Reliability of answers in semantic peer-to-peer networks

Gia Hien Nguyen

N d'ordre :

Universite Joseph Fourier

Dissertation

soumise pour le grade de

Docteur de l'universite Joseph Fourier

Spécialité : Informatique

par

Gia Hien NGUYEN

Fiabilité des réponses fournies par un réseau logique pair-à-pair

Présentée et défendue publiquement le 24 Novembre 2008 devant un jury composé de:

Serge ABITEBOUL, Directeur de recherche, INRIA	Président
Philippe CHATALIC, Maître de conférence, Université Paris-Sud	Co-directeur de thèse
Christine COLLET, Professeur, Institut national polytechnique de Grenoble	Examinateur
Anne DOUCET, Professeur, Université Paris 6	Rapporteur
Jérome EUZENAT, Directeur de recherche, INRIA	Examinateur
Jérome LANG, Directeur de recherche, CNRS	Rapporteur
Marie-Christine ROUSSET, Professeur, Université de Grenoble	Directeur de thèse

Laboratoire d'Informatique de Grenoble, U.M.R. CNRS 5217,
Universite Joseph Fourier, 38402, Grenoble, France

Remerciements

Je tiens tout d'abord à remercier les directeurs de cette thèse, Marie-Christine Rousset et Philippe Chatalic, pour m'avoir fait confiance, m'avoir guidé, encouragé et conseillé au cours de ces trois années de thèse. La thèse n'aurait pas pu être menée à son terme sans la patience, la disponibilité et l'exigence de leur part.

Je suis très sensible à l'honneur que m'ont fait Serge Abiteboul, Christine Collet, Anne Doucet, Jérôme Euzenat et Jérôme Lang, en acceptant de participer à mon jury lors de la soutenance de cette thèse. Je tiens à leur exprimer toute ma reconnaissance pour l'intérêt porté à ce travail.

J'adresse également ma profonde gratitude à toute l'équipe Hadas, en particulier Cyril Labbé et Alexandre Termier, ainsi qu'à toute l'équipe Iasi-Gémo, en particulier François Goasdoué, Nathalie Pernelle et Hélène Gagliardi, pour leur aide tant sur le plan scientifique qu'humain et pour l'amitié qu'ils m'ont témoigné tout au long de ce travail.

Je tiens également à exprimer ma reconnaissance envers toutes les personnes au Laboratoire de Recherche Informatique d'Orsay, à l'Inria Saclay, et au Laboratoire d'Informatique de Grenoble, qui m'ont apporté leur soutien en vue des procédures administratives ainsi que pour leur encouragement.

Enfin, j'adresse un grand merci à toute ma famille au Vietnam ainsi qu'à mon amie Thu Hà, qui m'ont toujours soutenu et aidé depuis le début de mon séjour en France jusqu'à la fin de ce long travail.

Abstract

Semantic peer-to-peer systems are fully decentralized overlay networks of people or machines (peers) sharing and searching varied resources (documents, videos, photos) based on their semantic annotations using ontologies. In such systems, no user imposes to others his ontology but logical mappings between ontologies make possible the creation of a network of people in which personalized semantic marking up of data cohabits nicely with a collaborative exchange of data. The mappings are exploited during information retrieval or query answering for query reformulation between peers. No central or external authority can neither know the global network (the union of peer ontologies and mappings), nor control the reliability of the peers. This may cause some of the results provided by some peers to be unreliable. This thesis improves the reliability of answers in such networks in different ways. The first part of the thesis focuses on the problem of returning only well-founded answers of a query when the global network is inconsistent. Two algorithms have been proposed. The first one detects causes of inconsistencies. It terminates, is decentralized, complete and correct. The second algorithm uses the results of the first one in order to make only well-founded reasoning. It terminates, is decentralized and correct. Promising results have been obtained from experimentations. The second part of the thesis considers the problem of modeling and handling peers' trust into returned answers. Based on a probabilistic setting, the proposed model of trust has a clear semantics and trust towards answers can be estimated by peers with a good precision using few observations.

Keywords : peer-to-peer, semantic peer-to-peer systems, peer-to-peer inference systems, inconsistencies, distributed reasoning, model of trust, Bayesian approach to statistics

Résumé

Les systèmes pair-à-pair sémantiques sont des systèmes décentralisés de personnes ou de machines (pairs) pour le partage et la recherche de diverses ressources (documents, vidéos, photos), basés sur leurs annotations sémantiques utilisant des ontologies. Dans ces systèmes, aucun utilisateur n'impose aux autres sa propre ontologie. Les mappings sémantiques entre les ontologies créent un réseau où un échange de données est rendu possible. Les mappings sont exploités lors des traitements des requêtes. Aucune autorité centrale ne peut ni connaître le réseau global (l'union des ontologies et des mappings), ni contrôler la fiabilité des pairs. Les réponses fournies dans ces réseaux peuvent être donc peu fiables. Cette thèse contribue à l'amélioration de la fiabilité de ces réponses de plusieurs façons. La première partie de la thèse a pour but de garantir de ne produire que des réponses bien-fondées quand le réseau global est inconsistant. Deux algorithmes ont été proposés. Le premier a pour but de détecter des causes d'inconsistances. Il termine, est décentralisé, complet et correct. Le deuxième algorithme profite des résultats du premier pour garantir de ne raisonner que de manière bien-fondée. Il termine, est décentralisé et correct. Des résultats prometteurs ont été obtenus à partir des expérimentations. La seconde partie de la thèse considère le problème de modéliser la confiance des pairs envers des réponses obtenues. Basé sur un modèle probabiliste, le modèle de confiance proposé a une sémantique claire. Un des avantages de ce modèle est que la confiance en des réponses obtenues peut être estimée par des pairs avec une bonne précision même avec peu observations.

Mot-clés : pair-à-pair, systèmes pair-à-pair sémantiques, systèmes d'inférence pair-à-pair, inconsistances, raisonnement distribué, modèle de confiance, approach statistiques Bayésienne

Résumé des chapitres

Chapitre 1. Introduction générale

Les architectures pair-à-pair sont des infrastructures supportant des réseaux dynamiques d'utilisateurs (appelés pairs), i.e. dans lesquels un pair peut arriver ou partir à n'importe quel moment. Les pairs d'un tel système collaborent librement pour partager leurs ressources. Il existe un large éventail des systèmes pair-à-pair : des réseaux de partage de fichiers, qui permettent des recherches avec des mot-clés; des réseaux de gestion de données, qui transposent l'approche base de données dans un mode pair-à-pair pour des requêtes structurées basées sur un modèle de données et son schéma; des systèmes pair-à-pair sémantiques dans lesquels les ressources sont annotées sémantiquement utilisant des ontologies et dont les requêtes sont basées sur le langage des ontologies; finalement, des systèmes d'inférence pair-à-pair qui sont des systèmes de raisonnement décentralisé. Le premier problème considéré dans cette thèse est celui de gestion d'inconsistances dans les systèmes d'inférence pair-à-pair. Le second se situe dans le contexte des systèmes pair-à-pair sémantiques et concerne la modélisation ainsi que la gestion de la confiance des pairs en des réponses fournies par ces systèmes.

Chapitre 2. Préliminaires

Ce chapitre rappelle des caractéristiques importantes du système d'inférence pair-à-pair SomeWhere, dans lequel le problème de gestion d'inconsistances a été étudié, et du système pair-à-pair sémantique SomeOWL, auquel l'application du modèle de confiance proposé a été considérée. Dans SomeWhere, chaque pair possède une théorie clausale dont une partie du vocabulaire peut être partagée avec d'autres pairs via des mappings. La recherche des conséquents d'une clause, considérée comme une requête, se fait tout d'abord par un raisonnement local dans le pair où la requête a été interrogée, et puis éventuellement par des propagations des variables partagées dans d'autres pairs via des mappings, qui initialisent des raisonnements décentralisés dans le réseau. Les conséquents trouvés par ces raisonnements, qui

sont à distant, sont renvoyés au pair initiale pour former des conséquents finaux de la requête. Dans SomeOWL, les pairs organisent leurs ressources en utilisant une ontologie de classes. Un mapping reliant deux pairs exprime une correspondance sémantique entre des classes de ces pairs. Un utilisateur pose une requête à un pair donné en utilisant le langage de l'ontologie du pair. SomeOWL est déployé au-dessus de SomeWhere: grâce à un encodage propositionnel des ontologies des pairs en des théories clausales et de la requête en une clause, la reformulation de la requête dans SomeOWL est réduite à la recherche des conséquents d'une clause dans SomeWhere.

Chapitre 3. Etat de l'art sur la gestion d'inconsistances

La gestion d'inconsistances est un problème qui a été beaucoup étudié. Des travaux ont été réalisés dans le cadre des bases de données pour garantir la correction dans les réponses des requêtes, et dans des bases de connaissances pour en tirer correctement de nouvelles connaissances. Ces travaux s'appuient sur deux approches principales. La première est de chercher à modifier ces bases pour restaurer leur consistance. La deuxième est d'accepter l'existence d'inconsistances mais d'adapter le mécanisme de raisonnement comme dans les logiques paraconsistantes. Une autre possibilité pour gérer des inconsistances est d'utiliser des approches numériques, pour préférer une source d'informations à une autre, ou un conséquent à un autre. Dans un système pair-à-pair, vu la nature décentralisée des pairs et l'accès restreint dans leur contenu, il est difficile d'y appliquer la première approche. Par conséquent, nous avons adopté la deuxième approche pour gérer des inconsistances dans SomeWhere.

Chapitre 4. Gestion d'inconsistances dans SomeWhere

Sous l'hypothèse que les théories locales sont consistantes, les inconsistances dans le réseau global de SomeWhere sont donc dues aux mappings. Deux algorithmes ont été proposés pour gérer ces inconsistances. L'algorithme P2P-NG détecte des causes des inconsistances créées par l'arrivée d'un nouveau mapping. Ces causes sont caractérisées par des *nogoods*, qui sont des ensembles de mappings possiblement distribués. Les *nogoods* détectés sont stockés dans le pair qui ajoute le nouveau mapping. P2P-NG termine, est correct et complet. L'algorithme WF-DeCA ne cherche que des conséquents bien-fondés des requêtes, en s'appuyant sur les résultats de P2P-NG. Il termine et est correct. Ces deux algorithmes ont été implémentés dans la plateforme SomeWhere. Un troisième algorithme a été conçu, de sorte

que les résultats de ses expérimentations puissent être transposés dans P2P-NG et WF-DeCA. Ces résultats sont prometteurs.

Chapitre 5. Etat de l'art sur les modèles de confiance et de réputation pour les réseaux sociaux et pair-à-pair

Ce chapitre commence par distinguer la confiance de la réputation. Il est pertinent de considérer la réputation d'une personne dans les contextes où il y a un consensus des gens sur ce qui caractérise son honnêteté. Par exemple, les gens ont le même point de vue de ce qui fait un bon vendeur dans le contexte du commerce électronique : celui qui fourni correctement l'article acheté à l'acheteur. Par contre, dans certains autres contextes où il n'y a pas de tel consensus, il est pertinent de parler de confiance. Par exemple, un novice fera confiance aux conseils d'une grande enseigne sur les problèmes techniques des portables, mais un expert n'y fera pas confiance. De nombreux modèles, de confiance ainsi que de réputation, ont été proposés pour les réseaux sociaux et pair-à-pair. Ils sont aussi distingués les uns des autres par leur modèle de calcul, qui soit basé sur un modèle mathématique, soit ad-hoc.

Chapitre 6. Un modèle probabiliste de confiance pour des systèmes pair-à-pair sémantiques

Dans ce chapitre, un modèle probabiliste a été proposé pour modéliser et calculer la confiance d'un pair dans les réponses obtenues avec des systèmes pair-à-pair sémantiques. La confiance a été définie, non au niveau des pairs, mais au niveau des annotations sémantiques des réponses renvoyées. En supposant qu'un utilisateur évalue aléatoirement des ressources qu'il obtient comme satisfaisantes ou non-satisfaisantes, il est facile de construire pour chaque pair une base d'observations directes sur l'annotation sémantique de ces ressources. Un pair peut donc estimer la probabilité qu'une ressource soit satisfaisante, compte tenu de son annotation sémantique et des observations déjà réalisées. Ce modèle de calcul est basé sur la méthode des statistiques Bayésienne. Même avec peu d'observations, la précision de l'estimation de la confiance est bonne. Dans le cas où les observations directes sont manquantes, il est possible pour un pair de collecter des observations d'autres pairs et s'en servir pour estimer sa confiance dans les ressources renvoyées. Quand ce modèle est appliqué au système pair-à-pair sémantique SomeOWL, une stratégie de collection d'observations, qui est pratique, a été construite. Alors que ce modèle reste à être

implémenté, son passage à l'échelle semble être garanti par la simplicité de son modèle de calcul.

Chapitre 7. Conclusion et perspectives

Après avoir rappelé les principales contributions de cette thèse, ce chapitre esquisse quelques directions pour sa poursuite. Premièrement, il est possible de permettre l'utilisation d'un mapping même si la détection des éventuelles inconsistances causées par celui-ci n'a pas encore fini. Deuxièmement, les informations collectées lors des traitements des requêtes pourront être exploitées davantage pour ordonner les réponses. Troisièmement, les valeurs de confiance pourront être utilisées pour découvrir de nouveaux mappings, pour l'acheminement des requêtes ainsi que des réponses. Finalement, il est aussi possible de considérer la gestion de temps dans le modèle de confiance proposé.

Résumé substantiel de la thèse

Les architectures pair-à-pair sont des infrastructures supportant des réseaux dynamiques d'utilisateurs (appelés pairs), i.e. dans lesquels un pair peut arriver ou partir à n'importe quel moment. Les pairs d'un tel système collaborent librement pour partager leurs ressources. Il existe un large éventail des systèmes pair-à-pair : des réseaux de partage de fichiers, qui permettent des recherches avec des mot-clés; des réseaux de gestion de données, qui transposent l'approche base de données dans un mode pair-à-pair pour des requêtes structurées basées sur un modèle de données et son schéma; des systèmes pair-à-pair sémantiques dans lesquels les ressources sont annotées sémantiquement utilisant des ontologies et dont les requêtes sont basées sur le langage des ontologies; finalement, des systèmes d'inférence pair-à-pair (P2PIS) qui sont des systèmes de raisonnement décentralisé. Le premier problème considéré dans cette thèse est celui de gestion d'inconsistances dans les systèmes d'inférence pair-à-pair. Le second se situe dans le contexte des systèmes pair-à-pair sémantiques et concerne la modélisation ainsi que la gestion de la confiance des pairs en des réponses fournies par ces systèmes.

I. Gestion des inconsistances dans les P2PIS

Notre étude se situe principalement dans le contexte du système d'inférence pair-à-pair SomeWhere.

1. Introduction de SomeWhere

Dans SomeWhere, chaque pair contient une théorie clausale. Les pairs se connectent par des mappings : un mapping entre deux pairs est une clause concernant des variables de ces deux pairs. Les variables apparaissant dans un mapping sont appelées des variables partagées. L'architecture de SomeWhere est totalement décentralisée. Chaque pair ne connaît que sa théorie locale et les mappings qu'il a avec ses voisins. Quand un nouveau pair arrive, il déclare des mappings avec certains pairs qu'il connaît. La théorie globale d'un réseau SomeWhere est l'union de toutes les théories locales et les mappings. Aucun pair ne connaît cette théorie globale.

