. Il est important de mentionner qu'avant d'appliquer l'algorithme du tableau, la requête est transformée en forme normale négative (NNF). Cette transformation permet de structurer la requête de manière à ce que toutes les négations soient appliquées directement aux concepts de base, simplifiant ainsi l'application des règles de propagation ultérieur. Ensuite l'algorithme explore le graphe en appliquant les règles d'inférence de la logique des descriptions. Lorsqu'il rencontre une contradiction ou communément appelé "clash" dans les informations, cela signifie que les conditions d'une règle ne sont pas satisfaites et que cette branche de l'arbre d'inférence doit être fermée.

Exemple d’un clash lors du parcours de l’arbre :

Le fait d’avoir un concept « A » ayant pour instance « a0 » et son contraire « ¬A » avec la même instance créera un clash et dans ce cas le parcours de cette branche s’arrêtera.

L'algorithme continue à explorer les autres branches jusqu'à ce qu'il n'y ait plus de nouvelles informations à inférer et de règles à appliquer, ce qui correspond à la fin du graphe.

Voici les règles de propagations sur lesquelles nous allons travailler :

Règle de Disjonction (OU)

La règle de disjonction s'applique lorsque nous avons une formule de la forme C ⊔ *D* dans notre tableau. Cette règle stipule que pour satisfaire C ⊔ *D*, au moins une des sous-formules *C* ou *D* doit être vraie. En termes d'algorithme du tableau, cela signifie que si un nœud contient une disjonction C ⊔ *D*, nous devons créer deux branches distinctes dans notre tableau : une où *C* est vrai et une autre où *D* est vrai. Voici comment cette règle s'applique :

1. **Forme initiale**: Supposons que nous avons un nœud contenant la formule C ⊔ D.
2. **Application de la règle**: Nous divisons ce nœud en deux nouvelles branches, une avec *C* et une avec D.
3. **Propagation**: Les deux branches sont maintenant individuellement ajoutées au tableau, et le processus de vérification continue en appliquant les règles de tableau à chacune de ces nouvelles branches.

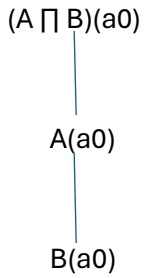
Règle de Conjonction (ET)

La règle de conjonction s'applique lorsque nous avons une formule de la forme C ⊓ *D* dans notre tableau. Cette règle stipule que pour satisfaire C ⊓ *D*, les deux sous-formules *C* et *D* doivent être vraies simultanément. En termes d'algorithme du tableau, cela signifie que si un nœud contient une conjonction C ⊓ D, nous devons créer deux nouveaux nœuds, un contenant *C* et l'autre contenant D. Voici comment cette règle s'applique :

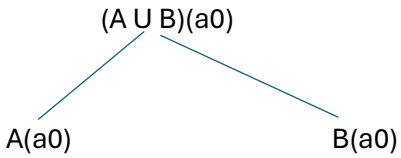
1. **Forme initiale**: Supposons que nous avons un nœud contenant la formule C ⊓ D.
2. **Application de la règle**: Nous étendons ce nœud en ajoutant deux nouvelles branches ou en ajoutant deux nouvelles formules au même nœud, une avec *C* et une avec *D*.
3. **Propagation**: Les deux sous-formules *C* et *D* sont maintenant individuellement ajoutées au tableau, et le processus de vérification continue en appliquant les règles de tableau à ces nouvelles formules.

Voici un exemple illustrant ces 2 règles :

Règle de propagation du ∏ :



Règle de propagation du U :



À ce stade, les informations obtenues représentent les inférences sur les connaissances disponibles, sans considération des probabilités.

Dans le cas où l'on utilise des probabilités associées aux concepts, l'algorithme du tableau est étendu pour tenir compte de ces probabilités. Les nœuds du graphe contiennent alors non seulement les informations sur les concepts eux-mêmes, mais aussi les probabilités associées à ces concepts. L'algorithme commence par initialiser les probabilités des concepts en fonction des informations probabilistes disponibles. Ensuite, il propage ces probabilités le long des arêtes du graphe en utilisant des règles de propagation probabilistes. Ces règles prennent en compte à la fois les relations entre les concepts et les probabilités associées à ces relations. Le processus de propagation se répète jusqu'à ce qu'un état stable soit atteint, où les probabilités des concepts ne changent plus significativement. À ce stade, les probabilités des concepts représentent les inférences probabilistes sur les connaissances disponibles, prenant en compte à la fois les relations entre les concepts et l'incertitude associée à ces relations.

