11/01/2021

Web Sémantique

Approfondissement sur les algorithmes de raisonnement

Alexis Guyot
MASTER 2 INFORMATIQUE DIJON

TABLE DES MATIERES

INTRODUCTION

Ce présent rapport fera office de compte-rendu des connaissances acquises lors du travail effectué pour le projet orienté recherche de Web Sémantique. Celui-ci consistait à approfondir mes connaissances théoriques sur le fonctionnement d'un raisonneur, outil central des bases de connaissances, qui leur permet notamment de proposer un processus d'inférence. Il était, pour cela, question d'étudier les sections 5.3.1 à 5.3.3 du chapitre 5 du livre de P. Hitzler, M. Krötzsch et S. Rudolph intitulé *Foundations of Semantic Web Technologies*. Ces dernières traitent d'un fameux algorithme de raisonnement sur la logique ALC, l'algorithme des tableaux.

Ce rapport sera structuré en deux parties principales. Dans un premier temps, je synthétiserai le contenu des trois sous-sections, en proposant également une explication plus détaillée des exemples proposés dans ces dernières. Et dans un second, je présenterai deux exemples créés par mes soins, afin d'illustrer le fonctionnement et les spécificités de l'algorithme des tableaux dans d'autres contextes que ceux présentés par le livre.

PRESENTATION DE L'ALGORITHME DES TABLEAUX

1. Synthèse du contenu des sous-sections

A travers les sections 5.3.1 à 5.3.3, les auteurs du livre s'attardent à expliquer ce qu'est l'algorithme des tableaux, les bases théoriques sur lesquelles il repose, ainsi que son intérêt dans le contexte d'une ontologie.

Ils expliquent ainsi que le besoin d'utiliser un tel algorithme vient du constat qu'une base de connaissance OWL doit pouvoir inférer un certain nombre de conséquences logiques. C'est-à-dire qu'elle doit pouvoir déterminer automatiquement un ensemble de connaissances implicites à partir des connaissances explicites qu'elle possède, et qui sont détaillées dans sa T-Box. Pour rappel, la T-Box d'une ontologie, ou Terminology-Box, est la partie dans laquelle le domaine de connaissance représenté est conceptualisé, et est utilisée en association avec la A-Box, ou Assertion-Box, un ensemble de faits. Parmi les types d'inférence attendus, on retrouve la subsumption, l'équivalence de concepts (*class equivalence*), la disjonction de concepts (*class disjointness*), la cohérence de l'ontologie (*global consistency*), la cohérence d'une classe (*class consistency*), la vérification de type (*instance checking*), et finalement l'extraction d'individus (*instance retrieval*). Pour rappel, la subsumption est le principe pour un concept d'être une sous-classe/d'être inclus dans un autre. Deux concepts sont considérés comme équivalents quand l'un implique l'autre et vice versa, impliquant que toutes leurs instances sont communes. Deux concepts sont disjoints quand il n'existe pas et qu'il ne peut pas exister d'instances communes entre eux, donnant lieu à une intersection toujours vide. Une ontologie est considérée comme consistante s'il n'y a pas de contradiction logique dans la définition de ses concepts et entre le contenu de sa T-Box et de sa A-Box, ce qui revient à dire qu'on peut lui trouver un modèle. Une classe est cohérente si elle n'est pas sous-classe de l'ensemble vide. La vérification de type consiste à vérifier si une instance appartient à un concept ou non. Et finalement, l'extraction d'individus consiste à vérifier pour un concept donné si chaque instance appartient ou non à celui-ci.

Lorsqu'il a fallu proposer un type d'algorithme pour effectuer une telle tâche, les regards se sont tournés vers la logique des prédicats du premier ordre, qui proposait déjà à ce moment-là une famille d'algorithmes qui lui permettait de faire de l'inférence : les algorithmes des tableaux. Puis la lumière a été mise sur un type d'inférence en particulier, la vérification de cohérence de l'ontologie, et ce pour deux raisons.

D'abord, il a été démontré que grâce à la logique de description, tous les types d'inférence cités précédemment pouvaient, d'une manière ou d'une autre, se représenter sous la forme d'un problème de vérification de cohérence sur une nouvelle ontologie, et ce de façon directe ou indirecte. Par exemple, déterminer si la subsumption entre deux concepts A et B est une conséquence logique d'une certaine ontologie O revient à déterminer si cette dernière, à laquelle on rajoute un nouvel individu auquel on assigne le concept ($A \cap \neg B$), n'est pas cohérente. Ici, la subsumption peut directement se représenter comme un problème de vérification de cohérence. L'équivalence entre deux concepts A et B, quant à elle, peut se faire en vérifiant qu'il y a subsumption entre A et B et également entre B et A. Dans ce cas, ce type d'inférence n'est pas converti directement en vérification de cohérence, mais est exprimé uniquement à travers deux subsumptions, qui elles peuvent être ensuite converties en ce type de problème. Cette

observation a donc un premier gros avantage. Effectivement, plutôt que de devoir chercher un algorithme pour chaque type d'inférence, on peut ainsi uniquement en trouver un pour vérifier la cohérence d'une ontologie, et ensuite l'utiliser pour tous les types en proposant une phase préliminaire de conversion d'un type d'inférence à un autre.

Ensuite, l'autre avantage très important de la vérification de cohérence d'une ontologie est que ce type d'inférence en particulier, en plus d'être compatible avec la logique de description, est avant tout un problème de vérification de satisfiabilité de la logique de prédicats du premier ordre, sur une partie de laquelle OWL est fondé. Et ce qui est intéressant avec cette logique, c'est qu'elle était utilisée depuis bien avant l'arrivée des ontologies comme cœur d'une famille d'algorithmes d'inférence : les algorithmes des tableaux. Il allait donc être possible de les utiliser pour résoudre le problème initial, après un léger travail d'adaptation pour correctement convenir aux ontologies : un peu d'optimisation et quelques modifications pour couvrir les contraintes supplémentaires apportées par les logiques de description, notamment le fait qu'elles doivent être décidables, propriété qui n'était pas garantie par la logique des prédicats du premier ordre pure.

La section 5.3.3 propose finalement une explication à travers plusieurs exemples de la méthode qui constitue l'algorithme des tableaux. Les ontologies utilisées sont définies sous forme normale négative, uniquement afin de faciliter la lecture et l'exécution à la main par un être humain. Cette dernière n'est pas du tout obligatoire, l'algorithme des tableaux fonctionnant très bien sur une ontologie qui n'est pas sous cette forme normale. Cependant, celle-ci implique, en utilisant les propriétés de la logique des prédicats du premier ordre, de transformer toutes les inclusions (subsumptions) sous la forme d'ensembles unis par des opérateurs ensemblistes, et de développer toutes les négations factorisées ou enchaînées pour que ce symbole ne puisse précéder que des concepts atomiques. Ainsi, pour deux concepts A et B de l'ontologie, une inclusion $A \subseteq B$, alors considérée comme une implication A -> B, devient l'union $\neg A \cup B$; et la négation d'une union $\neg(A \cup B)$ devient l'intersection $\neg A \cap \neg B$. Evidemment, les deux règles amenées par la forme normale négative entraînent également d'autres transformations, pas que celles décrites dans la phrase précédente. Mais de manière générale, on peut résumer son utilisation comme un moyen de représenter les axiomes d'une ontologie de façon plus simple, avec uniquement des opérateurs ensemblistes, et sans négations factorisées et/ou récursives, qui peuvent facilement entraîner des erreurs.