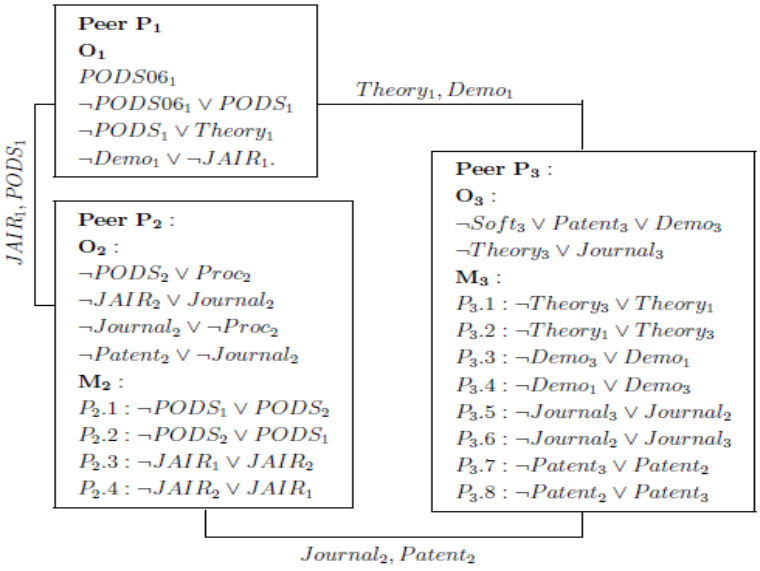

Figure 1: Exemple d'un réseau SomeWhere

Prenons un exemple d'un réseau SomeWhere, correspondant à Figure 1. Dans cet exemple :

- P_1 peut être interrogé par des chercheurs pour savoir où l'on peut soumettre ses travaux. Les connaissances de P_1 sont: $PODS06$ est ouvert ; soumettre à $PODS06$ implique soumettre à $PODS$; seuls les résultats théoriques sont acceptés pour $PODS$; une démonstration ne peut pas être soumise à $JAIR$.
- P_2 distingue des proceedings des journaux. Les connaissances de P_2 sont : soumettre à $PODS$ implique soumettre à une conférence avec proceedings ; soumettre à $JAIR$ implique soumettre à un journal ; un même résultat ne peut pas être soumis à une conférence et un journal ; les résultats en attente ne peuvent pas être soumis à un journal.
- P_3 a des connaissances sur la politique de valorisation des recherches: des logiciels devraient être en attente ou soumis comme des démonstrations ; des résultats théoriques ne peuvent être soumis à des journaux;
- les connaissances locales à ces trois pairs P_1, P_2, P_3 sont représentées respectivement par les ensembles O_1, O_2, O_3.
- l'ensemble M_2 des mappings stockés dans P_2 exprime l'équivalence

entre $PODS_1$ et $PODS_2$ (respectivement $JAIR_1$ et $JAIR_2$) via des mappings identifiés par $P_2.1$, $P_2.2$, $P_2.3$, $P_2.4$. L'ensemble M_3 des mappings stockés dans P_3 établissent l'équivalence entre des variables de P_3 avec celles de deux autres pairs.

Le système SomeWhere s'appuie sur l'algorithme DeCA dont l'objectif est de chercher des conséquents premiers propres d'une clause par rapport à la théorie globale. La correction de cet algorithme s'appuie cependant sur l'hypothèse que la théorie globale de SomeWhere est consistante. Pourtant, cette hypothèse n'est pas réaliste, vu la nature décentralisée et multi-auteurs des théories locales.

Dans la Figure 1, l'ensemble suivant des clauses est inconsistant : $\{\neg Journal_3 \vee Journal_2, \neg Theory_3 \vee Journal_3$, $\neg Theory_1 \vee Theory_3$, $\neg PODS_1 \vee Theory_1$, $\neg PODS06_1 \vee PODS_1$, $PODS06_1$, $\neg Journal_2 \vee \neg Proc_2$, $\neg PODS_2 \vee Proc_2$, $\neg PODS_1 \vee PODS_2\}$. Parmi ces clauses, il y a des mappings : $\neg Journal_3 \vee Journal_2$ $(P_3.5)$, $\neg Theory_1 \vee Theory_3$ $(P_3.2)$ et $\neg PODS_1 \vee PODS_2$ $(P_2.1)$.

Dans ce travail, nous supposons que les théories locales sont consistantes. Les inconsistances dans la théorie globale sont donc créées par des mappings. Nous caractérisons donc un nogood ng comme un ensemble de mappings tel que $O \cup ng$ est inconsistant, où O est l'union des théories locales (Définition 14).

2. Détection des nogoods

Nous proposons l'algorithme P2P-NG pour la détection des nogoods. P2P-NG est une adaptation de DeCA dans le but de chercher **toutes** les preuves de la clause vide conséquent d'une clause. Avant d'ajouter un nouveau mapping dans le réseau, un pair utilise P2P-NG pour vérifier si ce mapping peut être la cause d'une inconsistance, i.e. si la clause vide est un conséquent de ce mapping. Dans ce cas, le pair stocke localement l'ensemble formé par ce nouveau mapping et les autres mappings utilisés dans la preuve de la clause vide (son **mapping support**) comme un nogood.

Dans l'exemple de Figure 1, supposons que les pairs arrivent dans l'ordre P_1, puis P_2, puis P_3 et leurs mappings sont ajoutés dans l'ordre de leur numéro. L'ajout des 4 mappings de P_2 et de 4 premiers mappings de P_3 ne cause pas d'inconsistance. Quand le mapping $P_3.5$ est ajouté, P2P-NG est appelé et il détecte que la clause vide est un conséquent de $P_3.5$. Le mapping support de cette preuve est $\{P_3.2, P_2.1\}$. Le nogood $ng = \{P_3.5, P_3.2, P_2.1\}$ est donc stocké

dans P_3. L'ajout des autres mappings de P_3 ne crée pas de nouveaux nogoods.

Nous avons montré que l'algorithme P2P-NG termine, est correct et complet, dans le sens où tous les nogoods causés par l'ajout d'un mapping sont détectés.

3. Raisonnement bien-fondé dans SomeWhere

A partir de la définition de nogoods, nous savons que tout sous-ensemble du réseau global qui ne contient aucun nogood est forcément consistant. Donc, nous devons garantir de ne renvoyer un conséquent que s'il peut être déduit avec au moins une preuve qui ne contient aucun nogood. Un tel conséquent est considéré **bien-fondé** car il est déduit à partir d'un sous-ensemble consistant du réseau global (Définition 16).

Pour ne chercher que des conséquents bien-fondés, lors des traitements des requêtes, nous devons collecter **tous** les nogoods qui pourraient invalider une preuve d'un conséquent. Comme un nogood est un ensemble de mappings, nous allons faire des tests d'inclusion entre le mapping support d'une preuve et les nogoods collectés. Si le mapping support contient un des nogoods collectés, la preuve n'est pas bien-fondée et donc rejetée. Nous proposons l'algorithme WF-DeCA qui assume toutes ces procédures. WF-DeCA est une adaptation de DeCA, il est utilisé au moment de la recherche des conséquents d'une clause, il termine et est correct. Sa correction dépend de la complétude de P2P-NG, qui a été aussi démontrée.

Pour expérimenter les algorithmes P2P-NG et WF-DeCA, nous avons tout d'abord conçu un troisième algorithme, appelé DeCAbL. Cet algorithme est une adaptation de DeCA, ayant un paramètre supplémentaire indiquant la taille maximale qu'un conséquent peut avoir. Par exemple, si nous limitons ce paramètre à 0, nous ne cherchons que la clause vide. La correction et la complétude de DeCAbL ont été démontrées. L'implémentation de l'algorithme P2P-NG a été faite, en étendant DeCAbL pour la prise en compte des mappings supports de la clause vide, en vue de la détection des nogoods. L'implémentation de l'algorithme WF-DeCA a été faite aussi en étendant DeCAbL pour la collection des nogoods et les tests d'inclusion entre les mapping supports des conséquents et les nogoods collectés. Les résultats des expérimentations sur DeCAbL peuvent donc être transposés à P2P-NG et WF-DeCA. Différents tests ont été réalisés et nous ont apporté des résultats prometteurs.

II. Un modèle probabiliste de confiance pour les systèmes pair-à-pair sémantiques

Dans un système pair-à-pair sémantique, nous supposons qu'une ressource est renvoyée comme une réponse à une requête accompagnée d'une justification logique sous forme de label (annotation sémantique). Le label $L(r)$ de la ressource r est un ensemble de classes, de possiblement différents pairs. Nous supposons aussi qu'un utilisateur évalue, de manière aléatoire, des ressources qu'il obtient, comme satisfaisantes ou non-satisfaisantes. Chaque évaluation sur une ressource est une observation sur son label. Nous pouvons donc construire pour chaque pair P_i une base d'observations directes sur les labels des ressources. Chaque ligne d'une telle base a trois champs, correspondants au nom d'un label (L), le nombre des observations satisfaisantes ($\#_i^+(L)$) et le nombre des observations non-satisfaisantes ($\#_i^-(L)$) sur celui-ci. Table 1 est un exemple d'une base d'observations dans le pair P_1.

Label (L)	$\#_1^+(L)$	$\#_1^-(L)$
$P_2 : MyActionFilms$	30	6
$P_2 : MyCartoons$	3	15
$P_4 : Western$	2	8
$P_5 : Italian\ \ P_4 : Western$	0	6
$P_6 : AnimalsDocum$	14	14
$P_7 : JeanRenoir$	22	11
$P_3 : Bollywood$	6	35

Table 1: La base d'observations dans le pair P_1

Notons que, comme des labels sont des justifications logiques des ressources, une observation sur un label L', telle que $L \subseteq L'$, est aussi une observation pertinente sur L. Un pair P_i peut donc calculer le nombre total d'observations satisfaisantes (respectivement non-satisfaisantes) relevantes à un label L, noté par $O_i^+(L)$ (respectivement $O_i^-(L)$), en accumulant tous les nombres $\#_i^+(L')$ (respectivement $\#_i^-(L')$) où L' est un label relevant à L (Définition 17).

Nous définissons la confiance d'un pair P_i en un label L comme la probabilité qu'une ressource annotée par L soit satisfaisante pour P_i, étant données les observations relavantes à L (Définition 18). Théorème 8 permet d'estimer cette confiance par la formule

$$\frac{1 + O_i^+(L)}{2 + O_i^+(L) + O_i^-(L)}$$

avec un écart type de

$$\sqrt{\frac{(1+O_i^+(L)) \times (1+O_i^-(L))}{(2+O_i^+(L)+O_i^-(L))^2 \times (3+O_i^+(L)+O_i^-(L))}}$$

Le modèle de confiance que nous proposons est basé sur l'approche des statistiques Bayésienne, il a une sémantique claire. La formule de l'écart type permet aussi de calculer le nombre minimal d'observations pour atteindre un niveau souhaité de précision de l'estimation. Dans ce modèle, même avec peu d'observations, la précision de l'estimation de la confiance est bonne. Dans le cas où les observations directes sont manquantes, il est possible pour un pair de collecter des observations d'autres pairs et s'en servir pour estimer sa confiance dans les ressources renvoyées, sous l'hypothèse que ces pairs ont un même critère de satisfaction avec lui.

Le modèle de confiance que nous proposons est général et peut être utilisé dans de différents systèmes pair-à-pair sémantiques. Nous avons appliqué ce modèle au système SomeOWL, qui a été déployé au-dessus de la plateforme SomeWhere. Un point important dans SomeOWL est que la reformulation des requêtes peut être faite par l'algorithme DeCA de SomeWhere. Nous en avons profité pour concevoir spécialement pour SomeOWL une stratégie de collection des observations, qui étend les messages de réponse envoyés par DeCA pour transmettre également des observations éventuellement pertinentes pour estimer la confiance en des ressources renvoyées. Avec cette stratégie, le gain en termes de temps peut être significatif, car des observations d'autres pairs sont disponibles dès que les ressources sont trouvées.

III. Conclusion et perspectives

Nous avons étudié le problème de gestion d'inconsistances dans les systèmes d'inférence pair-à-pair, comme SomeWhere, ainsi que le problème de la modélisation et la gestion de la confiance des pairs en des réponses fournies par des systèmes pair-à-pair sémantiques, comme SomeOWL. Cette thèse ouvrirait quelques directions de recherche ultérieure. Premièrement, il est possible de permettre l'utilisation d'un mapping même si la détection des éventuelles inconsistances causées par celui-ci n'a pas encore fini. Deuxièmement, les informations collectées lors des traitements des requêtes pourront être exploitées davantage pour ordonner les réponses. Troisièmement, les

valeurs de confiance pourront être utilisées pour découvrir de nouveaux mappings, pour l'acheminement des requêtes ainsi que des réponses. Finalement, il est aussi possible de considérer la gestion de temps dans le modèle de confiance proposé.

Contents

Chapter 1

GENERAL INTRODUCTION

Nowadays, there are uncountable useful applications that are built on top of the Internet. The architecture of these applications have been evolving. One of the most used application running on the Internet is an Internet browser, such as Mozilla Firefox or Netscape Navigator. The architecture of this type of application is the so-called client-server. The browser on your computer is a client that sends (and receives) data to (and from) a centralized server, which is the source of information.

Although the application of Internet browser is still widespread, the client-server architecture itself has several limitations, such as the problem of bandwidth bottleneck or the computing power of a single server. The Encyclopedia of Life, at the address http://www.eol.org, is an example of this kind of problem. Just 5 hours after its opening ceremony on the 26^{th} February 2008, access to the page was not possible due to the very great amount of visits : more than 11 millions visits in just 5 hours. A single a server can not handle such a great amount of users.

The so-called *peer-to-peer* infrastructure is an alternative to the client-server infrastructure and does not have those limitations.

Peer-to-peer architecture

Peer-to-peer architecture is an infrastructure supporting peer-to-peer systems (applications) which are networks of users. A user, often called a *peer*, can join

or leave a peer-to-peer system at anytime. When participating to a peer-to-peer system, a user can benefit the resources (data, information, computing capacity) provided by the other users of the system. By participating to a system, a user also accepts to share the resources it has so that the other peers in the system can benefit. This makes each user of a peer-to-peer system a source and also a consumer of resources.

There are a lot of services that can be provided in a peer-to-peer mode, for example: file exchanging, processing power sharing, storage capacity sharing,... In a peer-to-peer setting, the peers cooperate for providing these services without any centralized control authority.

There is a broad spectrum of peer-to-peer systems : the peer-to-peer file sharing systems allow a keyword-based search ; the peer-to-peer data management systems transpose in the peer-to-peer setting the database approach of a data model and schema-based structured queries ; the semantic peer-to-peer systems provide semantic annotations of resources using terms of ontologies and ontology-based queries ; finally, the peer-to-peer inference systems are decentralized reasoning systems.

Keyword-based peer-to-peer file sharing systems

In keyword-based peer-to-peer systems, queries are made of keywords and query answering is the process of string matching, i.e. finding resources whose name containing the keyword of the query. In these systems, peers store their files with no special organization. Each file is associated with a few keywords. The peers share their files by making them available to search of any peer in the network.