D) Outils et Cadres de Développement

Dans le domaine de la logique des descriptions, plusieurs outils et cadres de développement existent pour faciliter la manipulation, le raisonnement et l'inférence sur des connaissances représentées sous forme de concepts et de relations. Parmi ces outils, on trouve des Framework telles que OWLready2 en Python, qui offrent des fonctionnalités avancées pour travailler avec des ontologies basées sur le langage OWL (Web Ontology Language). OWLready2 permet de charger, manipuler et raisonner sur des ontologies OWL en python assez simplement. Toutefois, nous concernant, il s'avère très peu utile lorsqu'on souhaite manipuler des concepts associés à des probabilités, en plus d'avoir une bibliographie très peu fournie. Dans la même idée, des logiciels dédiés aux inférences, tel que Protégé, offrent une interface conviviale pour la création, la gestion et l'exploration d'ontologies, ainsi que des capacités d'inférence (en grande partie l'algorithme du tableau est utilisé pour les inférences) pour déduire de nouvelles informations à partir des connaissances existantes.

III) Etude de la DL proposée et conception de la solution

Dans cette partie nous allons voir les choix de modélisations du langage DL choisi et adapté à la résolution du problème à savoir son expression syntaxique et sémantique. Enfin, en dernier temps nous allons expliciter l'approche générale d'un point de vue conceptuelle permettant de vérifier la satisfiabilité de la requête.

A) Syntaxe

Dans notre logique des descriptions, nous définissons une syntaxe où les concepts sont fournis par le client et les instances de ces concepts sont représentées par des probabilités et les services associés. Nous travaillons uniquement avec des opérateurs de négation, de conjonction et de disjonction, ainsi que des opérateurs de comparaison tels que supérieur, égal, inférieur, supérieur ou égal, et inférieur ou égal. Cette structure permet de formaliser les exigences de qualité de service du client de manière précise, en spécifiant les probabilités minimales ou maximales requises pour chaque concept afin qu'un service soit considéré comme satisfaisant. Cela permet une évaluation dynamique et rigoureuse des services, garantissant que seuls ceux qui répondent adéquatement aux critères probabilistes et de performance établis par le client sont sélectionnés.

Par exemple, un concept de "bon temps de réponse" peut être associé à une probabilité de 0.8, ce qui signifie que le client souhaite que ce critère soit satisfait avec une probabilité d'au moins 80%.

Ces requêtes seront formulées sous forme de conjonctions et de disjonctions de concepts avec des comparaisons probabilistes (>, <, ≥, ≤). Par exemple, une requête pourrait demander un service qui a une probabilité supérieure à 0,7 d'avoir une bonne réponse temporelle et une probabilité inférieure à 0,5 d'avoir une mauvaise consommation de mémoire.

B) Sémantique

Les données fournies par le client seront représentées par des concepts et leurs instances, traduites en assertions dans l'ABOX. L'ABOX (Assertional Box) contient des assertions qui spécifient quels individus (instances) appartiennent à quels concepts et les relations entre ces individus. Cela permet de représenter les faits spécifiques et les données réelles liées aux services. Par exemple, une assertion dans l'ABOX pourrait indiquer qu'un certain service a une probabilité de 0.8 d'avoir un bon temps de réponse.

La TBOX (Terminological Box), en revanche, consistera en des inclusions de concepts définissant les relations générales entre ces concepts. Elle contient les définitions et les hiérarchies de concepts, permettant de formaliser les règles et les structures logiques entre différents concepts. Par exemple, la TBOX pourrait inclure une règle stipulant que tous les services avec un bon temps de réponse sont également considérés comme performants.