2. Explication de l'algorithme

Cette partie traitera une à une des trois étapes principales qui constituent l'algorithme des tableaux pour résoudre un problème d'inférence dans une ontologie.

ETAPE 1 : TRANSFORMER EN PROBLEME DE VERIFICATION DE COHERENCE

Comme évoqué dans la précédente partie, une première phase préliminaire peut donc être nécessaire avant de pouvoir appliquer l'algorithme : la transformation en problème de vérification de cohérence d'une ontologie. En effet, chaque type d'inférence souhaité peut se transformer en problème de ce type grâce à la logique de description. Les méthodes pour ce faire, quant à alles, peuvent varier.

Pour 3 des 7 types d'inférence souhaités dans une ontologie K, la méthode est la suivante :

- Créer une nouvelle ontologie K' avec la même A-Box et la même T-Box que K.
- Ajouter à l'ontologie dupliquée K' un tout nouvel individu qu'elle ne contient pas déjà (qu'on appellera ici « a »).
- Ajouter dans la A-Box de K' une assignation au nouvel individu du concept anonyme de la façon suivante :
    - Pour vérifier dans K la **subsumption** entre deux concepts C et D ($C \subseteq D$), ajouter $(C \cap \neg D)(a)$.
    - Pour vérifier dans K la **disjonction** entre deux concepts C et D ($C \cap D \subseteq \bot$), ajouter $(C \cap D)(a)$.
    - Pour vérifier dans K la **cohérence d'un concept** C ($C \subseteq \bot$), ajouter C(a).
- Vérifier que la base K' n'est pas satisfiable.

Pour déterminer si **l'équivalence** entre deux concepts C et D ($C \equiv D$) est une conséquence logique d'une ontologie K, on va vérifier que la subsumption entre C et D ainsi que celle entre D et C sont toutes les deux conséquences logiques de K. On fait alors cela de la façon suivante : Créer une nouvelle ontologie identique à K, ajouter un nouvel individu, lui attribuer les types $C \cap \neg D$ et $D \cap \neg C$, vérifier que la base obtenue n'est pas satisfiable.

Pour vérifier **l'appartenance** d'une instance à un concept C, le principe est identique aux trois inférences précédentes. Cependant, l'individu sur lequel on travaille ne doit non pas être nouveau, mais bien celui dont on souhaite connaître l'appartenance. Si on appelle de nouveau cet individu « a », il faut alors ajouter à la A-Box de la base K' l'assignation $\neg C(a)$, et vérifier que la base K' n'est pas satisfiable.

Enfin, pour **trouver tous les individus** appartenant à un concept C, on vérifie l'appartenance pour chaque individu de K au concept C. Comme toujours on duplique donc la base K, et on ajoute pour chaque instance qu'elle contient une assignation au concept ¬C. On vérifie ensuite que la base obtenue n'est pas satisfiable.

Par exemple, imaginons une ontologie K composée d'une T-Box contenant deux axiomes, T-Box = { $C \subseteq \exists R.D$, $D \subseteq E$ }, et d'une A-Box contenant une assignation, A-Box = { C(a) }. On souhaite savoir si l'instance « a », qu'on sait appartenir au concept C, appartient également au concept anonyme ($\exists R.E$). Pour vérifier l'appartenance d'un individu à un concept, on sait qu'il faut vérifier l'insatisfiabilité d'une base K' identique à K, mais contenant en plus l'assignation du concept inverse à l'individu. On définirait alors la base K' de la façon suivante : K' => T-Box { $C \subseteq \exists R.D$, $D \subseteq E$ }, A-Box = { C(a), **$\neg(\exists R.E)(a)$** }. Puis, on vérifierait la cohérence de K'. On pourrait également transformer avant cela la base dans sa forme normale négative, ce qui est fait par les auteurs dans le livre, ce qui donnerait : K' => T-Box { $\neg C \cup \exists R.D$, $\neg D \cup E$ }, A-Box = { C(a), $(\forall R.\neg E)(a)$ }.

Une fois l'ontologie prête pour la vérification de cohérence obtenue, on peut alors passer à l'étape suivante.

ETAPE 2 : CONSTRUCTION DU TABLEAU INITIAL

Pour commencer, le tableau a besoin d'être initialisé avec le contenu de la A-Box de la base. Un tableau est constitué de plusieurs ensembles de deux types. D'abord, il existe un ensemble par individu de l'ontologie, dans lequel seront rangés des noms de concepts. Ensuite, il existe également un ensemble pour chaque paire d'individus possible, dans lequel seront rangés des noms de propriétés. Tous ces ensembles sont initialisés de la façon suivante :

- Pour chaque individu a de l'ontologie, créer une variable a et un ensemble L(a) initialisé à vide.
- Pour chaque paire (a, b) de variables de l'ontologie, créer un ensemble L(a,b) et l'initialiser à vide.
- Pour chaque déclaration présente dans la A-Box de l'ontologie :
    - Si c'est une assignation d'un concept C à une instance a, alors ajouter le concept C à l'ensemble L(a).
    - Si c'est une propriété d'objets R entre deux instances a et b, alors ajouter la propriété R à l'ensemble L(a,b).

Un point important à retenir est que le tout $\top$ est par défaut contenu dans tous les ensembles L(x) associés aux individus. On ne le représente cependant pas pour des raisons de simplicité d'écriture. Cela implique que si le symbole $\bot$ se retrouve à un moment de l'exécution dans l'un de ces ensembles, alors il faudra considérer cet ajout comme une contradiction. Pour des raisons de simplicité d'écriture, il est également possible de ne pas représenter les paires de variables qui ne possèdent pas de connexions entre eux. Dans l'exemple suivant elles seront représentées pour correctement illustrer le principe des tableaux initiaux, mais dans la suite non.

Par exemple, si on souhaite appliquer l'algorithme des tableaux pour vérifier la cohérence d'une ontologie K définie de la sorte : K => T-Box { $\neg$A U ($\forall$S.B) }, A-Box = { A(a), ($\exists$R.B)(a), (A U B)(c), R(a, b), R(a, c), S(b, b) }.

Dans la A-Box de K, on identifie trois individus, les instances « a », « b » et « c ». Conformément à la méthode présentée dans cette sous-partie, on va alors créer 3 variables a, b et c, 3 sous-ensembles de noms de concepts ainsi que 9 sous-ensembles de noms de propriétés, tous initialisés à vide :

L(a)	L(b)	L(c)	L(a, a)	L(a, b)	L(a, c)	L(b, a)	L(b, b)	L(b, c)	L(c, a)	L(c, b)	L(c, c)
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

On étudie ensuite chaque déclaration de la A-Box, et on en déduit les actions suivantes :

- A(a) : l'individu « a » appartient au concept A, on ajoute alors ce concept à son ensemble L(a) => L(a) = { A }.
- ($\exists$R.B)(a) : l'individu « a » appartient au concept anonyme $\exists$R.B, on ajoute donc ce concept à l'ensemble L(a) => L(a) = { A, $\exists$R.B }.
- (A U B)(c) : l'individu « c » appartient au concept anonyme A U B, on ajoute donc ce concept à l'ensemble L(c) => L(c) = { A U B }.
- R(a, b) : Une propriété R relie les individus « a » et « b », on l'ajoute à l'ensemble L(a,b) => L(a,b) = { R }.
- R(a, c) : Une propriété R relie les individus « a » et « c », on l'ajoute à l'ensemble L(a,c) => L(a,c) = { R }.
- S(b, b) : Une propriété S relie les individus « b » et « b », on l'ajoute à l'ensemble L(b,b) => L(b,b) = { S }.