Gnutella (5), Chord (62) and P-Grid (9) are three most known keyword-based peer-to-peer file sharing systems. They differ mainly in the routing of queries during the query answering process.

In Gnutella, a peer searches for a file, stored in some other peers in the network, by *using keywords as the query*. In a query answering process, a query is a message asked to the neighbors of the querying peer. These are the peers to and

from which the querying peer can send and receive messages via the Internet (TCP/IP). If a neighbor has the file satisfying the query, it sends the file back to the querying peer; otherwise it asks its neighbors for this file, always using the same keywords for the query. This simple routing strategy is called *unstructured*.

Like Gnutella, a query in Chord or P-Grid is also a keyword. However, the query routing for searching is *structured*. Actually, searching in Chord or P-Grid is supported by an overlay network. An overlay network is a logical network built on top of the physical network (TCP/IP). Chord and P-Grid use Distributed Hash Tables (DHTs) in their underlying network layer for ensuring messages exchange. A DHT provides the information look up service for peer-to-peer applications through the $put(key, data)$ and $get(key)$ primitives. This permits a more efficient routing for queries than in Gnutella.

The keyword-based search works well for domains where there is a consensus of peers on the names of the files. For example, music file names are normally the same in different peers if they are about the same songs (or the same albums or the same artists). The results of the searches in these domains are thus often correct. However, there are domains where there is no such a consensus on the keywords characterizing the files they store. A same picture in which there is a rose may be named *flower picture* by a peer, but *love picture* by an other. The query *rose* asked by a third peer in this case would not return the picture that the peer is looking for, because of the difference naming between peers. Consequently, when each peer develops its own naming standard, the result of a keyword-based search may not be accurate.

Peer Data Management Systems (PDMSs)

The notion of Peer Data Management Systems (PDMSs) has been presented first in (35; 36), through the Piazza system. In Piazza, each peer has its own data and schema and can mediate with some other peers by declaring mappings between its schema and the schemata of those peers. The underlying data model of the first version of Piazza (35) is relational and the mappings between relational peer

schemata are inclusion or equivalence statements between conjunctive queries. Such a mapping formalism encompasses the local-as-views and the global-as-views (37) formalisms used in information integration systems based on single mediators. This makes the query answering undecidable unless some restrictions are imposed on the mappings or on the topology of the network (35). The currently implemented version of Piazza (36) relies on a tree-based data model : the data is in XML and the mappings are equivalence and inclusion statements between XML queries. Query answering is implemented based on practical (but not complete) algorithms for XML query containment and rewriting. The scalability of Piazza so far does not go up to more than about 80 peers in the published experiments and relies on a wide range of optimizations (mappings composition (46), paths pruning (65)), made possible by the centralized storage of all the schemata and mappings in a global server.

Peer-to-Peer Inference Systems (P2PISs)

P2PISs are systems in which each peer can reason from its local theory but can also distribute reasoning among other peers with which it shares part of its vocabulary. This framework encompasses several applications like peer-to-peer information integration systems or intelligent agents, in which each peer has its own knowledge (about its data or its expertise domain) and some partial knowledge about some other peers. In this setting, when solicited to perform a reasoning task, a peer, if it cannot solve completely that task locally, must be able to distribute appropriate reasoning subtasks among its acquainted peers. This leads to a step by step splitting of the initial task among the peers that are relevant to solve parts of it. The outputs of the different split tasks must then be recomposed to construct the outputs of the initial task.

SomeWhere (13) is an example of a P2PIS. In SomeWhere, the local theory of each peer is composed of a set of propositional clauses defined upon a set of propositional variables (called its local vocabulary). Each peer may share part of its vocabulary with some other peers by establishing *mappings* with other peers's vocabularies. The *global theory* is considered as the union of all peer local theories and mappings. The reasoning task in SomeWhere is to find consequences, with

respect to (w.r.t.) the global theory, for a given input formula expressed using the local vocabulary of a peer. The challenge of this reasoning task is that no peer knows the global theory : each peer only knows its local theory and has little bit of knowledges with its acquainted peers through its mappings . To overcome this challenge, SomeWhere implements DeCA : Decentralized Consequence finding Algorithm (13) which performs this consequence finding task. DeCA is sound, complete and terminates. Moreover, it has been shown in (12) that DeCA scales up to 1000 peers.

Semantic Peer-to-Peer Systems

By 'semantic peer-to-peer systems' we refer to fully decentralized overlay networks of people or machines (called peers) sharing and searching varied resources (documents, videos, photos, data, services) based on their semantic annotations using ontologies. Ontologies are a formalization of the semantics of application domains (e.g., tourism, biology, medicine) through the definition of classes and relations modeling the domain objects and properties that are considered as meaningful for the application. In our view, the use of semantic annotations and ontologies when describing and structuring data makes the difference between a semantic peer-to-peer system with a PDMS (as the case of Piazza).

Different semantic peer-to-peer systems have been proposed. They differ mainly on the expressive power of their underlying data model and the way the different peers are semantically connected.

Edutella (50) is made of a network of super-peers, the topology of which is a hypercube. Super-peers are mediators with the same schema : a reference ontology (e.g., http://demoz.org). The data sources of a super-peer are its connected peers. Therefore, data is distributed over the peers while the ontologies are distributed over the super-peers. A peer must annotate its data in terms of the ontology of the super-peer to which it is connected. To answer queries, there is no need of mappings between the super-peer ontologies since they are identical: queries are efficiently routed in the network, using its topology of hypercube, in order to find super-peers that can provide answers with their peers.

In GridVine (8), the peers are organized according to a distributed hash table using Chord (62). As in Edutella, such a fixed structure allows efficient routing of messages between peers. On each GridVine peer, data is annotated with an RDFS (7) ontology and mappings with ontologies of other peers are stated by equivalences between properties of peer ontologies.

SomeOWL (13) is a semantic peer-to-peer system in which peers are not organized according to a fixed topology (as in Edutella or GridVine) : the topology is induced by the mappings between the peers ontologies. There is no super-peers : all the peers in SomeOWL are equivalent in functionality. SomeOWL is based on a simple data model : the ontologies and mappings are expressed in a fragment of OWL-DL that corresponds to the CLU description logic $(\neg, \sqcap, \sqcup)$ (19). Query answering in SomeOWL takes into account the ontologies and is achieved using a rewrite and evaluate strategy. In fact, SomeOWL is built on top of SomeWhere (13) : through some propositional encoding of the peer ontologies and mappings into propositional theories and clauses, the rewriting part of a query answering process is reduced to a consequence finding problem in distributed propositional theories. As mentioned above, this task is performed by the DeCA algorithm.

Like SomeOWL, SomeRDFS (14) is also a semantic peer-to-peer system that is built on top of SomeWhere. In SomeRDFS, the ontologies of peers and mappings between peers are expressed using the core fragment of RDFS allowing to state (sub)classes, (sub)properties, and to type the domain and range of properties. A mapping is an inclusion statement between classes or properties of two distinct peers, or a typing statement of a property of a given peer with a class of another peer. Like in SomeOWL, there is no super-peers and the topology of a SomeRDFS system is induced by the mappings between the peer ontologies. Built on top of SomeWhere, query answering in SomeRDFS is achieved using a rewrite and evaluate strategy, in which the corresponding rewriting problem is also reduced to the consequence finding problem in distributed propositional theories.

Problems studied in this thesis

The first problem studied in this thesis is handling inconsistencies in peer-to-peer inference systems.
The second one is the problem of modeling and handling trust in semantic peer-to-peer systems.

Our Contributions

1. **Handling inconsistencies in** SomeWhere

   Dealing with inconsistencies has been intensively studied for centralized clausal theories. In our context, the global theory is decentralized and no peer has the complete knowledge of it. That raises new algorithmic issues.

   Our **first contribution** is to propose an *incremental* and *decentralized* approach to detect causes of inconsistencies at each arrival of a new peer. We have designed and implemented the P2P-NG algorithm and we have proved that it is sound, complete and terminates.

   Our **second contribution** is a mechanism which relies on the detected causes of inconsistencies for reasoning in a well-founded way despite the existence of inconsistencies. Well-founded answers are those that are inferred from a consistent sub-theory. We have designed and implemented the WF-DeCA algorithm and we have proven its termination and its soundness.

   For experimenting P2P-NG and WF-DeCA, we have adapted the DeCA algorithm to produce only consequences whose length is bound by an input parameter. This new algorithm is called DeCAbL. We have run experiments on DeCAbL and obtained promising results that can be transposed to P2P-NG and WF-DeCA.

2. **Modeling trust in semantic peer-to-peer systems**

   Our **first contribution** is a probabilistic model to compute the trust of a peer towards the answers it receives based on their semantic annotation.

   Our **second contribution** is the application of this model to SomeOWL, resulting in adapting the DeCA algorithm to make possible the computation of trust *during* query processing.

Structure of this thesis

Firstly, Chapter 2 provides the preliminaries concerning SomeWhere and SomeOWL that are required for a good understanding of our contributions. Then, the contributions of this thesis are presented in two parts corresponding to the two problems that we have studied.

1. **PART 1 : Reasoning with inconsistencies in peer-to-peer inference systems**

   (a) Chapter 3 is a survey of the main approaches in Artificial Intelligence and Databases for dealing with inconsistencies.

   (b) Chapter 4 describes our contributions to the problem of reasoning with inconsistencies in a peer-to-peer inference system such as SomeWhere. We have designed and implemented two algorithms, P2P-NG and WF-DeCA, and we have proved their soundness, termination and completeness. An optimization of DeCA and results of experimentations are also described.

2. **PART 2 : A probabilistic model of trust for semantic peer-to-peer systems**

   (a) Chapter 5 summarizes the main existing approaches for modeling trust in peer-to-peer systems and related domains such as e-commerce.

   (b) Chapter 6 describes the probabilistic model of trust that we propose for semantic peer-to-peer systems and its application to SomeOWL.

Chapter 7 summarizes the contributions of this thesis and sketches some of its perspectives.

Chapter 2

PRELIMINARIES

In this chapter we specify the framework of the thesis. We present SomeWhere (13), a peer-to-peer inference system computing consequences of a given input clause w.r.t. distributed propositional theories in a totally decentralized manner. Then we present the SomeOWL (13) semantic peer-to-peer system. SomeOWL is built on top of SomeWhere, taking advantage of the DeCA algorithm implemented in SomeWhere for query answering.

2.1 SomeWhere

The SomeWhere (13) peer-to-peer inference system (P2PIS) is a network of peer theories. Each peer P is a finite set of propositional formulas of a language $\mathcal{L}_P$, which is the language of clauses without duplicated literals that can be built from a finite set of propositional variables $\mathcal{V}_P$, called the *vocabulary* of P. In SomeWhere, peers can be semantically related by sharing variables with others. A *shared variable* between two peers is in the intersection of the vocabularies of them. However it is not imposed that all the variables in common in the vocabularies of two peers are shared by them: they may not be aware of all the variables they have in common but only of some of them.

The architecture of SomeWhere is totally **decentralized**: there is no super-peers, all the peers are equivalent in functionality and no peer has the knowledge of the global P2PIS theory. Each peer only knows its own local theory and the

variables that it shares with some other peers of the P2PIS (its *acquaintances*). When a new peer joins a P2PIS, it simply declares its acquaintances in the P2PIS, i.e., the peers it knows to be sharing variables with, and it declares the corresponding shared variables.

2.1.1 Syntax and semantics

A SomeWhere P2PIS can be formalized using the notion of *acquaintance graph*.

Definition 1 *Acquaintance graph*
Let $\mathcal{P} = \{P_i\}_{i=1..n}$ *be a collection of clausal theories on their respective vocabularies* $\mathcal{V}_{P_i}$*, let* $\mathcal{V} = \cup_{i=1..n}\mathcal{V}_{P_i}$*. An acquaintance graph over* $\mathcal{V}$ *is a graph* $\Gamma = (\mathcal{P}, \text{ACQ})$ *where* $\mathcal{P}$ *is the set of vertices and* $\text{ACQ} \subseteq \mathcal{V} \times \mathcal{P} \times \mathcal{P}$ *is a set of labeled edges such that for every* $(v, P_i, P_j) \in \text{ACQ}$*,* $i \neq j$ *and* $v \in \mathcal{V}_{P_i} \cap \mathcal{V}_{P_j}$*.*

A labeled edge (v, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the variable v. For a peer P and a literal l, $\text{ACQ}(l, P)$ denotes the set of peers sharing with P the variable of l.

A SomeWhere P2PIS is interpreted using the standard semantics of propositional logic.

Definition 2 *Semantics of a* SomeWhere *P2PIS*
Let $\Gamma = (\mathcal{P}, \text{ACQ})$ *be a P2PIS with* $\mathcal{P} = \{P_i\}_{i=1..n}$*,*

- *An* interpretation *I of $\mathcal{P}$ is an assignment of the variables in $\bigcup_{i=1..n} P_i$ to true or false. In particular, a variable which is common to two theories P_i and P_j of a given P2PIS is interpreted by the same value in the two theories.*
- *I is a* model of a clause *c iff one of the literals of c is evaluated to true in I.*
- *I is a* model of a set of clauses *(i.e., a local theory, a union of a local theories, or a whole P2PIS) iff it is a model of all the clauses of the set.*
- *A P2PIS is* satisfiable *iff it has a model.*

- *The* consequence relation *for a P2PIS is the standard consequence relation* $\models$*: given a P2PIS* $\mathcal{P}$*, and a clause* c*,* $\mathcal{P} \models c$ *iff every model of* $\mathcal{P}$ *is a model of* c*.*

2.1.2 The consequence finding problem

For each theory P, a subset of *target variables* $\mathcal{TV}_P \subseteq \mathcal{V}_P$ is supposed to represent the variables of interest for the application, (e.g. observable facts in a model-based diagnosis application, or classes storing data in an information integration application). The goal is, given a clause provided as an input to a given peer, to find all the possible consequences belonging to some *target language* of the input clause and the union of the peer theories.

The interesting point in the consequence finding process in SomeWhere is that the input clause only uses the vocabulary of the queried peer, but that its expected consequences may involve target variables of different peers. The target languages handled by SomeWhere are defined in terms of target variables and require that a shared variable has the same target status in all the peers sharing it. This requirement is a local property : the peers sharing variables with a given peer are its acquaintances and, by definition, they are its direct neighbors in the acquaintance graph.

Definition 3 *Target Language*
Let $\Gamma = (\mathcal{P}, \text{ACQ})$ *be a P2PIS, and for every peer* P*, let* $\mathcal{TV}_P$ *be the set of its target variables such that if* $(v, P_i, P_j) \in \text{ACQ}$ *then* $v \in \mathcal{TV}_{P_i}$ *iff* $v \in \mathcal{TV}_{P_j}$*. For a set* SP *of peers of* $\mathcal{P}$*, we define its* target language $\mathcal{T}arget(SP)$ *as the language of clauses (including the empty clause) involving only variables of* $\bigcup_{P \in SP} \mathcal{TV}_P$*.*

The **reasoning problem** that is considered is to compute logical consequences of an input clause given a P2PIS. It corresponds to the notion of proper prime implicates of a clause w.r.t. a clausal distributed theory, which is formally defined in Definition 4.