La requête émise par le client sera formulée sous forme de conjonctions et de disjonctions de concepts, permettant ainsi au raisonneur de traiter les attentes de manière précise. Chaque concept est associé à des informations cruciales, comprenant l'ID du service auquel il est rattaché, ainsi qu'une probabilité calculée. Cette probabilité est déterminée en fonction de la proportion du nombre de fois où ce concept est apparu pour un service donné, divisée par le nombre total de concepts ayant ce même service ID. En utilisant cette structure, notre système peut évaluer dynamiquement les services disponibles, en sélectionnant ceux qui répondent aux exigences probabilistes spécifiées dans les requêtes des clients.

Exemple de données reçues :

Events	Service ID	Livraison	Préparation
1	1	12min	4min
2	2	19min	8min
3	2	20min	16min

Ensuite des règles fournies par le ANS nous permettent d'en retirer les concepts pour les services spécifique tels que :

Si (ID = 1,Livraison) < 12 min -> Rapide

Si (ID = 2,Livraison) <= 15 min -> Moyen

Si (ID = 2,Livraison) >30 min -> Lent

Ici les concepts sont : Rapide, Moyen,Lent qui seront associés à un service ID.

Calcul de la probabilité associé à chaque Concept :

Cette probabilité est déterminée en évaluant la proportion du nombre de fois où ce concept est apparu pour un ID de service donné, divisée par le nombre d'exécution de ce service. Ces informations permettent une évaluation précise de la pertinence et de la fiabilité de chaque concept dans le contexte spécifique du service auquel il est lié.

Choix du processus de sélection :

Dans le processus de sélection de chaque service, des exigences spécifiques sont fournies, et seuls les concepts qui répondent à ces exigences avec une probabilité adéquate sont choisis. Nous travaillons dans un environnement incertain (probabiliste) et ce processus de sélection se base sur un algorithme d'inférence (le raisonneur) qui a pour but de choisir les services satisfaisant les exigences demandées. Cette approche permet de garantir que seuls les services dont les attributs de qualité de service correspondent aux attentes prédéfinies sont sélectionnés. Ainsi, chaque service est évalué en fonction de sa capacité à satisfaire les critères requis.

C) Approche conceptuelle

Pour aborder la question de la satisfiabilité d'une requête dans notre système, nous avons opté pour une approche de conception basée sur l'algorithme du tableau explicité en détail dans la partie 2 du point C) du chapitre ci-dessus. Cet algorithme permet de déterminer si une requête est satisfaisable en faisant la négation de la requête initiale et en vérifiant l'absence de contradictions, appelées "clashes", dans les assertions de la ABOX. Plus précisément, cela implique de vérifier systématiquement si la négation de la requête introduit des incohérences au sein des assertions présentes dans la ABOX.

Si un clash est détecté dans chaque branche, cela signifie que la requête (le but) est prouvée. En revanche, l'absence de "clashes" indique que la requête n'est pas satisfiable. Cette méthode rigoureuse permet d'assurer que les décisions prises par notre raisonneur sont cohérentes et logiquement valides.

Dans l'application de l'algorithme du tableau, nous nous limitions à l'utilisation des règles de propagation de la conjonction et de la disjonction. Cette restriction volontaire exclut l'intégration des autres opérateurs logiques tels que le « Existe » ou le « Quelle que soit », ce qui permet de

simplifier le processus d'inférence et d'éviter la complexité liée à la gestion des concepts bloquants et non-bloquants dans l'algorithme du tableau.

L'algorithme tableau est alors déployé pour explorer cette configuration, en tenant compte des règles d'inclusion des concepts et des probabilités associées. À chaque étape, l'algorithme évalue la cohérence des informations disponibles par rapport aux conditions imposées par la négation de la requête et les assertions de la KB.

De plus, nous avons envisagé d'utiliser la forme normale conjonctive (CNF) pour faciliter le parcours de l'algorithme du tableau et l'application des règles d'inférence. La transformation de la requête en une conjonction de clauses simplifie le traitement en permettant d'appliquer les règles de conjonction (ET) dès le début, et de traiter les disjonctions (OU) plus tard dans l'arbre de décision. Comparée à la forme normale disjonctive (DNF), l'utilisation de la CNF réduit significativement le temps de computation. En effet, la CNF permet de structurer les formules de manière à ce que chaque clause soit gérée indépendamment, rendant plus aisée la détection des contradictions ou "clashes". Cette approche améliore non seulement l'efficacité de l'inférence, mais aussi la clarté et la simplicité du processus de vérification de la satisfiabilité des requêtes, assurant ainsi une réponse plus rapide et précise aux attentes des utilisateurs.