La A-Box ne contenant maintenant plus de déclaration non-traitée, on sait qu'il sera possible de continuer à l'étape suivante de l'algorithme des tableaux avec le tableau initial suivant :

L(a)	L(b)	L(c)	L(a, a)	L(a, b)	L(a, c)	L(b, a)	L(b, b)	L(b, c)	L(c, a)	L(c, b)	L(c, c)
A, ∃R.B	Ø	A U B	Ø	R	R	Ø	S	Ø	Ø	Ø	Ø

ETAPE 3 : CONSTRUCTION DU TABLEAU

Une fois le tableau initial obtenu, le cœur de l'algorithme des tableaux peut commencer, à savoir l'application successive d'un certain nombre de règles, appelées règles d'expansion. S'il est possible à la fin de l'exécution de l'algorithme d'obtenir un tableau ne contenant pas de contradiction et sur lequel plus aucune de ces règles ne peut être appliquée, alors la base est considérée comme satisfiable et donc cohérente. On considère qu'il y a contradiction dans le modèle quand au moins un des ensembles L contenus dans le tableau contient à la fois un concept C et son contraire ¬C. Cela implique également le concept ⊤ et son contraire ⊥.

Ces règles, au nombre de 5, sont les suivantes :

- La **règle des ∩:** Pour chaque concept anonyme de la forme C ∩ D contenu dans un ensemble L(x), ajouter les concepts C et D s'ils ne sont pas déjà inclus dans L(x).
- La **règle des U** : Pour chaque concept anonyme de la forme C U B contenu dans un ensemble L(x), ajouter au choix ou bien le concept C ou bien le concept D si l'un des deux n'est pas déjà inclus dans L(x).
    - NB : La notion de choix sera plus détaillée dans le paragraphe suivant.
- La **règle des** ∃ : Pour chaque concept anonyme de la forme $\exists R.C$ contenu dans un ensemble L(x), s'il n'existe pas de variable y dont l'ensemble L(y) contient C et dont l'ensemble L(x, y) contient R, alors :
    - Créer une nouvelle variable, appelée par exemple n, qui ne sera donc pas ajoutée à K mais qui sera utilisée dans l'algorithme.
    - Créer un ensemble L(x, n) qui contient R => L(x, n) = { R }.
    - Créer un ensemble L(n) qui contient C => L(n) = { C }.
- La **règle des** ∀: Pour chaque concept anonyme de la forme $\forall R.C$ contenu dans un ensemble L(x), s'il existe une variable y avec qui la variable x partage un ensemble L(x, y) qui contient R, alors ajouter C à l'ensemble L(y) si le concept n'est pas déjà inclus dans celui-ci.
- La **règle de la T-Box** : Chaque variable doit posséder une assignation à chaque concept défini dans la T-Box.
    - NB : Un concept défini dans la T-Box doit rester vrai pour toute assignation de l'ontologie.

Un point important à noter concernant l'algorithme des tableaux est qu'il n'est pas déterministe. Cela signifie que son plan d'exécution dépend de choix à faire, et qu'entre deux applications de l'algorithme sur les mêmes données de départ, le résultat final peut être différent. Deux types de choix différents apportent deux types de non-déterminisme différents. Le premier type concerne l'ordre dans lequel appliquer les règles d'expansion, et est qualifié de « don't care nondeterminism ». Il est nommé de la sorte car ce type de choix n'a pas d'influence sur le résultat final, mais uniquement sur le nombre d'étapes qui seront nécessaires pour arriver à celui-ci. Cela est dû au fait que rien n'est jamais supprimé ou modifié des ensembles du tableau, juste ajouté, et donc que si une contradiction doit avoir lieu pendant l'exécution,

alors celle-ci aura toujours lieu, que ce soit lors de l'application de la première ou de la dernière règle. Ce type de non-déterminisme est donc moins gênant, et le traiter relève de problématiques d'optimisation. Aucune piste n'est évoquée dans la sous-section pour déterminer le meilleur ordre d'application des règles, mais on pourrait imaginer un système similaire à celui mis en place dans les Systèmes de Gestion de Bases de Données pour l'optimisation de plan d'exécution de requêtes, à savoir donc une étude statistique du contenu du tableau initial. L'autre type de choix intervient lors de l'application de la règle des U, et est qualifié de « don't know nondeterminism ». En effet, cette dernière implique qu'à la rencontre d'un concept anonyme de la forme C U D, et à condition que ni C ni D ne se trouve déjà dans l'ensemble, le concept C ou le concept D soit ajouté. Ici, seule l'une des deux options sera considérée dans la suite du plan d'exécution, et on peut très bien imaginer un cas où utiliser un concept mènerait par la suite à une contradiction, alors qu'utiliser l'autre non. Ce type de choix a donc un impact réel sur le résultat final de l'algorithme. La solution mise en place pour gérer cette problématique est un système de *backtrack*, ou en français de retour en arrière, qui permet à l'algorithme de revenir, en cas de détection d'une contradiction, dans l'état dans lequel il était au moment du dernier choix de type « don't know » qu'il a fait, et de choisir l'option alternative.

Un autre point important est la notion de blocage de l'algorithme. En effet, si on utilise naïvement chacune de ces règles sans mettre en place de sécurité, il est tout à fait possible de se retrouver dans un cycle de règles qui boucle à l'infini. Par exemple, avec l'ontologie suivante : K => T-Box = { $\exists R.\top$ }, A-Box = { $\top(a)$ }. Après initialisation, le tableau est constitué d'un ensemble L(a) = { $\top$ }. La règle de la T-Box implique d'ajouter chacun de ses concepts à L, ce qui donne L(a) = { $\top, \exists R.\top$ }. Puisque nous n'avons qu'une seule variable, la règle des $\exists$ provoque forcément la création d'une nouvelle variable x et des ensembles L(x) = { $\top$ } et L(a, x) = { R }. En réexaminant les règles d'expansion, on remarque que la règle de la T-Box n'est plus respectée, puisque l'ensemble de la variable x ne contient pas le concept $\exists R.\top$. On obtient alors L(x) = { $\top, \exists R.\top$ }, ce qui nous amène exactement dans la même situation que précédemment avec la variable a. Cela impliquerait donc de recréer une nouvelle variable, de lui ajouter le concept de la T-Box, et de recommencer cette manipulation à l'infini.

Pour contrer ce problème, l'algorithme des tableaux dispose d'un système de blocage, qui indique tout simplement qu'une règle ne peut être appliquée sur un ensemble que si sa variable correspondante n'est pas bloquée. Trois caractéristiques sont utilisées pour détecter ce type de variable. La première consiste à vérifier si elle correspond à une instance définie dans la A-Box de l'ontologie ou non. La deuxième consiste à vérifier si la variable (qu'on va appeler x) a un ancêtre (qu'on va appeler y). On considère que y est l'ancêtre de x si leur ensemble L(y,x) n'est pas vide, ou si y est l'ancêtre d'une variable avec qui x n'a pas d'ensemble vide. Et finalement la dernière caractéristique qu'on doit vérifier est si l'ensemble L associé à la variable est totalement inclus dans celui d'une autre variable. Si ces trois caractéristiques sont détectées, alors la variable est directement bloquée, et toutes celles dont elle est ancêtre le sont aussi (indirectement cette fois). Si on reprend l'exemple précédent, juste après la création de x et juste avant l'application de la règle de la T-Box, on remarque que :

- x est bien juste une variable, elle ne correspond à aucune instance présente dans K.
- L(a, x) = { R }, et n'est donc pas vide, ce qui fait de a l'ancêtre de x.
- L(x) = { $\top$ } $\subseteq$ L(a) = { $\top, \exists R.\top$ }.