Definition 4 *Proper prime implicate of a clause w.r.t. a clausal theory*
Let P *be a clausal theory and* q *be a clause. A clause* m *is said to be:*

- *an* implicate *of* q *w.r.t.* P *iff* $P \cup \{q\} \models m$.
- *a* prime implicate *of* q *w.r.t.* P *iff* m *is an implicate of* q *w.r.t.* P, *and for any other clause* m' *implicate of* q *w.r.t.* P, *if* $m' \models m$ *then* $m' \equiv m$.
- *a* proper prime implicate *of* q *w.r.t.* P *iff it is a prime implicate of* q *w.r.t.* P *but* $P \not\models m$.

Definition 5 *The consequence finding problem in a P2PIS*
Let $\Gamma = (\mathcal{P}, \text{ACQ})$ *be a P2PIS, where* $\mathcal{P} = \{P_i\}_{i=1..n}$ *is a collection of clausal theories with respective target variables. The* consequence finding problem *in* Γ *is, given a peer* P, *its acquaintances in* Γ, *and a clause* $q \in \mathfrak{L}_P$, *to find the set of proper prime implicates of* q *w.r.t.* $\bigcup_{i=1..n} P_i$ *which belong to* $\mathcal{T}arget(\mathcal{P})$.

2.1.3 Example

The following example illustrates a SomeWhere P2PIS consisting of 4 peers:

- P_1 describes a tour operator. Its theory expresses that its current Far destinations are either $Chile$ or $Kenya$. These far destinations are international destinations (Int) and expensive (Exp).
- The peer P_2 is only concerned with police regulations and expresses that a passport is required ($Pass$) for international destinations.
- P_3 focuses on sanitary conditions for travelers. It expresses that, in Kenya, yellow fever vaccination ($YellowFev$) is strongly recommended and that a strong protection against paludism should be taken ($Palu$) when accommodation occurs in $Lodges$.
- P_4 describes travel accommodation conditions : Lodge for Kenya and $Hotel$ for Chile. It also expresses that when anti-paludism protection is required, accommodations are equipped with appropriate anti-mosquito protections ($AntiM$).

The respective theories of each peer are described on Figure 2.1 as the nodes of the acquaintance graph. Shared variables are mentioned as edge labels. Target variables are defined by : $\mathcal{TV}_{P_1} = \{Exp\}$, $\mathcal{TV}_{P_2} = \{Pass\}$, $\mathcal{TV}_{P_3} = \{Lodge, YellowFev, Palu\}$ and $\mathcal{TV}_{P_4} = \{Lodge, Hotel, Palu, AntiM\}$.

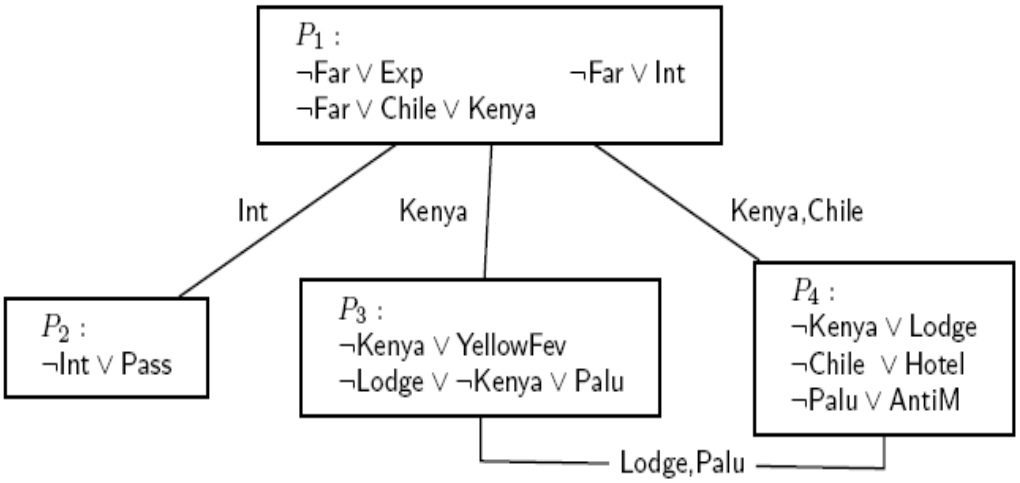

Figure 2.1: Acquaintance graph for the tour operator example

2.2 DeCA : Decentralized Consequence Finding Algorithm

The DeCA algorithm is the first consequence finding algorithm for decentralized propositional theories. It has been shown in (12) that DeCA scales up to at least 1000 peers.

There are two versions of DeCA : a recursive version and a message-passing version. The message-passing version of DeCA is the implemented one in SomeWhere. It has been shown in (13) that the recursive version and the message-passing version of DeCA are equivalent. The soundness and completeness of DeCA have been proved on its recursive version. As our algorithms P2P-NG and WF-DeCA for handling inconsistencies are based on DeCA, we recall the two versions of DeCA.

We will use the following notations :

- for a literal q, $Resolvent(q, P)$ denotes the set of clauses obtained by resolution from the set $P \cup \{q\}$ but not from P alone. We call such clauses *proper resolvents of* q *w.r.t.* P,

- *for a literal q, $\bar{q}$ denotes its complementary literal,*
- *for a clause c of a peer P, $S(c)$ (resp. $L(c)$) denotes the disjunction of literals of c whose variables are shared (resp. not shared) with some acquaintance of P. The condition $S(c) = \square$ thus expresses that c does not contain any variable shared with an acquaintance of P,*
- *a history hist is a sequence of triples (l, P, c) (where l is a literal, P a peer, and c a clause). An history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, which either contains the clause $\neg l_0 \vee c_0$ (in that case c_0 may have been splitted into its different literals among which l_1 is propagated in P_1), or not (in that case $l_0 = c_0$ and l_0 is propagated into P_1, and thus $l_0 = l_1$): for every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1},*
- *$\otimes$ is the distribution operator on sets of clauses: $S_1 \otimes \cdots \otimes S_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, we will use the notation $\otimes_{l \in L} S_l$ to denote $S_{l_1} \otimes \cdots \otimes S_{l_p}$.*

Recursive version of DeCA

Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a P2PIS, P one of its peers, and q a literal whose variable belongs to the vocabulary of P. The entry point of DeCA is a function called RCF. $RCF(q, P)$ computes implicates of the literal q w.r.t. $\mathcal{P}$, starting with the computation of local consequences of q, i.e., implicates of q w.r.t. P, and then recursively following the acquaintances of the visited peers. To ensure termination, it is necessary to keep track of the literals already processed by peers. This is done by the recursive algorithm $RCFH(q, SP, hist)$, where $hist$ is the history of the reasoning branch ending up to the propagation of the literal q in SP, which is the set of acquaintances of the last peer added to the history.

It has been shown in (13) that DeCA terminates, is sound and complete.

Algoritm 1: Recursive consequence finding algorithm
$RCF(q, P)$
(1) **return** $RCFH(q, \{P\}, \emptyset)$

$RCFH(q, SP, hist)$
(1) **if** there exists $P \in SP$ s.t. $q \in P$ or if for every $P \in SP$, $(q, P, \_) \in hist$ **return** $\emptyset$
(2) **else if** $(\bar{q}, \_, \_) \in hist$ **return** $\{\Box\}$
(3) **else** for every $P \in SP$ $\text{LOCAL}(P) \leftarrow \{q\} \cup Resolvent(q, P)$
(4) **if** there exists $P \in SP$ s.t. $\Box \in \text{LOCAL}(P)$ **return** $\{\Box\}$
(5) **else** for every $P \in SP$ $\text{LOCAL}(P) \leftarrow \{c \in \text{LOCAL}(P) | L(c) \in \mathcal{T}arget(P)\}$
(6) **if** for every $P \in SP$ and for every $c \in \text{LOCAL}(P)$, $S(c) = \Box$, **return** $\bigcup_{P \in SP} \text{LOCAL}(P)$
(7) **else**
(8) RESULT $\leftarrow \bigcup_{P \in SP} \{c \in \text{LOCAL}(P) | S(c) \in \mathcal{T}arget(P)\}$
(9) **foreach** $P \in SP$ and $c \in \text{LOCAL}(P)$ s.t. $S(c) \neq \Box$
(10) **if** $\neg q \vee c \in P$, $P' \leftarrow P \backslash \{\neg q \vee c\}$
(11) **foreach** literal $l \in S(c)$
(12) ANSWER$(l) \leftarrow RCFH(l, \text{ACQ}(l, P), [(q, P, c) | hist])$
(13) DISJCOMB $\leftarrow (\otimes_{l \in S(c)} \text{ANSWER}(l)) \otimes \{L(c)\}$
(14) RESULT $\leftarrow$ RESULT $\cup$ DISJCOMB
(15) **return** RESULT

Message-passing version of DeCA

The message-passing version of DeCA runs locally on each peer. It is composed of three procedures, each one being triggered by the reception of a message.

- The procedure ReceiveForthMessage is triggered by the reception of a $forth$ message $m(Sender, Receiver, forth, hist, l)$ sent by the peer *Sender* to the peer *Receiver* which executes the procedure: on the demand of *Sender*, with which it shares the variable of l, it processes the literal l.

- The procedure ReceiveBackMessage is triggered by the reception of a *back* message $m(Sender, Receiver, back, hist, r)$ sent by the peer *Sender* to

the peer *Receiver* which executes the procedure: it processes the consequence r (which is a clause the variables of which are target variables) sent back by *Sender* for the literal l (last added in the history) ; it may have to combine it with other consequences of literals being in the same clause as l.

- The procedure ReceiveFinalMessage is triggered by the reception of a $final$ message $m(Sender, Receiver, final, hist, true)$: the peer *Sender* notifies the peer *Receiver* that computation of the consequences of the literal l (last added in the history) is completed.

Those procedures handle two data structures stored at each peer: cons$(l, hist)$ caches the consequences of l computed by the reasoning branch corresponding to $hist$; final$(q, hist)$ is set to true when the propagation of q within the reasoning branch of the history $hist$ is completed.

The reasoning is initiated by the user (denoted by a particular peer $User$) sending to a given peer P a message $m(User, P, forth, \emptyset, q)$. This triggers on the P the local execution of the procedure ReceiveForthMessage$(m(User, P, forth, \emptyset, q))$. In the description of the procedures, since they are locally executed by the peer which receives the message, $Self$ is used to denote this receiver peer.

DeCA is anytime : the consequences of a query are returned as a stream and the order of their returning is unpredictable. The important point is that DeCA notifies its termination.

Algoritm 2: Message passing procedure for propagating literals forth
ReceiveForthMessage($m(Sender, Self, forth, hist, q)$)

(1) **if** $(\bar{q}, \_, \_) \in hist$
(2) **send** $m(Self, Sender, back, [(q, Self, \Box)|hist], \Box)$
(3) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(4) **else if** $q \in Self$ or $(q, Self, \_) \in hist$
(5) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(6) **else**
(7) $\text{LOCAL}(Self) \leftarrow \{q\} \cup Resolvent(q, Self)$
(8) **if** $\Box \in \text{LOCAL}(Self)$
(9) **send** $m(Self, Sender, back, [(q, Self, \Box)|hist], \Box)$
(10) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(11) **else**
(12) $\text{LOCAL}(Self) \leftarrow \{c \in \text{LOCAL}(Self) |\ L(c) \in \mathcal{T}arget(Self)\}$
(13) **if** for every $c \in \text{LOCAL}(Self), S(c) = \Box$
(14) **foreach** $c \in \text{LOCAL}(Self)$
(15) **send** $m(Self, Sender, back, [(q, Self, c)|hist], c)$
(16) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(17) **else**
(18) **foreach** $c \in \text{LOCAL}(Self)$
(19) **if** $S(c) = \Box$
(20) **send** $m(Self, Sender, back, [(q, Self, c)|hist], c)$
(21) **else**
(22) **foreach** literal $l \in S(c)$
(23) **if** $l \in \mathcal{T}arget(Self)$
(24) $\text{CONS}(l, [(q, Self, c)|hist]) \leftarrow \{l\}$
(25) **else**
(26) $\text{CONS}(l, [(q, Self, c)|hist]) \leftarrow \emptyset$
(27) $\text{FINAL}(l, [(q, Self, c)|hist]) \leftarrow false$
(28) **foreach** $RP \in \text{ACQ}(l, Self)$
(29) **send** $m(Self, RP, forth, [(q, Self, c)|hist], l)$

Algoritm 3: Message passing procedure for processing the return of consequences

ReceiveBackMessage($m(Sender, Self, back, hist, r)$)

(1) $hist$ is of the form $[(l', Sender, c'), (q, Self, c)|hist']$

(2) $\text{CONS}(l', hist) \leftarrow \text{CONS}\ (l', hist) \cup \{r\}$

(3) $\text{RESULT} \leftarrow \bigotimes_{l \in S(c) \setminus \{l'\}} \text{CONS}(l, hist) \otimes \{L(c) \vee r\}$

(4) **if** $hist' = \emptyset$, $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$

(5) **foreach** $cs \in$ RESULT

(6) **send** $m(Self, U, back, [(q, Self, c)|hist'], cs)$

Algoritm 4: Message passing procedure for notifying termination

ReceiveFinalMessage($m(Sender, Self, final, hist, true)$)

(1) $hist$ is of the form $[(l', Sender, true), (q, Self, c)|hist']$

(2) $\text{FINAL}(l', hist) \leftarrow true$

(3) **if** for every $l \in S(c)$, $\text{FINAL}(l, hist) = true$

(4) **if** $hist' = \emptyset$ $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$

(5) **send** $m(Self, U, final, [(q, Self, true)|hist'], true)$

(6) **foreach** $l \in S(c)$

(7) $\text{CONS}(l, [(l, Sender, \_), (q, Self, c)|hist']) \leftarrow \emptyset$

Example

We now illustrate the behavior of the algorithm on the example of Figure 2.1, when the input clause Far is provided to the peer P_1 by the user. We present the propagation of the reasoning as a tree structure, the nodes of which correspond to peers and the branches of which materialize the different reasoning paths induced by the initial input clause. Edges are labeled on the left side by literals which are propagated along paths and/or on the right side by consequences that are transmitted back. A downward arrow on an edge indicates the step during which a literal is propagated from one peer to its neighbor. For instance, the initial step can be represented here by the tree in Figure 2.2.

Local consequences of a literal propagated on a peer are then made explicit within the peer node. Target literals are outlined using a grey background, as well as transmitted back consequences. Vertical arrows preceding consequences distinguish the last returned consequences from earlier ones. Although the successive

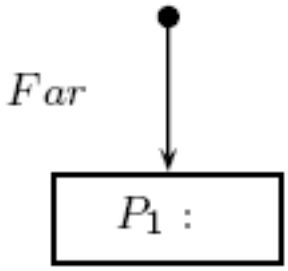

Figure 2.2: The input clause Far is provided to the peer P_1

trees presented here have increasing depth, as if all reasoning paths were explored synchronously and in parallel, all messages are in fact exchanged in an asynchronous way and that the order in which consequents are produced cannot be predicted.