Pour illustrer cela, voici un exemple qui montre l'enchainement des étapes :

Étape 1 : Négation et transformation

1. **Transformer la requête en Forme Normale Négative (NNF) :**
    - Applications des lois de De Morgan pour déplacer les négations vers les variables atomiques pour obtenir une forme où les négations n'apparaissent qu'au niveau des variables atomiques.
2. **Convertir la négation du but en CNF :**
    - Utilisation des règles de distribution pour transformer les disjonctions de conjonctions en une conjonction de disjonctions. Par exemple $A \wedge (B \vee C))(A \wedge (B \vee C))$ en $(A \vee B) \wedge (A \vee C)(A \vee B) \wedge (A \vee C)$.

Étape 2 : Préparation de la ABOX et de la TBOX

La ABOX qui contenaient déjà les assertions fournies par le client (ANS) auquel on va ajouter la CNF

La TBOX a été fourni au préalable lors de l'expression des exigences.

Étape 3 : Application de l'algorithme tableau

4. **Initialisation :**
    - Ajout de la négation de la requête (convertie en CNF) à la ABOX. Si la requête est Q, cela devient $\neg Q$.
5. **Propagation de la CNF :**
    - **Conjonction** $(A \wedge B)$: Ajouter A et B aux nœuds de manière séparée et continuer l'exploration pour chacun. Les deux littéraux doivent être vrais simultanément.

- **Disjonction** ($A \vee B$) : Créer des branches séparées pour A et B. Chaque branche représente un cas où l'un des littéraux doit être vrai.

Dans notre cas, nous nous limitons à l'utilisation des règles de propagation de la conjonction et de la disjonction uniquement, ce qui simplifie l'algorithme en évitant d'intégrer la notion de blocs bloquants et non-bloquants.

Étape 4: Évaluation de la satisfiabilité

6. **Développement des branches :**
    - Continuez à appliquer les règles de propagation jusqu'à ce que toutes les branches soient explorées.
    - Si une branche se termine sans clash, cela signifie que la négation de la requête n'est pas contradictoire avec la base de connaissances, donc la requête initiale est insatisfiable.
    - Si toutes les branches mènent à un clash, cela signifie que la négation de la requête est contradictoire avec la base de connaissances, donc la requête initiale est satisfiable.

Exemple simplifié

Supposons que la requête soit $Q=(A \vee B) \wedge (C \vee \neg D)$

1. **Négation :**
    - $\neg Q = \neg((A \vee B) \wedge (C \vee \neg D))$
2. **NNF :**
    - (¬A ∧ ¬B) V (¬C ∧ D).
3. **CNF:**
    - CNF =(¬A∨¬C)∧(¬A∨D)∧(¬B∨¬C)∧(¬B∨D)
4. **Ajouter à la ABOX :**
    - ABOX : Abox U CNF
5. **Algorithme de tableau et recherche de Clashes :**
    - Explorer toutes les branches et vérifier les clashes.
    - Si une branche sans clash existe, ¬Q est satisfiable, donc Q est insatisfiable.
    - Si toutes les branches ont un clash, ¬Q est insatisfiable, donc Q est prouvé.

Remarque : L'utilisation de DNF à la place de CNF constitue aussi une approche correcte.

IV) Etude expérimentale

Nous avons adopté une approche hybride en utilisant deux langages de programmation distincts pour différentes parties du développement, afin de tirer parti des points forts de chacun.