Toutes les caractéristiques sont détectées, ce qui signifie que x doit être bloquée, ce qui empêche l'application de la règle de la T-Box et brise donc le cycle. On effectue ce test avant d'appliquer chaque règle dans un ensemble donné, et on ne fait rien si la variable est bloquée.

En prenant en compte toutes les informations présentées dans cette sous-partie et les précédentes, on peut donc résumer la méthode de l'algorithme des tableaux de la façon suivante :

1. Transformer si besoin l'inférence souhaitée en un problème de satisfaction/cohérence de base et modifier l'ontologie donnée en entrée en conséquence.
2. Créer les variables et les ensembles vides contenus dans le tableau initial.
3. Remplir le tableau initial en parcourant la A-Box de l'ontologie.
4. Déterminer le meilleur ordre d'application des règles d'expansion à partir du contenu du tableau initial.
5. Tant qu'il est possible d'appliquer une règle sur un concept non traité, sur lequel aucune règle n'a déjà été appliquée, d'un ensemble dont la variable n'est pas bloquée, le faire.
   5.1. Lorsqu'un choix est effectué après application de la règle des U, enregistrer l'état du tableau et répertorier le choix.
   5.2. Si une contradiction est détectée dans un des ensembles :
      5.2.1. Si un choix a été effectué par le passé lors de l'exécution et que l'option alternative n'a pas déjà été essayée, alors faire revenir le tableau dans l'état dans lequel il était à ce moment-là et faire le choix alternatif.
      5.2.2. Sinon arrêter l'exécution ici et conclure que la base n'est pas cohérente.
6. Si les différentes applications de règles ont été un succès jusque-là et que plus aucune ne peut être appliquée sur le tableau, alors terminer l'exécution et conclure que la base est cohérente.

3. Explication détaillée des exemples

A l'aide des explications fournies dans la partie précédente, cette partie aura pour objectif de détailler pas à pas les 4 « worked examples » de la section 5.3, présentés respectivement dans la figure 5.13 de la page 192 du livre pour le premier, puis dans la sous-section 5.3.3.4 pour les trois autres.

PREMIER EXEMPLE

On dispose d'une ontologie K, définie de la façon suivante :

T-Box = { Human ⊆ ∃hasParent.Human, Orphan ⊆ Human ∩ ∀hasParent.¬Alive }
A-Box = { Orphan(harrypotter), hasParent(harrypotter, jamespotter) }

On remarque dans cette ontologie la présence de 3 concepts, Human, Orphan et ¬Alive ; de 2 individus, harrypotter et jamespotter ; ainsi que d'une propriété d'objets, hasParent.

Le type d'inférence qu'on souhaite obtenir de la base est de savoir si l'instance jamespotter appartient à la classe ¬Alive.

La première étape consiste donc à transformer ce problème, qui est une vérification de type, en un problème de cohérence d'ontologie. Pour cela, on sait qu'il faut dupliquer la base K en une base K', ajouter dans cette dernière une assignation pour l'individu demandé du concept inverse de celui recherché, puis de vérifier que K' n'est pas satisfiable. On ajoute donc ¬¬Alive(jamespotter) à K', ce qui nous donne :

T-Box = { Human ⊆ ∃hasParent.Human, Orphan ⊆ Human ∩ ∀hasParent.¬Alive }
A-Box = { Orphan(harrypotter), hasParent(harrypotter, jamespotter), ¬¬Alive(jamespotter) }

Si on le souhaite, on peut transformer K' dans sa forme normale négative. Les deux concepts de la T-Box, qui présentent des inclusions, vont devoir être transformés en (¬Human U ∃hasParent.Human) et (¬Orpan U (Human ∩ ∀hasParent.¬Alive)). ¬¬Alive(jamespotter) va également devoir être simplifié, puisque le symbole ¬ ne peut précéder qu'un nom de concept atomique, donnant l'assignation Alive(jamespotter). On obtient alors la nouvelle ontologie NNF(K') :

T-Box = { ¬Human U ∃hasParent.Human, ¬Orpan U (Human ∩ ∀hasParent.¬Alive) }
A-Box = { Orphan(harrypotter), hasParent(harrypotter, jamespotter), Alive(jamespotter) }

La deuxième étape consiste à créer, initialiser et remplir le tableau initial. On commence par créer deux variables, qu'on va appeler h et j, respectivement pour les individus harrypotter et jamespotter. On crée ensuite deux ensembles de noms de concepts, L(h) et L(j) ainsi que les 4 ensembles de noms de propriétés, qu'on initialise tous avec l'ensemble vide. On parcourt la A-Box :

- Orphan(harrypotter) : On ajoute à L(h) le concept Orphan => L(h) = { Orphan }.
- hasParent(harrypotter, jamespotter) : On ajoute à L(h, j) la propriété hasParent => L(h, j) = { hasParent }.
- Alive(jamespotter) : On ajoute à L(j) le concept Alive => L(j) = { Alive }

Toutes les déclarations de la A-Box ont été traitées, la phase d'initialisation se termine donc avec le tableau initial suivant (seuls les ensembles non vides sont représentés) :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent

La troisième étape peut donc commencer. On choisit un ordre arbitraire pour la prise en considération des règles d'expansion, puisque l'optimisation ne nous intéresse pas ici. On prend le tableau initial de gauche à droite, et on commence donc par traiter les valeurs de L(h). Avant de continuer, on effectue le test pour vérifier que h n'est pas bloquée, et puisque ce n'est pas le cas on démarre l'analyse. On commence par remarquer que la règle de la T-Box n'est pas respectée pour l'ensemble L(h). On décide donc d'ajouter le concept ¬Orpan U (Human ∩ ∀hasParent.¬Alive). A noter que rien ne nous oblige à respecter l'ordre dans lequel sont inscrits les concepts dans la T-Box. Ici on choisit donc d'insérer le deuxième concept en premier, et on obtient le tableau suivant :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent

(¬Orpan U (Human ∩ ∀hasParent.¬Alive))		

Toujours dans L(h), on remarque la présence d'un concept anonyme composé avec une union, alors que ni le concept ¬Orphan et ni le concept (Human ∩ ∀hasParent.¬Alive) ne sont présents individuellement dans l'ensemble. Les conditions sont donc réunies pour appliquer la règle des U. Il faut alors faire un choix, et on décide de garder ¬Orphan. On répertorie cette décision, on garde en mémoire l'état du tableau ci-dessus, et on ajoute ¬Orphan à L(h) :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent
(¬Orpan U (Human ∩ ∀hasParent.¬Alive))		
¬Orphan		

A cet instant, on se retrouve donc avec une contradiction, puisque L(h) contient à la fois Orphan et ¬Orphan. Puisqu'il reste un choix répertorié avec une valeur alternative non-essayée, on restaure l'état du tableau, on effectue l'autre choix, et on continue. Le concept (Human ∩ ∀hasParent.¬Alive) est ajouté à L(h). On se retrouve avec un tableau de cette forme :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent
(¬Orpan U (Human ∩ ∀hasParent.¬Alive))		
(Human ∩ ∀hasParent.¬Alive)		

Toujours dans L(h), on s'intéresse maintenant au concept nouvellement ajouté (Human ∩ ∀hasParent.¬Alive), et on remarque qu'il s'agit d'un concept anonyme composé d'une intersection, et que les concepts Human et ∀hasParent.¬Alive ne sont pas présents dans l'ensemble. En adéquation avec la règle des ∩, on les ajoute donc et on obtient le tableau suivant :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent
(¬Orpan U (Human ∩ ∀hasParent.¬Alive))		
(Human ∩ ∀hasParent.¬Alive)		
Human		
∀hasParent.¬Alive		