In this example, (see Figure 2.3) consequences of Far derived by local reasoning on P_1 are Exp, Int and Chile $\vee$ Kenya. Since Exp is in $\mathcal{T}arget(P_1)$ it is a local consequence of Far. Int is not a target literal but is shared with P_2, it is therefore transmitted to P_2. The clause Chile $\vee$ Kenya is also made of shared variables. Such clauses are processed by the algorithm using a split/recombination approach. Each shared literal is processed independently, and transmitted to its appropriate neighbors. Each literal is associated with some queue data structure, where transmitted back consequences are stored. As soon as at least one consequent has been obtained for each literal, the respective queued consequents of each literal are recombined, to produce consequences of the initial clause. This recombination process continues, as new consequences for a literal are produced. Note that since each literal is processed asynchronously, the order in which the recombined consequences are produced is unpredictable. Here, the component Chile is transmitted to P_4 and Kenya is transmitted to P_3 and P_4. Let us note that the peer P_4 appears two times in the tree, because two different literals are propagated on this peer, which induces two different reasoning paths.

While Exp is transmitted back to the user as a first (*local*) consequence of Far, (see Figure 2.4)

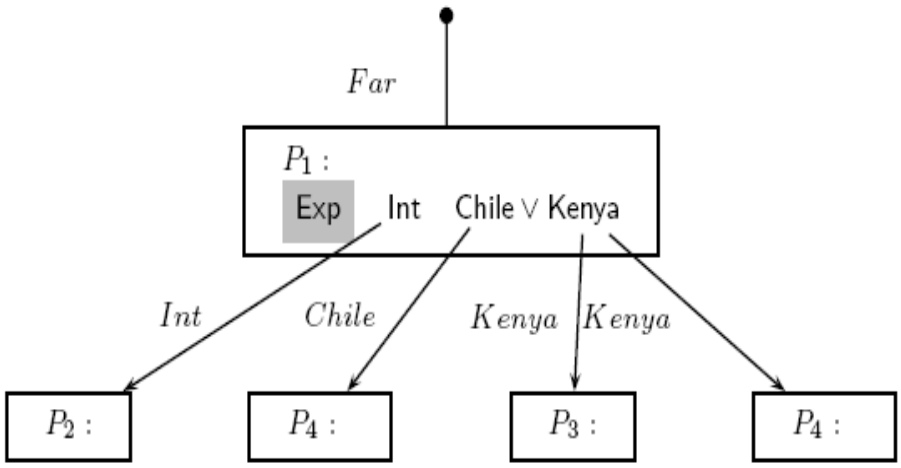

Figure 2.3: Finding consequences of Far using DeCA

- The propagation of Int on P_2 produces the clause Pass, which is in $\mathcal{T}arget(P_2)$ but is not shared and therefore, cannot be further propagated.
- The clause Chile, when transmitted to P_4, produces Hotel which is in $\mathcal{T}arget(P_4)$ but is not shared and cannot be further propagated.
- When transmitted to P_3, the clause Kenya produces YellowFev as well as the clause ¬Lodge ∨ Palu. The three variables are in $\mathcal{T}arget(P_3)$. Lodge and Palu are also shared variables and therefore, after splitting of the second clause, their corresponding literals are transmitted (independently) to P_4.
- When transmitted to P_4, Kenya produces Lodge, which is in $\mathcal{T}arget(P_4)$ and is also shared and therefore further transmitted to P_3.
- The clause Pass, produced on P_2, is transmitted back to P_1 as a consequence of Int and then to the user as a *remote consequence* of Far.
- The clause Hotel, produced on P_4, is transmitted back to P_1 where it is queued as a consequent of Chile, since it has to be combined with consequences of Kenya.

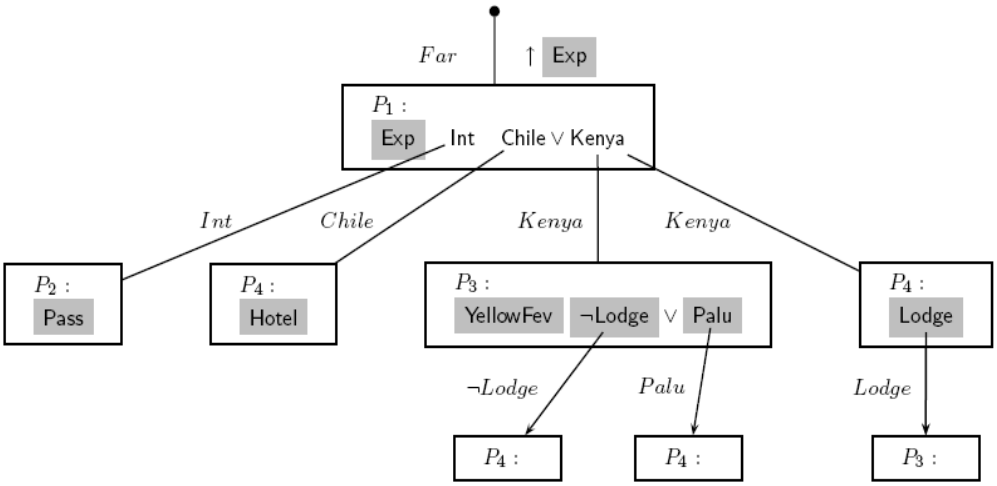

Figure 2.4: Finding consequences of Far using DeCA (cont)

- The two local consequences of Kenya obtained on P_3 contain only target variables. They are transmitted back to P_1 and queued there. They may now be combined with Hotel to produce two new *combined consequences* of Far : Hotel $\vee$ YellowFev and Hotel $\vee$ $\neg$Lodge $\vee$ Palu, which are transmitted back to the user.

- Similarly on P_4, Lodge is a local target consequent of Kenya, that is transmitted back to P_1 as a consequent of Kenya, where it is combined with Hotel to produce a new consequence of Far that, in turn, is transmitted back to the user.

Simultaneously, (see Figure 2.5) the reasoning further propagates in the network of peers. The propagation of $\neg$Lodge and Palu on P_4 respectively produces $\neg$Kenya, which is not a target literal but is shared and thus further propagated on P_1, as well as AntiM, which is a target literal, but not shared. We do not detail here the propagation of Lodge in the right most branch of the reasoning tree.

Note (see Figure 2.6) on the deepest node that P_1 is here asked to produce the implicates of $\neg$Kenya, while the complementary literal Kenya is still under process. Such situations are handled in the algorithm by mean of histories keeping track

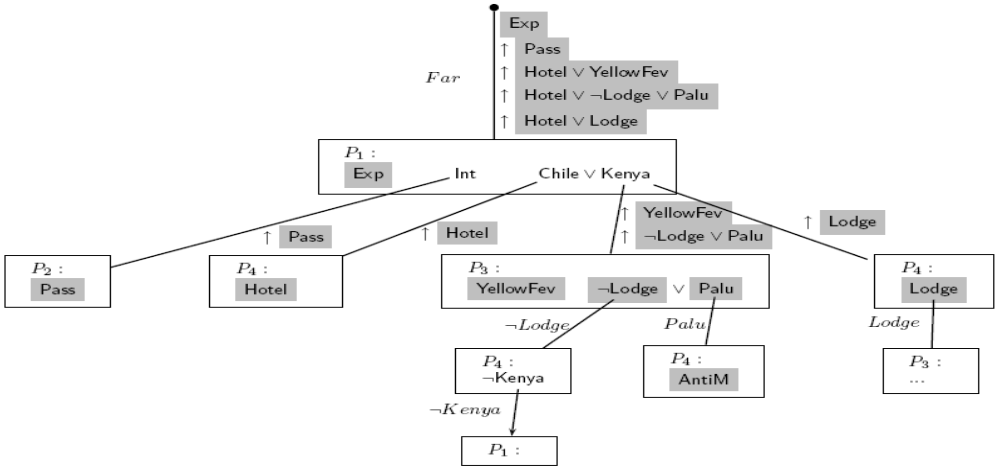

Figure 2.5: Finding consequences of Far using DeCA (cont)

of the reasoning branches ending up to each transmitted literal. When a same history contains two complementary literals, the corresponding reasoning branch is closed and the empty clause $\square$ is returned as a consequence of the literals in the history.

In this example, the consequence produced by P_1 for $\neg$Kenya is thus $\square$, which is sent back to P_4 and then to P_3. After combination on P_3 with Palu we thus obtain Palu as a new consequent of Kenya, which subsumes the previously obtained $\neg$Lodge $\vee$ Palu. When transmitted back to P_1 and combined with Hotel we obtain Hotel $\vee$ Palu which subsumes the previously obtained consequent Hotel $\vee \neg$Lodge $\vee$ Palu. Since AntiM is not a shared variable it is the only consequent of Palu on P_4. When transmitted back to P_3 for combination with $\square$, we thus obtain AntiM which, in turn, is returned to P_1 for combination with Hotel, thus giving Hotel $\vee$ AntiM as a new consequent of Far.

We have not detailed the production of consequences of Lodge on P_3 in the right most branch, but it can be verified that it also produces AntiM (which has already been produced through P_3/P_4 in the third branch). Eventually, the whole set of consequences of Far is {Exp, Pass, Hotel $\vee$ Lodge, Hotel $\vee$ Palu, Hotel $\vee$ AntiM, Hotel $\vee$ YellowFev}. Among those consequences, it is important to note that some of them

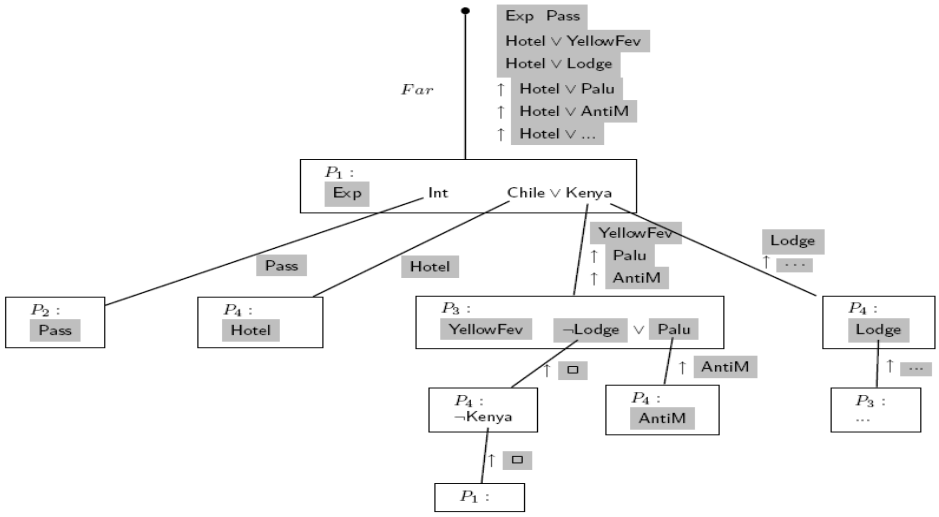

Figure 2.6: Finding consequences of Far using DeCA (cont)

(e.g., Hotel∨YellowFev) involve target variables from different peers. This is made possible thanks to the split/recombination strategy of the algorithm.

2.3 The SomeOWL semantic peer-to-peer system

SomeOWL is a semantic peer-to-peer system that has been deployed on top of the SomeWhere platform (11). In SomeOWL, data is a set of resources identified by URIs. Each peer categorizes its local resources (documents, files, web pages) by declaring the corresponding URIs as instances of some class(es) of its own ontolgoy.

A new peer joins the network through some peers that it knows (its acquaintances) by declaring mappings between its own ontology and the ontologies of its acquaintances. Queries are posed to a given peer using its local ontology. The answers that are expected are not only instances of local classes but possibly instances of classes of peers distant from the queried peer if it can be inferred,

from the peer ontologies and the mappings that they satisfy the query.

Local ontologies, storage descriptions and mappings in SomeOWL are defined using a fragment of OWL DL which is the description logic fragment of the Ontology Web Language recommended by W3C.

Each peer ontology is made of a set of class definitions and possibly a set of equivalence, inclusion or disjointness axioms between class descriptions. A class description is either the universal class ($\top$), the empty class ($\bot$), an atomic class or the union ($\sqcup$), intersection ($\sqcap$) or complement ($\neg$) of class descriptions. The name of atomic classes is unique to each peer: the notation $P{:}A$ is used for identifying an atomic class A of the ontology of a peer P. The *vocabulary* of a peer P is the set of names of its atomic classes. The following table summarizes the class descriptions and the axioms that are allowed for defining ontologies in SomeOWL.

Class descriptions

	Logical notation	*OWL notation*
universal class	$\top$	$Thing$
empty class	$\bot$	$Nothing$
atomic class	$P{:}A$	$classID$
conjunction	$D1 \sqcap D2$	$intersectionOf(D1\ D2)$
disjunction	$D1 \sqcup D2$	$unionOf(D1\ D2)$
negation	$\neg D$	$complementOf(D)$

Axioms on class descriptions

	Logical notation	*OWL notation*
equivalence	$D1 \equiv D2$	$EquivalentClasses(D1\ D2)$
inclusion	$D1 \sqsubseteq D2$	$SubClassOf(D1\ D2)$
disjointness	$D1 \sqcap D2 \equiv \bot$	$DisjointClasses(D1\ D2)$

The following table summarizes the storage descriptions allowed in SomeOWL. The axioms defining the extensional classes are restricted to be inclusion statements between an atomic extensional class and a description combining atomic classes of the ontology. The notation $P{:}ViewA$ is used to denote an extensional class $ViewA$ of the peer P.

Storage description

declaration of extensional classes:

Logical notation	*OWL notation*
$P{:}ViewA \sqsubseteq C$	$SubClassOf(P{:}ViewA \;\; C)$

assertional statements:

Logical notation	*OWL notation*
$P{:}ViewA(a)$	$individual(a\ type(P{:}ViewA))$

Mappings are disjointness, equivalence or inclusion statements involving atomic classes of different peers. They express the semantic correspondence that may exist between the ontologies of different peers.

Let $\mathcal{P}$ be a SomeOWL peer-to-peer network made of a collection of peers $\{P_i\}_{i=1..n}$. For each peer P_i, let O_i, V_i and M_i be the sets of axioms defining respectively the local ontology of P_i, the declaration of its extensional classes and the set of mappings stated at P_i between classes of O_i and classes of the ontologies of the acquaintances of P_i. The schema of $\mathcal{P}$, denoted $S(\mathcal{P})$, is the union $\bigcup_{i=1..n} O_i \cup V_i \cup M_i$ of the ontologies, the declaration of extensional classes and of the sets of mappings of all the peers.

The semantics of the data model used in SomeOWL is the standard first-order logic semantics defined in terms of *interpretations*. An interpretation I is a pair $(\Delta^I, .^I)$ where Δ is a non-empty set, called the domain of interpretation, and $.^I$ is an interpretation function which assigns a subset of Δ^I to every class identifier and an element of Δ^I to every data identifier.