Lecture et sortie de fichiers log avec Python et Pandas

Pour la première partie de notre projet, qui consiste à lire, traiter les fichiers d'entrées log, nous avons choisi Python. Cette décision repose sur plusieurs raisons. Python est un langage de programmation hautement lisible et facile à utiliser, ce qui accélère le développement du code. De plus, Python dispose de bibliothèques intéressantes tel que Pandas permettant de gérer des structures de données complexes et de réaliser des opérations de lecture, de filtrage et de transformation de données avec une syntaxe spécifique. Grâce à cela nous avons pu lire et analyser les fichiers log pour en extraire les informations pertinentes sur les services disponibles et leurs qualités associées. Nous avons également exploré l'utilisation du Framework Owlready2 dans cette phase. Owlready2 est conçu pour la manipulation des ontologies et l'interaction avec les fichiers OWL. Cependant, nous avons rencontré plusieurs difficultés lors de son utilisation. Owlready2 s'est avéré difficile à prendre en main en raison de sa documentation limitée et insuffisante. Et surtout, il n'était pas bien adapté à notre cas d'utilisation spécifique impliquant des concepts probabilistes, ce qui a réduit son utilité pour notre projet.

Implémentation la DL en Java

Pour la seconde partie du projet, impliquant l'implémentation de l'algorithme du tableau pour vérifier la satisfiabilité des requêtes nous avons opté pour Java. Java est reconnu pour sa rapidité d'exécution par rapport à Python, ce qui est crucial pour les opérations intensives en calcul que nécessite notre algorithme. De plus ayant opté pour une approche orienté objet afin de permettre de modéliser de façon plus cohérente le projet et de permettre par la suite des extensions plus facilement comme dans le cas d'ajout de composants de gestions des services ou autres. Java est un langage qui gère mieux l'orienter objet que Python. Cette caractéristique est particulièrement avantageuse pour la modélisation des concepts de services, des probabilités et des règles d'inférence sous forme de classes et d'objets.

Une fois la lecture et le traitement du fichier log effectués dans la partie Python, nous représentons la requête formulée par l'utilisateur sous forme d'arbre, cette structure permet une meilleure représentation des opérateurs logique et est plus adapté a une approche orienté objet. Une autre approche aurait été de directement traiter en chaine de caractères mais celle-ci est non seulement coûteuse elle est plus limitée et n'ouvre pas la possibilité d'étendre le projet.

La première étape qu'est conversion en forme normale négative (NNF), se voit être beaucoup plus simple a réaliser sous la structure d'arbre les étapes de propager les négation vers les littéraux en parcourant l'arbre récursivement. La négation de la requête peut être faite ou pas lors de l'appelle de fonction qui réalise cette étape.

Une fois la requête convertie en NNF et mise en négation, elle est ajoutée à la Base de connaissances. Afin d'appliquer l'algorithme de tableau une approche est de transformer la requête en CNF afin de réduire le temps d'exécution de l'algorithme (une transformation en DNF aiderait tout autant) seulement la transformation en CNF elle-même est couteuse en temps, une méthode exhaustive peut avoir une complexité exponentiel , bien qu'il existe l'algorithme de Tseitin qui réalise la tache en une complexité quadratique au nombre de symboles nous avons opté pour une approche implémentant directement l'algorithme de tableau, une approche exhaustive mais pas exponentiel étant donné qu'elle est en complexité au pire cas de O(n4). Note que notre Abox est une conjonction d'assertion simple, il n'y a donc plus rien à déduire de cette dernière. Il n'y a donc plus que les déductions faites à partir des assertions contenues dans la requête. L'idée étant de parcourir l'arbre de la requête récursivement, de construire les branches de l'algorithme tableau et de vérifier la satisfiabilité à chaque fin de branche.

Cette approche a eu ses propres défis, le plus important étant d'ignorer certaines assertions liées par un "et " logique lorsque celui-ci lie un "ou" logique, bien que ce problème n'aurait pas existé si la formule avait été en DNF. Nous avons cependant fini par surmonter ces défis et les problèmes précédents en utilisant principalement une structure de pile afin de suivre la récursion en enregistrant les sous-formules pas encore traité lorsque ces dernières sont liées pas des "et" logique. Lorsqu'une feuille de l'arbre (requête) est ajoutée á la Abox et la pile est vide (tous les "et" logique ont été traités) la vérification de la cohérence de la Abox est faite (branche de l'algorithme tableau)

V) Conclusion

Ce projet nous a permis d'acquérir une connaissance approfondie d'un nouveau langage, la Logique des Descriptions, ainsi qu'une compréhension des capacités de l'IA symbolique, notamment à travers les inférences et les règles logiques qui l'accompagnent. Au cours de cette entreprise, nous avons eu l'occasion d'explorer en profondeur un langage relativement nouveau pour nous, la Logique des Descriptions, et d'approfondir notre compréhension des capacités de l'IA symbolique, en particulier en ce qui concerne les processus d'inférence et les règles logiques sous-jacentes.