Toujours dans L(h), on détecte maintenant la présence du concept ∀hasParent.¬Alive. Ce concept anonyme utilise un « pour tout », et la variable h associée à l'ensemble possède bien un ensemble commun L(h, j) avec la variable j dans lequel se trouve la relation hasParent. La règle des ∀ implique donc d'ajouter à L(j) le concept qui fait office de domaine d'arrivée de la relation, soit ¬Alive :

L(h)	L(j)	L(h, j)
Orphan	Alive	hasParent
(¬Orpan U (Human ∩ ∀hasParent.¬Alive))	¬Alive	
(Human ∩ ∀hasParent.¬Alive)		
Human		
∀hasParent.¬Alive		

Une nouvelle fois, on se retrouve face à une contradiction. Or, cette fois-ci, il n'y a plus de choix répertorié avec une valeur alternative non essayée. On en conclut donc que la base K' est incohérente, et donc par

extension que l'hypothèse que l'individu jamespotter appartienne au concept ¬Alive est bien une conséquence logique de K.

DEUXIEME EXEMPLE

Pour cet exemple et les suivants, la présentation sera plus directe et un peu moins détaillée que pour le précédent, à part sur les parties précises de l'exécution de l'algorithme que l'exemple est censé illustrer.

On considère une ontologie K définie de la sorte : K => T-Box = { Human ⊆ ∃hasParent.Human }, A-Box = { Bird(tweety) }.

On cherche une nouvelle fois à effectuer une vérification de type, en cherchant cependant cette fois-ci à savoir si l'individu tweety n'appartient pas au concept ¬Human. Il s'agit donc du même type d'exercice, mais on espère juste cette fois-ci obtenir une base cohérente après transformation en problème de satisfiabilité, résultat qui indiquerait que tweety appartenant au concept ¬Human n'est pas une conséquence logique de K. Le fonctionnement est cependant identique, et on ajoute donc au duplicata K' l'assignation inverse à celle recherchée, soit ¬¬Human(tweety), ce qui en forme normale négative donne Human(tweety). On continue les transformations impliquées par cette dernière et on obtient la base NNF(K') suivante : T-Box = { ¬Human ∪ ∃hasParent.Human }, A-Box = { Bird(tweety), Human(tweety) }.

Un individu, une variable, un ensemble L(t), qui après parcours de la A-Box prend la valeur L(t) = { Bird, Human }. On commence par vérifier que t n'est pas bloquée, ce qui n'est évidemment pas le cas, puis peut alors démmarer l'exécution des règles sur chaque valeur de ce tableau initial réduit. On remarque très vite que la seule manipulation possible est d'ajouter le contenu de la T-Box après application de la règle du même nom. Après ceci, on obtient alors L(t) = { Bird, Human, ¬Human ∪ ∃hasParent.Human }, et on remarque que le concept qu'on vient d'ajouter respecte les critères pour utiliser la règle des U. En effet, le concept est composé avec une union, et ni le concept de gauche ni celui de droite n'apparait unitairement dans L(t). La règle veut donc qu'on fasse un choix. Décider de continuer avec ¬Human mène directement à une contradiction, on peut donc choisir cette option puis directement revenir continuer par retour en arrière avec ∃hasParent.Human. On se retrouve alors avec un tableau de la forme L(t) = { Bird, Human, ¬Human ∪ ∃hasParent.Human, ∃hasParent.Human }. On continue en remarquant la possibilité d'utiliser la règle des ∃ sur la dernière valeur ajoutée à L(t), puisque le concept emploie cet opérateur et que la variable t ne possède pas d'ensemble contenant des noms de propriétés avec une autre variable. Comme préconisé par la règle, on crée alors une nouvelle variable x, son ensemble de noms de concepts L(x) = { Human } et l'ensemble de noms de propriétés L(t, x) = { hasParent }. Les valeurs de L(t) étant toutes traitées, on passe à celles de L(x). On effectue d'abord le test pour savoir si x est bloquée ou non, et on remarque effectivement que c'est le cas. En effet, x est une variable qui n'est pas associée à un individu de l'ontologie, l'ensemble L(t, x) n'est pas vide et L(x) ⊆ L(t). Puisque la variable x est directement bloquée par la variable t, on ne peut pas appliquer de nouvelle règle dans son ensemble. Si cela n'avait pas été le cas, la seule règle applicable sur L(x) aurait été la règle de la T-Box, ce qui aurait entraîné la création à l'infini de nouvelles variables à cause du concept défini avec un ∃. Tous les ensembles étant maintenant traités, plus aucune règle ne peut transformer le tableau. Ce dernier a donc atteint son état final et forme un modèle de l'ontologie K', ce qui signifie que celle-ci est bien cohérente.

Puisque l'ontologie est cohérente, on en déduit alors que l'hypothèse comme quoi tweety appartiendrait au concept ¬Human n'est pas une conséquence logique de K. Cela peut donc être potentiellement symptomatique d'une mauvaise définition de l'ontologie K. En effet, si tweety n'appartient pas au concept ¬Human, alors il pourrait potentiellement appartenir au concept Human, ce qui ferait de lui le premier homme-oiseau depuis Michael Keaton.

TROISIEME EXEMPLE

Ce troisième exemple montre la façon dont l'algorithme des tableaux permet de représenter la réalité impliquée par l'hypothèse du monde ouvert. L'exemple travaille une nouvelle fois sur un exemple d'inférence pour vérification de type, nous commencerons les explications directement avec l'ontologie résultante en forme normale négative. On espère montrer que celle-ci est satisfiable pour impliquer que la vérification de type n'est pas une conséquence logique de la base de départ. On a NNF(K') = T-Box = {}, A-Box = { h(j, p), h(j, a), m(p), m(a), $\exists h. \neg m(j)$).

Le tableau initial, en ignorant les ensembles vides, serait le suivant : L(j) = { $\exists h. \neg m$ }, L(a) = { m }, L(p) = { m }, L(j, p) = { h }, L(j, a) = { h }. On commence par L(j), dont la variable n'est pas bloquée car associée à une instance de la base, et sur lequel on peut appliquer la règle des ∃ sur son premier concept. La variable j possède bien plusieurs ensembles de noms de propriétés qui incluent la propriété h, mais aucune des variables avec qui c'est le cas (p et a) ne possède le concept ¬m dans leur ensemble de noms de concepts. La règle nous amène donc à créer une nouvelle variable x, ainsi que les ensembles L(x) = { ¬m } et L(j, x) = { h }. Le tableau après cette manipulation est le suivant :

L(j)	L(p)	L(a)	L(j, p)	L(j, a)	L(x)	L(j, x)
$\exists h. \neg m$	m	m	h	h	¬m	h

Arrivé à cette étape, l'algorithme ne peut rien faire de plus. Plus aucune règle ne peut être appliquée. L'algorithme a pu terminer, ce qui signifie que l'ontologie K' a un modèle et est donc cohérente, ce qui était le résultat qu'on cherchait à obtenir.