An interpretation I is a *model* of the distributed schema of a SomeOWL peer-to-peer network $\mathcal{P} = \{P_i\}_{i=1..n}$ iff each axiom in $\bigcup_{i=1..n} O_i \cup V_i \cup M_i$ is satisfied by I. Interpretations of axioms rely on interpretations of class descriptions which are inductively defined as follows:

- $\top^I = \Delta^I$, $\bot^I = \emptyset$
- $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I$
- $(C_1 \sqcap C_2)^I = C_1^I \cap C_2^I$
- $(\neg C)^I = \Delta^I \backslash C^I$

Axioms are satisfied if the following holds:

- $C \sqsubseteq D$ is satisfied in I iff $C^I \subseteq D^I$
- $C \equiv D$ is satisfied in I iff $C^I = D^I$
- $C \sqcap D \equiv \bot$ is satisfied in I iff $C^I \cap D^I = \emptyset$

A SomeOWL peer-to-peer network is *satisfiable* iff its (distributed) schema has a model. Given a SomeOWL peer-to-peer network $\mathcal{P} = \{P_i\}_{i=1..n}$, a class description C *subsumes* a class description D iff for any model I of $S(\mathcal{P})$ $D^I \subseteq C^I$.

2.3.1 Illustrative example

We illustrate the SomeOWL data model on a small example of four peers modeling four persons Ann, Bob, Chris and Dora, each of them bookmarking URLs about restaurants they know or like, according to their own taxonomy for categorizing restaurants.

Ann, who is working as a restaurant critic, organizes its restaurant URLs according to the following classes:

- the class $Ann{:}G$ of restaurants considered as offering a "good" cooking, among which she distinguishes the subclass $Ann{:}R$ of those which are rated: $Ann{:}R \sqsubseteq Ann{:}G$
- the class $Ann{:}R$ is the union of three disjoint classes $Ann{:}S1$, $Ann{:}S2$, $Ann{:}S3$ corresponding respectively to the restaurants rated with $1, 2$ or 3 stars:

  $Ann{:}R \equiv Ann{:}S1 \sqcup Ann{:}S2 \sqcup Ann{:}S3$

  $Ann{:}S1 \sqcap Ann{:}S2 \equiv \bot \quad Ann{:}S1 \sqcap Ann{:}S3 \equiv \bot$

  $Ann{:}S2 \sqcap Ann{:}S3 \equiv \bot$
- the classes $Ann{:}I$ and $Ann{:}O$, respectively corresponding to Indian and Oriental restaurants
- the classes $Ann{:}C$, $Ann{:}T$ and $Ann{:}V$ which are subclasses of $Ann{:}O$ denoting Chinese, Taï and Vietnamese restaurants respectively: $Ann{:}C \sqsubseteq Ann{:}O$, $Ann{:}T \sqsubseteq Ann{:}O$, $Ann{:}V \sqsubseteq Ann{:}O$

Suppose that the data stored by Ann that she accepts to make available deals with restaurants of various specialties, and only with those rated with 2 stars among the rated restaurants. The extensional classes declared by Ann are then:

$Ann{:}ViewS2 \sqsubseteq Ann{:}S2$, $Ann{:}ViewC \sqsubseteq Ann{:}C$, $Ann{:}ViewV \sqsubseteq Ann{:}V$, $Ann{:}ViewT \sqsubseteq Ann{:}T$, $Ann{:}ViewI \sqsubseteq Ann{:}I$

Bob, who is fond of Asian cooking and likes high quality, organizes his restaurant URLs according to the following classes:

- the class $Bob{:}A$ of Asian restaurants
- the class $Bob{:}Q$ of high quality restaurants that he knows

Suppose that he wants to make available every data that he has stored. The extensional classes that he declares are $Bob{:}ViewA$ and $Bob{:}ViewQ$ (as subclasses of $Bob{:}A$ and $Bob{:}Q$): $Bob{:}ViewA \sqsubseteq Bob{:}A$, $Bob{:}ViewQ \sqsubseteq Bob{:}Q$

Chris is more fond of fish restaurants but recently discovered some places serving a very nice cantonese cuisine. He organizes its data with respect to the following classes:

- the class $Chris{:}F$ of fish restaurants,
- the class $Chris{:}CA$ of Cantonese restaurants

Suppose that he declares the extensional classes $Chris{:}ViewF$ and $Chris{:}ViewCA$ as subclasses of $Chris{:}F$ and $Chris{:}CA$ respectively:
$Chris{:}ViewF \sqsubseteq Chris{:}F$, $Chris{:}ViewCA \sqsubseteq Chris{:}CA$

Dora organizes her restaurants URLs around the class $Dora{:}DP$ of her preferred restaurants, among which she distinguishes the subclass $Dora{:}P$ of pizzerias and the subclass $Dora{:}SF$ of seafood restaurants.
Suppose that the only URLs that she stores concerns pizzerias: the only extensional class that she has to declare is $Dora{:}ViewP$ as a subclass of $Dora{:}P$: $Dora{:}ViewP \sqsubseteq Dora{:}P$

Ann, **Bob**, **Chris** and **Dora** express what they know about each other using mappings stating properties of class inclusion or equivalence.
Ann is very confident in Bob's taste and agrees to include Bob' selection as good restaurants by stating $Bob{:}Q \sqsubseteq Ann{:}G$. Finally, she thinks that Bob's Asian restaurants encompass her Oriental restaurant concept: $Ann{:}O \sqsubseteq Bob{:}A$
Bob knows that what he calls Asian cooking corresponds exactly to what Ann classifies as Oriental cooking. This may be expressed using the equivalence state-

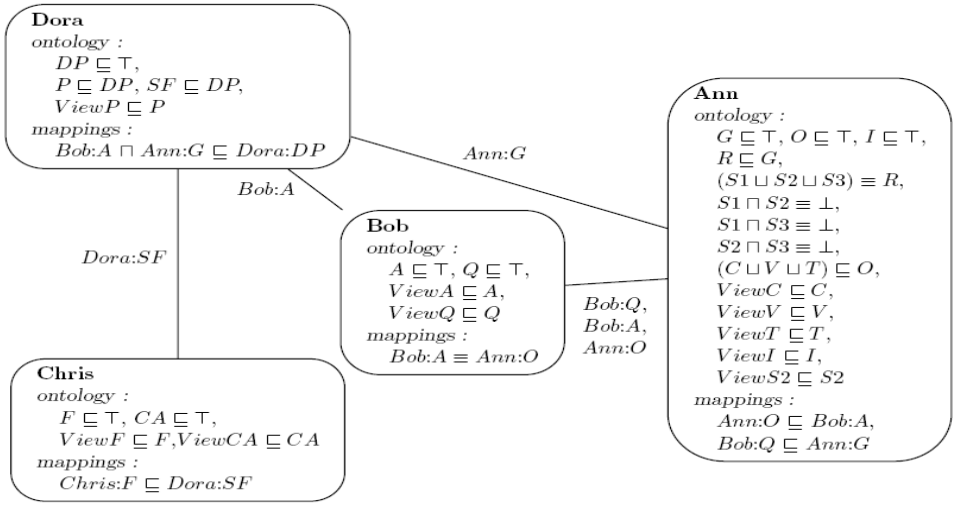

Figure 2.7: A SomeOWL network with peer ontologies and mappings

ment : $Bob{:}A \equiv Ann{:}O$ (note the difference of perception of Bob and Ann regarding the mappings between $Bob{:}A$ and $Ann{:}O$)
Chris considers that what he calls fish specialties is a particular case of Dora seafood specialties: $Chris{:}F \sqsubseteq Dora{:}SF$
Dora counts on both Ann and Bob to obtain good Asian restaurants : $Bob{:}A \sqcap Ann{:}G \sqsubseteq Dora{:}DP$

Figure 2.7 describes the resulting overlay network. In order to alleviate the notations, we omit the local peer name prefix except for the mappings. Edges are labeled with the class identifiers that are shared through the mappings between peers.

2.3.2 Query answering in SomeOWL

Query answering in SomeOWL is done by rewriting the query into a union of conjunctions of extensional classes, which are then evaluated.

Definition 6 *Rewritings*
Given a SomeOWL *peer-to-peer network* $\mathcal{P} = \{P_i\}_{i=1..n}$, *a conjunction* Q_e *of extensional classes is a* rewriting *of a query* Q *iff* Q *subsumes* Q_e *w.r.t.* $\mathcal{P}$.

Q_e *is a* proper rewriting *if there exists some model of* I *of* $S(\mathcal{P})$ *such that* $Q_e^I \neq \emptyset$. Q_e *is a* maximal (conjunctive) rewriting *if there does not exist another (conjunctive) rewriting* Q'_e *of* Q *(strictly) subsuming* Q_e *w.r.t.* $\mathcal{P}$.

A rewriting of a query is thus an intensional answer expressed in terms of extensional classes. Queries are conjunctions of classes of a given peer ontology. An answer is a resource for which it can be logically inferred from the axioms defining the ontologies, the mappings and the storages descriptions that the URI is an instance of the classes in the query.

From the query answering point of view, it is the notion of *proper* rewriting which is relevant because it guarantees a non empty set of answers. If a query has no proper rewriting, it won't get any answer.

In the SomeOWL setting, query rewriting can be equivalently reduced to distributed reasoning over logical propositional theories by a straightforward propositional encoding of the query and of the distributed schema of a SomeOWL network. It consists in transforming each query and schema statement into a propositional formula using class identifiers as propositional variables.

The propositional encoding of a class description D, and thus of a query, is the propositional formula $Prop(D)$ obtained inductively as follows:

- $Prop(\top) = true$, $Prop(\bot) = false$
- $Prop(A) = A$, if A is an atomic class
- $Prop(D_1 \sqcap D_2) = Prop(D_1) \wedge Prop(D_2)$
- $Prop(D_1 \sqcup D_2) = Prop(D_1) \vee Prop(D_2)$
- $Prop(\neg D) = \neg(Prop(D))$

The propositional encoding of the schema $\mathcal{S}$ of a SomeOWL network $\mathcal{P}$ is the distributed propositional theory $Prop(\mathcal{S})$ made of the formulas obtained inductively from the axioms in $\mathcal{S}$ as follows:

- $Prop(C \sqsubseteq D) = Prop(C) \Rightarrow Prop(D)$
- $Prop(C \equiv D) = Prop(C) \Leftrightarrow Prop(D)$
- $Prop(C \sqcap D \equiv \bot) = \neg Prop(C) \vee \neg Prop(D)$

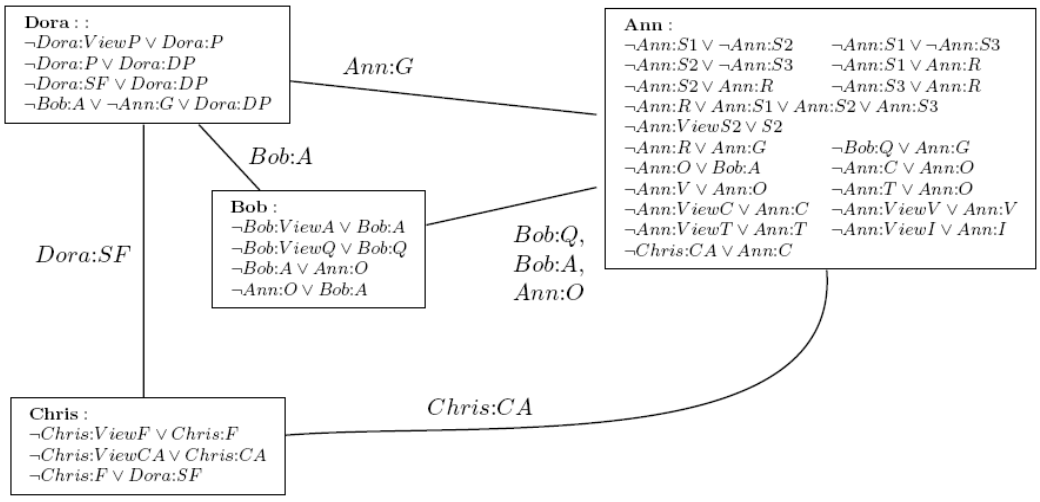

Figure 2.8: Propositional encoding of the SomeOWL network of the figure 2.7

Figure 2.8 shows the propositional encoding (in clausal form) of the SomeOWL network of Figure 2.7

It has been shown in (13) that query rewriting in SomeOWL can be reduced to consequence finding in SomeWhere: Proposition 1 states that the propositional encoding transfers satisfiability and establishes the connection between proper (maximal) conjunctive rewritings and proper (prime) implicates.

Proposition 1 *(13)*
Let $\mathcal{P}$ be a SomeOWL *peer-to-peer network and let $Prop(S(\mathcal{P}))$ be the propositional encoding of its schema. Let V_e be the set of all the extensional classes.*

- *$S(\mathcal{P})$ is satisfiable iff $Prop(S(\mathcal{P}))$ is satisfiable.*
- *q_e is a proper maximal conjunctive rewriting of a query q iff $\neg Prop(q_e)$ is a* proper prime implicate of $\neg Prop(q)$ w.r.t. *$Prop(S(\mathcal{P}))$ such that all its variables are extensional classes.*

Part I

REASONING WITH INCONSISTENCIES IN PEER-TO-PEER INFERENCE SYSTEMS

INTRODUCTION

In SomeWhere, a peer local theory is a set of clauses defined upon a set of propositional variables. These variables are the local vocabulary of the peer. It is supposed that each local peer theory is consistent.

When a new peer joins the network, it establishes mappings with other peers, called its acquaintances. Mappings are clauses involving variables of distinct peers that state semantic correspondences between different peer vocabularies. It is supposed that every peer can declare mappings to express its own viewpoint about the relation between its own theory and others, without the control of a central server, but with the agreement of the peer with which it declares the mapping.

In this context, the mappings, while relating peer theories, may create inconsistencies in the global theory of SomeWhere, which is unknown to any peer and is the union of all peer local theories and all the mappings between peer theories. When the global theory is inconsistent, query answering in SomeWhere may be meaningless.

In this part, we focus on the problem of handling in a decentralized way inconsistencies caused by mappings in SomeWhere. We divide this problem into two sub-problems : to detect causes of inconsistencies in the global theory, and to reason in SomeWhere in presence of inconsistencies but produce only meaningful answers of queries.

The structure of this part is as follows :

- in Chapter 3, we present a state of the art on the approaches for dealing with inconsistencies,

- in Chapter 4, we formally define the problem of handling inconsistencies that we consider in SomeWhere and we present our contributions for dealing with this problem.

  We propose two algorithms: the decentralized P2P-NG algorithm which is complete for detecting causes of inconsistencies; the decentralized WF-DeCA algorithm for computing only well-founded answers of queries, based on the results of P2P-NG.

  We propose an other algorithm, DeCAbL, which is an optimization of DeCA, computing only consequences the maximum length of which is predefined as an input parameter. We report the results of the experimentations that we have made on DeCAbL and we explain how these results can be transposed to both P2P-NG and WF-DeCA.

Chapter 3

STATE OF THE ART ON DEALING WITH INCONSISTENCIES

One of the most used definitions of the word 'inconsistency' in dictionaries such as Longman or Oxford is : a situation in which two statements are different and cannot both be true. Here, 'statements' can not only be understood as its first meaning, but also as 'actions', 'reactions', 'behaviors', 'thoughts', or 'expressions', 'point of views',...

Actually, inconsistency may happen everywhere and in every domain. For example, an individual may have an inconsistent behavior so that people get confused about his personality. In a family, the wife and the husband may have inconsistent opinions about how to raise their children. In a country, different parties have inconsistent points of view of how to develop the country, etc...

Inconsistency is often understood as **incompatibility** or **contradiction**, which are synonyms in fact: two contradictory (incompatible) statements cannot both be true. Drawing right conclusions by reasoning from inconsistent facts is problematic as illustrated by the following example (64).