Dès le départ, nous avons été confrontés à des défis de taille : comprendre les subtilités de la logique des descriptions, saisir les enjeux spécifiques du domaine et naviguer dans les choix complexes de modélisation pour l'implémentation de l'algorithme du tableau en Java. Cette phase initiale du projet a demandé un temps de réflexion plutôt conséquent mais une fois mise en place elle nous a permis d'avoir des bases solides nécessaires à la réussite du projet.

L'un des aspects les plus intéressant de notre travail a été notre exploration approfondie de la logique des descriptions. Cette discipline nous a ouvert les portes à un nouveau mode de représentation des connaissances et des données, offrant des moyens formels et expressifs pour décrire et raisonner sur le monde. À travers notre étude, nous avons découvert comment des concepts simples, tels que les opérations d'union et d'intersection, peuvent déjà avoir un pouvoir expressif conséquent. Généraliser notre système aux autres opérateurs logiques peut constituer dans l'avenir un projet intéressant.

Une autre étape cruciale de notre projet a été la conception et l'implémentation de l'algorithme du tableau. Cette tâche complexe nous a confrontés à des défis techniques et conceptuels, mais elle nous a également permis de mettre en pratique les concepts théoriques que nous avions étudiés. En développant notre raisonneur en Java, nous avons dû faire des choix minutieux quant à la modélisation de notre système, en veillant à ce qu'il soit à la fois performant et extensible pour répondre aux exigences changeantes du domaine.

Enfin, ce projet nous a également offert l'opportunité de développer nos compétences en programmation et en gestion de projet avec l'utilisation de nouveaux outils tels que Trello et les fonctionnalités diverses de Github. En travaillant en duo, nous avons appris à collaborer efficacement, à répartir les tâches et à s'organiser correctement. Ce projet nous a également permis de mettre en pratique nos connaissances en programmation Java.

En conclusion, ce projet davantage orienté recherche peu s'avérer complexe au départ mais se révèle moins flou au fur et à mesure que l'on saisit le sujet. Nous avons acquis de nouvelles compétences, élargi nos horizons et développé un profond respect pour la complexité et la richesse de la logique des descriptions et de l'intelligence artificielle symbolique.

Bibliographie

Découverte et présentation de la logique de description :

Markus Krötzsch, František Simancík, and Ian Horrocks ˇ

Department of Computer Science, University of Oxford, UK

https://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2014/KrSH14.pdf

Explication et prise en main de l'algorithme du tableau :

Meghyn Bienvenu, CNRS researcher at LaBRI

https://www.labri.fr/perso/meghyn/teaching/crashcourse/2-tableau-expressive-DLs.pdf

Brefs aperçus de méthodes d'implémentation de l'algorithme du tableau (partie 3 et 4) avec extension des probabilités

Thomas Herchenroder

University of Edinburgh 2006

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ade28efb8eb6d3d564e95674495056c07c68f0e8

Riccardo Zese · Elena Bellodi · Fabrizio Riguzzi · Giuseppe Cota · Evelina Lamma

https://ml.unife.it/wp-content/uploads/Papers/ZesBelRig-AMAI16.pdf

Publication TER : Managing QoS Acceptability for Service Selection : A Probabilistic Description Logics Based Approach

Salima Benbernou , Allel Hadjali , Naouel Karam , Mourad Ouziri

Références

1.Lukasiewicz, T., & Straccia, U. (2008). Managing uncertainty and vagueness in description logics for the Semantic Web. Journal of Web Semantics, 6(4), 291-308.

2.Lutz, C., & Schroder, L. (2009). Probabilistic description logics for subjective uncertainty. Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR), 393-403.

3.Ding, Z., Peng, Y., & Pan, R. (2004). BayesOWL: Uncertainty modeling in semantic web ontologies. Soft Computing in Ontologies and Semantic Web, 3-29