L'intérêt de cet exemple réside dans la forme du tableau final obtenu. En effet, les abréviations utilisées ont la signification suivante : les variables a, j et p correspondent aux prénoms alex, john et peter ; La propriété h correspond à la propriété hasChild ; et le concept m à male. La question à laquelle on essayait de répondre était la suivante : Les enfants de John sont-ils tous des garçons ? Avec le résultat obtenu à la fin de l'exécution de l'algorithme, nous savons maintenant que non, ce qui est en accord avec l'hypothèse du monde ouvert. En effet, la définition de John nous apprend qu'il possède deux enfants qui sont des garçons, certes, mais n'indique à aucun moment que alex et peter sont ses seuls enfants. Or, une conclusion dans un contexte de monde ouvert doit être vraie au moment du raisonnement, mais doit aussi toujours pouvoir rester vraie ensuite. Et dans les faits, rien n'empêche d'imaginer que John a également un autre enfant, pas encore inscrit dans la base de connaissance, qui n'est pas un garçon. Et cet enfant hypothétique est représenté dans le modèle obtenu à la fin de l'exécution de l'algorithme au travers de la variable x, qui n'est pas associée à une instance existante de l'ontologie, qui est l'enfant de John (propriété hasChild de John vers x), et qui n'est pas un garçon, ce qui correspond à la remarque que nous venons de faire à propos de l'hypothèse du monde ouvert.

QUATRIEME EXEMPLE

Pour ce dernier exemple, l'objectif est de décrire quelques astuces pour correctement choisir les règles à appliquer afin de résoudre relativement rapidement un exemple un peu plus compliqué de problème de satisfaction de base de connaissance.

On a NNF(K') => T-Box = { $\neg C \cup \forall S.A$, $\neg A \cup \exists R.\exists S.A$, $\neg A \cup \exists R.C$ } et A-Box = { C(a), C(c), R(a, b), R(a, c), S(a, a), S(c, b), $\forall R.\forall R.\forall S.\neg A(a)$ }

Le tableau initial équivalent est :

L(a) = { C, $\forall R.\forall R.\forall S.\neg A$ }
L(c) = { C }
L(a, a) = { S }
L(a, b) = { R }
L(a, c) = { R }
L(c, b) = { S }

A partir de maintenant de nombreux choix sont possibles pour l'ordre d'application des règles. Pour avancer efficacement dans l'exécution de l'algorithme, il faut identifier des situations qui peuvent devenir avantageuses à court ou à long terme. Par exemple, en étudiant la T-Box, on voit que celle-ci est composée de beaucoup d'unions, qui vont nous amener à devoir faire des choix entre des concepts composés assez complexes, et d'autres atomiques. En étudiant le tableau, on remarque que seul le concept atomique C est utilisé. Si on souhaite par la suite faire apparaitre des contradictions pour réduire les branches et arriver plus vite à la conclusion finale, une bonne approche serait alors de commencer par remplir un des deux ensembles L(a) ou L(c) avec le concept de la T-Box qui proposera d'ajouter le concept ¬C. On décide alors de commencer par L(c), pour éviter de surcharger trop vite L(a). Puisque a et c sont des variables associées à des instances de la base, elles ne peuvent pas être bloquées, et on peut donc appliquer les règles sans crainte au sein de leur ensemble respectif. On décide donc d'appliquer la règle de la T-Box sur L(c), et on commence par ajouter $\neg C \cup \forall S.A$. Après cet ajout, on décide d'appliquer la règle des U sur ce nouveau concept. On détecte tout de suite une contradiction entre C et ¬C, et on continue avec $\forall S.A$, qu'on ajoute à l'ensemble L(c). On décide finalement de terminer le traitement de ce dernier en appliquant la règle des ∀ sur le dernier ajout, ce qui nous amène à rajouter le concept A dans l'ensemble de la variable avec qui c possède un ensemble de noms de propriétés contenant S, soit L(b). On obtient le tableau intermédiaire suivant :

L(a) = { C, $\forall R.\forall R.\forall S.\neg A$ }

L(b) = { A }
L(c) = { C, $\neg C \cup \forall S.A$, $\forall S.A$ }

Avant de continuer à appliquer la règle de la T-Box sur les différents ensembles, on décide de terminer la simplification des termes déjà existants en s'occupant de la valeur $\forall R.\forall R.\forall S.\neg A$ de L(a) grâce à la règle des ∀. Celle-ci amène à ajouter $\forall R.\forall S.\neg A$ dans les ensembles des variables avec qui la variable a partage

un ensemble de noms de propriétés contenant R, par exemple la variable b. On ajoute donc à L(b) la valeur en question.

L(a) = { C, $\forall R. \forall S. \neg A$ }

L(b) = { A, $\forall R. \forall S. \neg A$ }
L(c) = { C, $\neg C \cup \forall S. A, \forall S. A$ }

Pour comprendre la meilleure action à effectuer ensuite, il faut comprendre qu'avec un concept comme $\forall R. \forall S. \neg A$, il va être possible par deux décompositions successives de placer le concept ¬A dans un ensemble. Dans la T-Box, on remarque également la présence du concept $\exists R. \exists S. A$, qu'on pourrait obtenir à travers un choix. Ce dernier permettra lui de placer le concept A après deux décompositions. On va donc vouloir ramener dans L(b) cette valeur, pour après pouvoir placer la contradiction dans un même ensemble, sans retour en arrière possible. On décide donc d'appliquer sur L(b) la règle de la T-Box, et on ajoute à l'ensemble le concept $\neg A \cup \exists R. \exists S. A.$ Ensuite, on décide d'utiliser la règle des U sur ce nouvel ajout. On commence par choisir ¬A, ce qui nous amène directement à une contradiction et à nous faire revenir en arrière pour choisir l'autre valeur alternative. On se retrouve donc avec les ensembles suivants :

L(a) = { C, $\forall R. \forall R. \forall S. \neg A$ }

L(b) = { A, $\forall R. \forall S. \neg A$, $\exists R. \exists S. A$ }
L(c) = { C, $\neg C \cup \forall S. A, \forall S. A$ }

On va donc à partir de maintenant vouloir effectuer la décomposition des deux concepts pour placer A et ¬A. On décide de commencer par décomposer $\exists R. \exists S. A$, pour pouvoir profiter des effets de cette règle sur le tableau. En effet, celle-ci force à la création de nouveaux ensembles de noms de propriétés si ceux-ci n'existent pas ou sont vides, ce qui est une évidence ici puisque aucune propriété n'a pour domaine de départ la variable b. On applique donc la règle des ∃, qui implique la création d'une nouvelle variable x, de son ensemble L(x) = { $\exists S. A$ } et de L(b, x) = { R }. On décompose ensuite $\forall R. \forall S. \neg A$, en plaçant le concept $\forall S. \neg A$ dans L(x), ce qui est possible puisque L(b, x) contient la propriété R. On s'intéresse maintenant à l'ensemble L(x). Sa variable associée n'est pas bloquée car le contenu de son ensemble n'est pas inclus dans celui de son ancêtre b. On réeffectue donc exactement la même manipulation, en commençant par décomposer $\exists S. A$ avec sa règle associée, ce qui amène à la création d'une nouvelle variable y, de son ensemble L(y) = { A } et à L(x, y) = { S }. Puisque L(x, y) existe et contient la propriété S, on peut appliquer la règle des ∀ sur le concept $\forall S. \neg A$ de L(x), ce qui nous amène à placer le concept ¬A dans L(y). On détecte ainsi une contradiction dans L(y), sans retour en arrière possible puisque toutes les valeurs alternatives des choix effectués ont déjà été essayées, ce qui signifie que l'ontologie n'est pas cohérente. Le tableau final est le suivant :

L(a) = { C, $\forall R. \forall R. \forall S. \neg A$ }

L(b) = { A, $\forall R. \forall S. \neg A$, $\exists R. \exists S. A$ }
L(c) = { C, $\neg C \cup \forall S. A, \forall S. A$ }

L(x) = { $\forall S. \neg A$, $\exists S. A$ }

L(y) = { ¬A, A }

APPLICATION SUR DE NOUVEAUX EXEMPLES

Dans cette seconde partie seront détaillées de nouveaux exemples personnels, créés de toute pièce pour illustrer le fonctionnement de l'algorithme des tableaux dans de nouvelles situations.