Suppose a university payroll system says that Johns salary is 50K, while the university personnel database says it is 60K. In addition, there may be an axiom that

says that everyone has exactly one salary. One simple way to model this is via the theory, denoted as sal^{base}, below.

1. *John's salary is 50K*
2. *John's salary is 60K*
3. *Everyone has exactly one salary*

This sal^{base} is obviously inconsistent. Suppose (3) is definitely known to be true. Then a bank manager considering John for a loan may choose the 50K number to determine a maximal loan amount that John qualifies for. But a national tax agency may use the 60K figure to send John a letter asking him why he underpaid his taxes.

Neither the bank manager nor the tax officer is making any attempt to find out the truth (thus far) - however, both of them are making different decisions based on the same facts.

Handling inconsistencies has been extensively studied in databases as well as in information integration systems. In a database, inconsistency corresponds to the violation of local integrity constraints. In an information integration system, inconsistencies can occur even if each local database is consistent. Given different independent data sources, each with its schema, data integration is the problem of combining these sources and providing a unified view on these data sources. This unified view is called the global (or mediated) schema (on which queries are asked). A database resulting from such an integration may be inconsistent. It is because different databases that are consistent by themselves can contain conflicting tuples that, when integrated, would not satisfy the integrity constraints on the global schema.

Knowledge bases are machine-readable repository of information from which automated reasoning programs can deduce new knowledge. An ontology-based structure is often used for modeling a knowledge base. The problem of inconsistency may arise during the reasoning process if the ontology-based structure is not well organized.

For the Semantic Web, several typical scenarios which may cause inconsistencies are described in (40). They identified four possible causes of inconsistency.

1. Inconsistency by mis-presentation of defaults. A notorious example is the bird ontology in which penguins are specified as the birds which cannot fly, however, it contradicts the default statement birds are flying animals. Another typical example is the Mad-Cows ontology in which MadCow is specified as a cow which eats brains of sheep, whereas a cow is considered as a vegetarian by default.

2. Inconsistency caused by polysemy. Polysemy refers to the concept of words with multiple meanings. An example of an inconsistent ontology which is caused by polysemy is the MarriedWoman concept. This concept is used to refer both to a woman who has a husband and to a woman who had a husband but may no longer have one.

3. Inconsistency resulting from migration from another formalism. When an ontology specification is migrated from other data sources, inconsistencies may occur. As has been found in (60), the DICE terminology (a DL terminology in the Intensive Care domain) suffered from a high number of unsatisfiable concepts due to its migration from a frame-based system. In order to make the semantics as explicit as possible, a very restrictive translation has been chosen to highlight as many ambiguities as possible. (60) shows the inconsistent ontology specification in the Brain example, in which a brain is considered to be both a body part and a central nervous system, whereas body parts and nervous systems are considered to be disjoint.

4. Inconsistency generated by multiple sources. When a large ontology specification is generated from multiple sources, in particular when these sources are created by several authors, inconsistencies easily occur.

Reasoning when there is inconsistency is thus an important issue. Two main kinds of approaches have been proposed for dealing with inconsistency.

- The *first kind of approach* is to *revise the inconsistent data source in order to restore its consistency.*

- The *second kind of approach* is to *accept inconsistencies* and *cope with them by using non classical reasoning mechanism* to produce plausible conclusions.

In the following, we review some representative works of the different ways for handling inconsistencies. It is to note that a majority of these works is in centralized context. They have considered the problem of computing consistent answers of a query in an inconsistent single database or knowledge base. More recently, the advent of applications accessing to distributed information sources has raised the problem of consistent integration of data from distributed multiple sources. We consider that the works having treated this problem are also in a centralized context, because they only consider the problem of reasoning in the integrated database, viewed as a single and centralized database.

While considering these works, we use the term *data source* when talking about either a *database* or a *knowledge base* in general. When it is necessary to distinguish the kind of data source, we use the corresponding term. We present the two main approaches one after the other.

3.1 Consistency restoration

The main idea of this approach is to change the data source by giving up or modifying the piece of data responsible for the inconsistency. By this, we obtain one or several *repairs* of the inconsistent data source.

3.1.1 Restoring consistency of databases

Restoring the consistency for a data source has been studied for a long time. (17; 45) have considered the problem of finding *repairs* for an inconsistent database. Given a fixed database schema and a fixed set of *integrity constraints IC*, the notion of consistency is defined as follows:

Definition 7 *Consistency in a database*
A database instance r is consistent if r satisfies IC in the standard model-theoretic sense, that is, $r \models IC$*; r is inconsistent otherwise.*

Given a database instance r which is inconsistent with the set IC, an answer a of a query asked to the database is considered consistent if it can be obtained from every *repair* of r. A *repair* of an instance r is an instance r' obtained from a modification on r in such a way that $r' \models IC$ and that the cardinality of symmetric difference between r and r' is as little as possible. A modification on r may be an addition of some new formulas, or a suppression of some formulas. The following example shows this idea.

Example 1 *Finding repairs for an inconsistent instance*
Given two unary relations P and Q and domain D = { a,b,c }. Assume that for an instance r the set of formulas that r satisfies is $\Sigma(r) =$ *{ P(a),P(b),Q(a),Q(c) }. Assume that IC =* $\{ \forall x(P(x) \supset Q(x)) \}$*. Clearly, r does not satisfy IC because* $r \models P(b) \wedge \neg Q(b)$.
In this case, there are two possible repairs for r. First, by falsifying P(b) we obtain an instance r′ with $\Sigma(r) =$ *{ P(a),Q(a),Q(c) }. As a second alternative, we can make Q(b) true, obtaining an instance r″ with* $\Sigma(r) =$ *{ P(a), P(b), Q(a), Q(b), Q(c) }.*

We consider now the problem of integrating data from different sources and the integration must preserve the coherence of the integrated database. In fact, as discussed in (29) , the information stored in the different databases may be contradictory, especially in the presence of integrity constraints (key constraints, foreign key constraints etc). The point is to provide consistent answers to queries.

A simple example of this problem is the following. Consider three databases which store information about students and the departments in which they are registered. Furthermore, each database is associated with the key constraint expressing that a student is registered in only one department. The first database stores the facts: $Student(John, maths)$, $Student(Sandra, maths)$, the second one stores $Student(John, physics)$, $Student(Paul, maths)$ and the last one stores $Student(John, maths)$, $Student(Sandra, physics)$. What is the answer to the query : where is John registered ?

We can determine the data that violates integrity constraints and then delete it. Alternatively, by adding new data to the integrated database, it is also possible to

restore its coherence. (18) demonstrates this idea. Given n consistent databases $DB_i = (D_i, IC_i), i = 1, ...n$ where D_i is the instance of DB_i and IC_i is its set of integrity constraints, the goal is to do in such a way that the combined data will contain everything that can be deduced from one source, without violating any integrity constraint of another source.

Let $DB = (D, IC)$ be the integrated database, where $D = \cup_{i=1}^{n} D_i$ and $IC = \cup_{i=1}^{n} IC_i$. In the case where the integrated database is inconsistent, the goal is to modify it to restore its consistency. By a model-theoretic analysis, the *repairs* are characterized in terms of a certain set of models of the inconsistent database (intuitively, those that minimize the amount of inconsistent information).

Definition 8 *Repair*
A repair of $DB = (D, IC)$ *is a pair* $(Insert, Retract)$ *such that* $Insert \cap D = \emptyset$, $Retract \subseteq D$ *and* $(D \cup Insert \backslash Retract, IC)$ *is a consistent database.*

$Insert$ is a set of elements that should be inserted into D and $Retract$ is a set of elements that should be removed from D. As there are several possible *repairs* for the integrated database, the preferred repairs are defined following set inclusion or cardinality criteria. Intuitively, these are those that minimize the difference with the original database.

In order to compute these repairs, a characterization of them based on a three-valued logic, called $THREE$, is proposed. In the semantics of $THREE$, a formula is interpreted as t (true), or f (false), or $\top$ (incoherent). The repairs are then computed as follows: let N a model of $D \cup IC$, $Insert$ is computed to be $N^{\top} \backslash D$, and $Retract$ is computed to be to $N^{\top} \cap D$ where $N^{\top}$ is the set of formulas the interpretation of which in the model N is $\top$. This characterization of $Insert$ and $Retract$ satisfies the definition 8, that makes $(Insert, Retract)$ a repair of the integrated database. The coherence of this database is preserved, but its structure is not.

3.1.2 Restoring consistency of knowledge bases

Similarly, several works have dealt with restoring consistency for a single inconsistent knowledge base. We focus on knowledge bases described in a Description

Logic language (38). Such a knowledge base consists of a TBox T and an ABox A. T contains intensional knowledge such as concept definitions in a terminological axiom form $C \sqsubseteq D$, where C and D are concepts. Hence, T is also called a terminology. The *ABox* A contains extensional knowledge and is used to describe individuals.

The definition of inconsistency in a knowledge base is different from the one in a database and is often referred to as incoherency. A Tbox which is logically consistent (i.e. satisfiable) may be considered as incoherent if it contains a concept the definition of which is such that it is interpreted as an empty set in every model of the Tbox. The relation between incoherence and inconsistency is that the incoherence of a Tbox causes the inconsistency of any Abox defined as an instance of that Tbox : if a Tbox T is incoherent, it contains a concept C such that the knowledge base $T \cup C(a)$ is inconsistent for any instance a.

The semantics of DL knowledge bases is the standard first-order logic semantics, defined in terms of interpretations. An interpretation I is a pair $(\Delta^I, \cdot^I)$ where Δ^I is a non-empty set, called the domain of interpretation, and $\cdot^I$ is an interpretation function which assigns a subset of Δ^I to every class name in the Tbox and an element of Δ^I to every instance identifier of the Abox. An interpretation I is a model of a Tbox if and only if every inclusion axiom of the Tbox are satisfied in I. A concept name C in a terminology T is unsatisfiable if and only if for each model I of T, $C^I = \emptyset$. A *TBox* T is incoherent if and only if there exists an unsatisfiable concept name in T. Repairing an inconsistent DL knowledge base consisting thus in repairing the TBox T as in (55; 59) or to also consider the ABox as in (56; 57).

For example, in (59), the approach used to repair the TBox T consists in two steps, trying to locate the cause of the incoherence.

- The first step is called *axiom pinpointing*. It is to identify the axioms which are *relevant* for the debugging of the incoherent Tbox in question. An axiom is *relevant* if the Tbox becomes coherent once the axiom is removed, or if, at least, a particular, previously unsatisfiable concept turns satisfiable. These axioms can be localized by doing the following:

Firstly, given the incoherent Tbox T, for every unsatisfiable concept A in T, we find *minimal unsatisfiability-preserving sub-TBoxes of* T *and* A, in short: $MUPS(T, A)$. $MUPS(T, A)$ are subsets of T in which A is unsatisfiable. In general there are several of these sub-TBoxes and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

Secondly, from all the $MUPS(T, A_i), i = 1..n$ (each A_i is an unsatisfiable concept in the Tbox), we calculate *minimal incoherence-preserving sub-TBoxes of* T, in short: $MIPS$. Intuitively, $MIPS$ are the smallest subsets of an original TBox preserving unsatisfiability of at least one atomic concept.

For example, consider the Tbox T that contains the following axioms:

1. ax_1: $A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3$
2. ax_2: $A_2 \sqsubseteq A \sqcap A_4$
3. ax_3: $A_3 \sqsubseteq A_4 \sqcap A_5$
4. ax_4: $A_4 \sqsubseteq \forall s.B \sqcap C$
5. ax_5: $A_4 \sqsubseteq \exists s.\neg B$

In this TBox, A, B and C are primitive concepts, whereas A_1, A_2, A_3 and A_4 are defined concepts. Moreover, A_1 and A_3 are unsatisfiable. The set $MUPS(T, A_1)$ is $\{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$ and the set $MUPS(T, A_3)$ is $\{ax_3, ax_4, ax_5\}\}$.

From $MUPS(T, A_1)$ and $MUPS(T, A_3)$, $MIPS(T)$ is calculated. It is equal to $\{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}\}$. We can verify that each sub-Tbox in $MIPS(T)$ is the smallest one incoherence-preserving: it contains only faulty axioms.

- The second step is called *concept pinpointing*. It is to point out the error in the definition of a concept in the faulty axioms. Given a faulty axiom $A \sqsubseteq B$, the idea is to generalize B using the definition of B in an other faulty axioms.

For example, for the axiom $ax_1 = A_1 \sqsubseteq \neg A \sqcap A_2 \sqcap A_3$, by generalizing A_2 using its definition in the axiom ax_2, we obtain that $A_1 \sqsubseteq \neg A \sqcap A \sqcap A_3$. This new axiom shows the precise error for the unsatisfiability of A_1: it is in the intersection of A and $\neg A$.

Similarly, by generalizing A_4 and A_5 using their definition in ax_4 and ax_5, we obtain that $A_3 \sqsubseteq \forall s.B \sqcap \exists s.\neg B$. This shows why A_3 is unsatisfiable.

After finding the precise errors concerning the definition of unsatisfiable concepts, the reparation task is thus done by re-defining these concepts with new axioms. The structure of the original knowledge base is not preserved.

3.2 Inconsistency tolerance

In this section, we consider a set of representative works adopting the *second kind of approach* when treating inconsistencies. This approach is to *accept inconsistencies* and *cope with them*. The point distinguishing this approach from the previous one is that one does not change the structure of the data source. The inconsistent information still exists in the source but wrong conclusions are avoided.

Two main categories of logical frameworks have been designed for reasoning in presence of inconsistencies.

1. The first one is the category of *paraconsistent logics*. These logics, that we present in the subsection 3.2.1, are those that do not verify the *principle of explosion* of classical logics.

2. The second one is the category of *preservationist logics* or *abstract logics* that we will present in the subsection 3.2.2.

It is also to notice that these logical frameworks can be applied to different context (centralized or decentralized databases, knowledge bases). Some other methods for coping with inconsistencies, based on techniques using *majority in point of views*, or *the reliability of data sources*, will also be presented briefly.

3.2.1 Logical frameworks based on paraconsistent logics

In classical logic, contradictions entail everything. This curious feature, known as the principle of explosion or *ex contradictione sequitur quodlibet* (from a contradiction, anything follows), can be expressed formally as

$$A, \neg A \models B$$

where $\models$ represents logical consequence. Thus if a theory contains a single inconsistency, it is trivial that it has every sentence as a theorem. The characteristic of a paraconsistent logic (24) is that it rejects the principle of explosion. As a result, paraconsistent logics, unlike classical logics, can be used to formalize inconsistent but non-trivial theories. It should be emphasized that paraconsistent logics are in general weaker than classical logic: it does not validate everything that classical logic does. In that sense, paraconsistent logic is more "conservative" or "cautious" than classical logic.

3.2.1.1 Annotated Predicate Calculus

Annotated Predicate Calculus (APC) (43) is a paraconsistent logic for treating any set of clauses, either consistent or not, in a uniform way. In this logic, consequences of a contradiction are not as damaging as in the standard predicate calculus (PC), and meaningful information can still be extracted from an inconsistent set of formulas while wrong conclusions are not deduced. We will only present the main difference between APC and PC to show the mechanism of APC that avoids to draw wrong conclusions when there is an inconsistent relation in the set of formulas.