1. Premier exemple

Pour ce premier exemple, présentant une structure similaire à ceux proposés par le livre, on dispose de l'ontologie K suivante :

T-Box : TravailSérieux ⊆ Travail ∩ ∀sontComportéesDans.MeriteBonneNote
A-Box : TravailSérieux(explicationsDétaillées), sontComportéesDans(explicationsDétaillées, projetWS)

On cherche à savoir si le fait que l'individu projet WS appartienne au concept MériteBonneNote est une conséquence logique de l'ontologie.

Pour commencer, on transpose notre problème en un problème de satisfiabilité de base. Puisqu'il s'agit d'une vérification de type, on sait qu'il faut ajouter dans une base équivalente à K nommée K' l'assignation à l'individu du concept inverse, puis vérifier que l'ontologie obtenue n'est pas satisfiable. On obtient alors l'ontologie K' suivante, directement exprimée sous forme normale négative :

T-Box : ¬TravailSérieux U (Travail ∩ ∀sontComportéesDans.MériteBonneNote)

A-Box : TravailSérieux(explicationsDétaillées), sontComportéesDans(explicationsDétaillées, projetWS), ¬MériteBonneNote(projetWS)

A partir de cette ontologie, on peut commencer par créer le tableau initial de l'algorithme. Pour cela, on crée deux variables p et e, correspondant respectivement aux individus projetWS et explicationsDétaillées. On crée également 2 ensembles de noms de concepts, L(p) et L(e), ainsi que 4 ensembles de noms de propriétés, tous initialisés à vide dans un premier temps. On parcourt ensuite chaque déclaration de la A-Box de l'ontologie, et on modifie le tableau en conséquence :

- TravailSérieux(explicationsDétaillées) : On ajoute le concept TravailSérieux à l'ensemble L(e).
- sontComportéesDans(explicationsDétaillées, projetWS) : On ajoute la propriété sontComportéesDans à l'ensemble L(e, p).
- ¬MériteBonneNote(projetWS) : On ajoute le concept ¬MériteBonneNote à l'ensemble L(p).

On obtient finalement le tableau initial suivant (seuls les ensembles non vides sont représentés :

L(e)	L(p)	L(e, p)
TravailSérieux	¬MériteBonneNote	sontComportéesDans

Sur ce tableau initial, on commence à appliquer les différentes règles d'expansion. On vérifie avant que les variables e et p ne sont pas bloquées, ce qui n'est pas le cas puisqu'elles sont toutes associées à de réels

individus de l'ontologie. On peut donc appliquer nos règles sans problème. On remarque dans un premier temps que seule la règle de la T-Box peut être appliquée, puisque ni L(e) ni L(p) ne contient d'assignation au concept défini dans cette dernière. On décide de l'ajouter dans un premier temps dans L(e). On obtient le tableau suivant :

L(e)	L(p)	L(e, p)
TravailSérieux	¬MériteBonneNote	sontComportéesDans
¬TravailSérieux ∪ (Travail ∩ ∀sontComportéesDans.MériteBonneNote)		

On remarque que L(e) contient un concept composé à partir d'une union, mais ne contient par pour autant ni le concept de gauche, ni le concept de droite. Il est donc possible d'appliquer la règle des ∪. On a alors le choix pour insérer un des deux membres. On décide d'ajouter le concept ¬TravailSérieux, ce qui crée une contradiction, puisque L(e) contient à la fois TravailSérieux et ¬TravailSérieux. L'algorithme nous demande donc de faire revenir le tableau dans l'état où il se trouvait au moment du dernier choix, et de choisir le concept alternatif. L'état en question est celui décrit dans le tableau ci-dessus, on repart de cette base et on décide donc d'ajouter le concept (Travail ∩ ∀sontComportéesDans.MériteBonneNote) à L(e) :

L(e)	L(p)	L(e, p)
TravailSérieux	¬MériteBonneNote	sontComportéesDans
¬TravailSérieux ∪ (Travail ∩ ∀sontComportéesDans.MériteBonneNote)		
Travail ∩ ∀sontComportéesDans.MériteBonneNote		

Toujours dans L(e), on remarque que le dernier ajout apporte un concept composé d'une intersection entre deux concepts qui sont eux absents de l'ensemble. La règle des ∩ est alors applicable, et implique d'ajouter à cet ensemble les concepts de gauche et de droite de l'intersection, soit Travail et ∀sontComportéesDans.MériteBonneNote :

L(e)	L(p)	L(e, p)
TravailSérieux	¬MériteBonneNote	sontComportéesDans
¬TravailSérieux ∪ (Travail ∩ ∀sontComportéesDans.MériteBonneNote)		
Travail ∩ ∀sontComportéesDans.MériteBonneNote		
Travail		
∀sontComportéesDans.MériteBonneNote		

Parmi les concepts ajoutés lors de la précédente étape, on remarque ∀sontComportéesDans.MériteBonneNote. Pour pouvoir appliquer la règle des ∀, il faudrait que la variable e intervienne dans un ensemble de noms de propriétés de la forme L(e, _), avec _ représentant n'importe quelle variable de l'ontologie, et que cet ensemble contienne la propriété sontComportéesDans. Or, on remarque que c'est le cas. Avec la variable p, il existe l'ensemble L(e, p) qui contient la propriété recherchée. La règle est donc applicable, et implique d'ajouter la valeur MériteBonneNote à l'ensemble L(p) :

L(e)	L(p)	L(e, p)
TravailSérieux	¬MériteBonneNote	sontComportéesDans
¬TravailSérieux ∪ (Travail ∩ ∀sontComportéesDans.MériteBonneNote)	MériteBonneNote	
Travail ∩ ∀sontComportéesDans.MériteBonneNote		

Travail		
∀sontComportéesDans.MériteBonneNote		

Après cet ajout, on détecte une contradiction à l'intérieur de L(p) entre les concepts MériteBonneNote et ¬MériteBonneNote. Plus aucune alternative de choix effectué par le passé n'est disponible, ce qui signifie donc que l'algorithme doit se terminer ici et conclure que l'ontologie K' n'est pas satisfiable.

Puisque l'ontologie K' n'est pas cohérente, on peut alors valider le fait que l'individu projetWS appartenant au concept MériteBonneNote est bien une conséquence logique de l'ontologie K.

2. Second exemple

Pour ce second et dernier exemple, j'ai décidé d'innover et d'illustrer une situation jamais représentée dans le livre, à savoir le besoin d'inférer autre chose qu'une vérification de type. On cherchera donc dans cet exemple à vérifier s'il y a une disjonction entre les concepts PizzaRaclette et PizzaAutreFromage dans l'ontologie K suivante :

T-Box :
Raclette ⊆ ¬AutreFromage
PizzaFromage ⊆ ∃comporte.Fromage
PizzaAutreFromage ⊆ PizzaFromage ∩ ∀comporte.AutreFromage ∩ ∀comporte.¬Raclette
PizzaRaclette ⊆ PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage
A-Box : vide

Dans le cas d'une disjonction de cas, la méthode veut qu'on ajoute un nouvel individu, qu'on appelle « a » et auquel on assigne le concept PizzaRaclette ∩ PizzaAutreFromage, dans une ontologie K' identique à K. On obtient alors la définition suivante pour K' en forme normale négative :

T-Box :
¬Raclette ∪ ¬AutreFromage
¬PizzaFromage ∪ ∃comporte.Fromage
¬PizzaAutreFromage ∪ (PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀comporte.¬Raclette)
¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage)
A-Box : (PizzaRaclette ∩ PizzaAutreFromage)(a)

On construit ensuite le tableau initial, pour lequel on crée une variable a correspondant à l'instance du même nom et l'ensemble L(a). On le remplit ensuite en ajoutant à L(a) le seul concept qui lui est assigné. On obtient alors le tableau initial suivant :

L(a)
PizzaRaclette ∩ PizzaAutreFromage

La variable a n'étant bien sûr pas bloquée, nous allons pouvoir appliquer successivement plusieurs règles d'expansion sur le tableau. On commence par appliquer la règle des ∩ pour étendre L(a). Celle-ci implique d'ajouter les deux concepts à gauche et à droite du symbole d'intersection, ce qu'on fait pour obtenir le tableau suivant :

L(a)
PizzaRaclette ∩ PizzaAutreFromage
PizzaRaclette
PizzaAutreFromage

La seule modification possible à présent consiste à appliquer la règle de la T-Box. On décide donc d'ajouter dans un premier temps une assignation au dernier concept défini, à savoir ¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage). On remarque immédiatement que cet ajout est candidat à l'utilisation de la règle des U, et un premier choix doit être effectué puisque ni le membre de gauche, ni le membre de droite n'est présent dans L(a). Comme toujours, on identifie le fait que le membre ¬PizzaRaclette permettrait de créer une contradiction dans l'ensemble. On l'ajoute, ce qui enclenche le système de *backtrack* qui nous oblige à continuer avec la valeur alternative. Ce concept est composé grâce à une intersection, ce qui rend possible l'utilisation de la règle dédiée à ce type de composition. Celle-ci a toujours le même effet que précédemment, ce qui transforme le tableau de la façon intermédiaire suivante :

L(a)
PizzaRaclette ∩ ¬ PizzaFromage
PizzaRaclette
PizzaAutreFromage
¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage)
PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage
PizzaFromage
∀comporte.Raclette
∀comporte.¬AutreFromage

On décide maintenant d'importer un nouveau concept de la T-Box grâce à la règle du même nom. On choisit le deuxième, ¬PizzaFromage ∪ ∃comporte.Fromage. Après ajout, on peut utiliser sur lui la règle des U, et après avoir identifié la contradiction entre PizzaFromage et ¬PizzaFromage, l'algorithme nous oblige à continuer avec le concept ∃comporte.Fromage :

L(a)
PizzaRaclette ∩ ¬ PizzaFromage
PizzaRaclette
PizzaAutreFromage
¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage)
PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage
PizzaFromage
∀comporte.Raclette
∀comporte.¬AutreFromage
¬PizzaFromage ∪ ∃comporte.Fromage
∃comporte.Fromage

On décide maintenant de traiter le dernier concept ajouté, qui nous permet d'appliquer la règle des ∃. Puisque la variable a est la seule du modèle, elle ne possède pas encore d'ensemble de noms de propriété qui contient « comporte », et il est donc nécessaire d'introduire une nouvelle variable x. L'apparition de

celle-ci entraîne aussi la création de deux ensembles, L(x) = { Fromage } et L(a, x) = { comporte }. On décide ensuite d'appliquer une nouvelle fois la règle de T-Box sur L(a), afin d'ajouter une assignation au concept ¬PizzaAutreFromage ∪ (PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀ comporte.¬Raclette). Comme pour les fois précédentes, on le décompose d'abord avec la règle des ∪, ce qui crée une contradiction avec PizzaFromage qu'on élimine grâce au système de backtrack, puis avec la règle des ∩ pour le membre de droite. On obtient alors le tableau suivant :

L(a)	L(x)	L(a, x)
PizzaRaclette ∩ ¬ PizzaFromage	Fromage	comporte
PizzaRaclette		
PizzaAutreFromage		
¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage)		
PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage		
PizzaFromage		
∀comporte.Raclette		
∀comporte.¬AutreFromage		
¬PizzaFromage ∪ ∃comporte.Fromage		
∃comporte.Fromage		
¬PizzaAutreFromage ∪ (PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀comporte.¬Raclette)		
PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀comporte.¬Raclette		
∀comporte.AutreFromage		
∀comporte.¬Raclette		

Toutes les pièces sont maintenant en place pour conclure. On décide de s'attaquer aux différents membres de L(a) candidats à l'utilisation de la règle des ∀. On commence par ∀comporte.¬AutreFromage. La règle nous indique de placer le concept ¬AutreFromage dans les ensembles L(a, _) de noms de propriétés qui contiennent « comporte ». On l'ajoute donc dans L(x). Et on fait enfin de même avec le concept anonyme ∀comporte.AutreFromage, qui lui implique d'ajouter la valeur AutreFromage à L(a, x). On détecte ainsi une contradiction, aucun retour en arrière n'est possible, l'algorithme peut s'arrêter et conclure que l'ontologie K' n'est pas satisfiable. Le tableau final est le suivant :

L(a)	L(x)	L(a, x)
PizzaRaclette ∩ ¬ PizzaFromage	Fromage	comporte
PizzaRaclette	¬AutreFromage	
PizzaAutreFromage	AutreFromage	
¬PizzaRaclette ∪ (PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage)		
PizzaFromage ∩ ∀comporte.Raclette ∩ ∀comporte.¬AutreFromage		
PizzaFromage		
∀comporte.Raclette		
∀comporte.¬AutreFromage		
¬PizzaFromage ∪ ∃comporte.Fromage		
∃comporte.Fromage		
¬PizzaAutreFromage ∪ (PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀comporte.¬Raclette)		

PizzaFromage ∩ ∀comporte. AutreFromage ∩ ∀comporte.¬Raclette		
∀comporte.AutreFromage		
∀comporte.¬Raclette		

Puisque l'ontologie K' n'est pas satisfiable, alors la disjonction entre les concepts PizzaRaclette et PizzaAutreFromage (PizzaRaclette ∩ PizzaAutreFromage ⊆ ⊥) est une conséquence logique de l'ontologie K.

CONCLUSION

Pour conclure, j'ai pu découvrir le fonctionnement classique d'un raisonneur d'ontologie utilisant un algorithme des tableaux à travers ce projet orienté recherche. Ce type d'algorithme vient à l'origine de la logique des prédicats du premier ordre, qui a aussi servi de base à OWL. Grâce à la logique de description, chaque type d'inférence souhaité dans une ontologie peut s'écrire sous la forme d'un problème de vérification de cohérence d'une ontologie, qui est lui-même une forme de problème de satisfaction de la logique du premier ordre. L'algorithme des tableaux est un algorithme qui est non-déterministe, c'est-à-dire que son exécution amène à un ensemble de prises de décisions qui font que d'une exécution à l'autre, avec les mêmes données en entrée, l'allure du tableau final construit peut varier, tout comme le nombre d'étapes à parcourir pour arriver au résultat. A la base, il est aussi non-décidable, propriété qui a dû être couverte par un système de blocage pour être en phase avec la logique de description, qui elle l'est. L'algorithme présenté dans ce présent rapport fonctionne pour la logique ALC, mais a ensuite été étendu pour répondre aux problématiques d'autres logiques de description, notamment SHIQ.