- At the syntactic level, the language of APC is similar to that of PC. The only syntactic difference is that atomic formulas in APC are constructed from those of predicate calculus (e.g. l) by appending to them *annotations* (e.g. $l : r$), where r is an annotation drawn from a so-called *upper semi-lattice of degrees of belief*, in short: BSL. Intuitively, r represents the reasoner's belief in the truth of the statement l. It can take one of the following four values: t (true), f (false), $\top$ (contradiction) and $\bot$ (unknown), with the following

order: $\forall s \in$ BSL, $\bot \leq s \leq \top$. Some examples of literals of APC are $q : \top$, or $p(X) : s$ where $\top$ and $s \in$ BSL. Some examples of formulas of APC are $(\forall Y)r(Y) : t$, $(\exists X)\neg q(X, Z) :\bot$ where t and $\bot \in$ BSL.

- At the semantic level, the notion of interpretation in APC differs from the one of PC. In APC, an interpretation I is a tuple $< D, I_F, I_P >$. As in PC, D is the domain of I, I_F associates to each k-ary function symbol f a mapping $I_F(f)$:$D^k \rightarrow D$. The difference between APC and PC is that in APC, I_P associates to each m-ary predicate symbol p a function $I_P(p)$:$D^m \rightarrow BSL$, instead of $D^k \rightarrow \{true, false\}$ as in PC.

  A valuation ν is a function that assigns values from D to variables. For an atomic formula $p(t_1, ..., t_k) : s$, we write $I \models_\nu p(t_1, ..., t_k) : s$ if and only if $I_P(p)(\nu(t_1), ..., \nu(t_k)) = r \in$ BSL and $s \leq r$.

  Complex formulas are interpreted as usual:

  - $I \models_\nu \phi \vee \psi$ if and only if $I \models_\nu \phi$ or $I \models_\nu \psi$
  - $I \models_\nu \phi \wedge \psi$ if and only if $I \models_\nu \phi$ and $I \models_\nu \psi$
  - $I \models_\nu \neg\phi$ if and only if not $I \models_\nu \phi$
  - $I \models_\nu (\forall X)\phi$ if and only if $I \models_u \phi$ for every u that may differ from ν only in its X-value
  - $I \models_\nu (\exists X)\phi$ if and only if $I \models_u \phi$ for some u that may differ from ν only in its X-value

  A formula ϕ is satisfied by an interpretation I if for every valuation ν, $I \models_\nu \phi$. In this case, we write $I \models \phi$. An interpretation I is a *model* of a set S of formulas if and only if every formula ϕ in S is satisfied by I. A set of formulas S *logically entails* a formula ϕ if and only if every model of S is also a model of ϕ.

  In APC, a set of formulas is always satisfiable because the inconsistent relations are characterized by $\top$. The interpretation in which all atomic formulas of a set S are interpreted to $\top$ will satisfy all these formulas, because $\top$ is the highest element (in terms of order) in a BSL.

A main difference between APC and PC is that APC presents a new notion of *implication* that avoid drawing anything from an inconsistent relation .

- In PC, the set of formulas $\{\neg q, q, q \rightarrow p\}$ is inconsistent. It has every formula as a consequence, such as p, even if q is an inconsistent relation.
- In APC, consider the set $\{q : f, q : t, q : t \rightsquigarrow p : t\}$ where $q : t \rightsquigarrow p : t$ is equivalent to $q : f \vee p : t$. This set is also inconsistent because of the relation q. However, we will show that $p : t$ is not deduced from this set.

  Consider the interpretation I in which $I \models q : \top$ and $I \models p :\bot$. Because $f < \top$ and $t < \top$ in terms of order of BSL, by definition we have $I \models q : f$ and $I \models q : t$. In addition, as we have $I \models q : f$, we also have $I \models q : f \vee p : t$ by definition. Therefore I is a model of the considered set. However, this is not a model of $p : t$, because in I, p is interpreted to $\bot$ and $\bot < t$. Thus, $p : t$ is not deduced from the above inconsistent set of formulas.

3.2.1.2 LFI1

$LFI1$ (31) is a logic in the family of *Logics of Formal Inconsistency* (LFIs). It uses a new symbol $\bullet$ which stands for *partially true*, assimilated to the case of inconsistency. A PC's well-formed formula F is also a $LFI1$'s well-formed formula, as well as $\bullet F$.

At the semantic level, $LFI1$ is a *three-valued* logic. An interpretation assigns to a formula a truth value drawn from $\{0, \frac{1}{2}, 1\}$, where 0 stands for *false*, 1 stands for *true* and $\frac{1}{2}$ stands for *inconsistent*. As well as the APC paraconsistent logic just presented above, $LFI1$ also redefines the notion of implication. In $LFI1$, it is defined as $A \rightarrow B \equiv B \vee \neg A \vee \bullet A$. The notion of *models* and *logical consequences* are as usual as in PC but extended to accept the value $\frac{1}{2}$. It means, an interpretation I satisfies a formula F, denoted as $I \models F$, if $I(F) = 1$ or $\frac{1}{2}$.

Because of the addition of the new symbol $\bullet$ and a new definition of implication, new connective matrices are defined. Figure 3.1 below shows the connective matrices for the primitive logical symbols $\vee$, $\neg$, $\bullet$.

$\vee$	1	$\frac{1}{2}$	0
1	1	1	1
$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
0	1	$\frac{1}{2}$	0

	$\neg$	$\bullet$
1	0	0
$\frac{1}{2}$	$\frac{1}{2}$	1
0	1	0

Figure 3.1: Connective matrices used for $LFI1$

The following example shows that $LFI1$ does not verify the *principle of explosion*, i.e., $A, \neg A \not\models B$ for all B. Let R be a binary predicate symbol, Let I be the interpretation such that $I(R(a,b)) = 1$, $I(R(c,b)) = \frac{1}{2}$ and $I(R(p,q)) = 0$ for all (p,q) such that $p \neq c$ and $p \neq a$, or $q \neq b$. As $I(R(c,b)) = \frac{1}{2}$, using the connective matrices, we also have $I(\neg R(c,b)) = \frac{1}{2}$. It means, $I \models R(c,b)$ and $I \models \neg R(c,b)$. However, $I \not\models R(b,a)$ as $I(R(b,a)) = 0$.

3.2.1.3 Distributed Description Logics with holes

(61) is one of the work aiming at proposing a logical framework for reasoning with distributed ontologies. This work extends the *Distributed Description Logic* (DDL) presented in (20) by introducing the notion of *distributed interpretation with holes*.

The DDL in (20) can be briefly described as follows.

- At the syntactic level, consider a set of local ontologies, each with a TBox T_i, where $i \in I$, a set of integers. A local concept C in T_i is denoted as $i : C$. Semantic mappings between different ontologies are expressed via *bridge rules*. A bridge rule from an ontology i to an ontology j is an expression restricted to being one of the following two forms:

  1. $i : x \xrightarrow{\sqsubseteq} j : y$ - an *into-bridge rule*
  2. $i : x \xrightarrow{\sqsupseteq} j : y$ - an *onto-bridge rule*

  where x and y are any concepts of T_i and T_j. Intuitively, a bridge rule from an ontology i to an ontology j expresses the relation between concepts of T_i

and T_j, viewed from the subjective point of view of the j-th ontology. For example, the mapping $i : C \xrightarrow{\sqsubseteq} j : D$ says that in the viewpoint of j, the individuals in the concept C of T_i correspond to a subset of the individuals in its local concept D.

A *distributed TBox* $\Re$ is defined as $\Re =< \{T_i\}, \mathfrak{B} >$: it consists of a collection of local TBoxes and all the semantic mappings $\mathfrak{B}$ between them.

- At the semantic level, each local ontology is interpreted locally by a local interpretation $\mathfrak{I}_i = \{\Delta^{\mathfrak{I}_i}, \cdot^{\mathfrak{I}}\}$. A domain relation r_{ij} from $\Delta^{\mathfrak{I}_i}$ to $\Delta^{\mathfrak{I}_j}$ is a subset of $\Delta^{\mathfrak{I}_i}$ x $\Delta^{\mathfrak{I}_j}$. A distributed interpretation $\Im$ of a distributed TBox $\Re$ is defined as $\Im = < \{\mathfrak{I}_i\}, \{r_{ij}\} >$. $\Im$ is said to satisfy (written $\Im \models_d$) the elements of $\Re$ if:

  1. $\mathfrak{I}_i \models_d A \sqsubseteq B$ for all $A \sqsubseteq B$ in T_i
  2. $\Im \models_d i : x \sqsubseteq\!\!\rightarrow j : y$, if $r_{ij}(x^{I_i}) \subseteq y^{\mathfrak{I}_j}$
  3. $\Im \models_d i : x \sqsupseteq\!\!\rightarrow j : y$, if $r_{ij}(x^{I_i}) \supseteq y^{\mathfrak{I}_j}$
  4. $\Im \models_d \Re$, if for every i,j, $\Im \models_d T_i$ and $\Im \models_d \mathfrak{B}_{ij}$ where $\mathfrak{B}_{ij}$ is the set of semantic mappings from the ontology i to the ontology j.

  A distributed TBox $\Re$ is satisfiable if there exists an distributed interpretation $\Im$ such that $\Im \models_d \Re$. A concept $i : C$ is satisfiable with respect to $\Re$ if there is a distributed interpretation $\Im$ such that $\Im \models_d \Re$ and $C^{\mathfrak{I}_i} \neq \emptyset$

The DDL in (20) have some problems when there are some local inconsistent TBoxes in the distributed TBox: they make the whole distributed TBox to not have a model. As a consequence, every statement is true even in the other consistent TBoxes. In this case, we say that the local inconsistency spreads and contaminates the other local ontologies. In order to fix this problem, (61) introduced the notion of *distributed interpretation with holes*. The definition of a *hole* is as follows:

Definition 9 *A hole for a TBox T is an interpretation $\mathfrak{I}^\epsilon = < \emptyset, \cdot^\epsilon >$, where the domain is empty.*

Let us extending the d-entailment $\models_d$, obtaining $\models_\epsilon$ by also allowing holes as interpretations for local TBoxes. $\models_\epsilon$ has two important properties.
The first property of $\models_\epsilon$ is that $\mathcal{I}^\epsilon \models C \sqsubseteq D$ for every C and D because both are interpreted as the empty set. Therefore, even if some local TBox T_i is inconsistent, we still have a distributed interpretation for the whole distributed TBox, the one that uses $\mathcal{I}^\epsilon$ to satisfy T_i.
For the second property of $\models_\epsilon$, let us consider a distributed TBox $\mathfrak{R}$ in which n local TBoxes $T_{i1}, ..., T_{in}$ are inconsistent. For any local inconsistent TBox T_i, let $\mathfrak{R}(\epsilon_i)$ denote the distributed TBox obtained by removing T_i, $\mathfrak{B}_{ij}$, $\mathfrak{B}_{ji}$ from $\mathfrak{R}$ and by extending each T_j with the set of axioms $\{G \sqsubseteq \bot\}$ whenever there is a bridge rule $i : A \xrightarrow{\sqsupseteq} j : G \in \mathfrak{B}_{ij}$. Let J be the set $\{i1, ..., in\}$ and let $\mathfrak{R}(\epsilon_J)$ be $\mathfrak{R}(\epsilon_{i1})...(\epsilon_{in})$. The second property of $\models_\epsilon$ is that it permits to avoid the problem of spreading and contamination of local inconsistencies into other local ontologies, due to the following theorem (theorem 1).

Theorem 1 *$\mathfrak{R} \models_\epsilon i : X \sqsubseteq Y$ if and only if for every J not containing i, $\mathfrak{R}(\epsilon_J) \models_d i : X \sqsubseteq Y$*

3.2.2 Logical frameworks using other logics

In this subsection, we present some other logical frameworks for dealing with inconsistent knowledge bases. These framework use logics that are not paraconsistent: they verify the *principle of explosion*. However, by redefining the notion of entailment (the case of *Abstract Logic*) or the property to be preserved (the case of *preservationist logics*), wrong conclusions can also be avoided.

3.2.2.1 Preservationist logic

In (15) , the characteristics of *preservationist logics* are presented. The way to use these logics is simple: we identify the property of the data source (satisfiability for example) that we want to preserve, then we adapt the notion of logical consequence in such a way that the chosen property is always preserved. Consequently, given a property to be preserved, if the data source permits to imply a new formula while preserving the chosen property then the formula is a *plausible* logical consequence of the data source.

A special point of *preservationist logics* is that one can choose not only the satisfiability as the property to be preserved, but also *any* other properties. An example of work using *preservationist logics* is (16). (16) chose two properties to be preserved: *the level of coherence* and *the dilution of level*. In order to define the *level of coherence* of a data source, one consider the different coherent partitions of this source (the partitions such that each of them is classically consistent). A partition of a data source Σ is defined as a set of subsets of Σ such that these subsets do not intersect and that together they cover Σ. The size of a partition is the number of subsets of Σ that it contains. It is also said that a partition is a *n-partition* if it contains exactly n subsets of Σ. A partition is considered coherent if all of its elements (subsets of Σ) are classically coherent. The level of coherence of the initial data source Σ is defined as the size of its smallest coherent partition.

The companion notion *dilution of level* is defined based on the notion of *level of coherence*. Given a data source Σ with the *coherence level* n, the *dilution of level-n* of Σ is the size of the smallest subset Σ' of Σ (in terms of number of formulas) such that the level of coherence of Σ' is also equal to n. Intuitively, the dilution measures the size of the subset of Σ that participate into the incoherence in Σ. The more the dilution is small, the more the incoherence is concentrated on a small number of formulas of Σ. By preserving in the same time the *level of coherence* and the *dilution* relative to this level during a reasoning, one can warrant that the inferred formulas will not create new incoherences in the data source, neither to increase the concentration of the incoherence in the source.

Preserving the *level of coherence* can be done using the following result. Given a source Σ and suppose that the level of coherence of Σ is n, it can be shown that a new formula α added to Σ will preserve its level of coherence if α follows from every coherent n-partition of Σ. It means, for each coherent n-partition, there exists a subset of Σ that implies classically α.

For preserving the *dilution of level*, the point is to keep the dilution from decreasing. It is done using the following result. Given a source Σ and suppose that the level of coherence of Σ is n and the dilution at level n is equal to m, it can

be shown that m is preserved when adding a new formula α to the source if for every subset Σ' of Σ such that the size of Σ' is $m-1$ then the addition of α into Σ' preserves the level of coherence $n-1$ for Σ'.

The *preservationist logics* are flexible because we are free to choose the properties to be preserved. However, it is not easy to preserve a property. In the example above, we have to check for all the n-partitions of Σ in order to preserve its *level of coherence*. This task could be expensive.

3.2.2.2 Abstract logic

In 1982, Dana Scott (33) defined an abstract logic which consists of a set L (whose members are called well-formed formulas) and a consequence operator CN. CN is any function from 2^L (the powerset of L) to 2^L. Formally, $CN(X)$ is the set of formulas that are logical consequences of X according to the logic in question, that satisfies the following axioms:

- Expansion $X \subseteq CN(X)$
- Idempotence $CN(CN(X)) = CN(X)$
- Monotonicity $X \subseteq Y \Rightarrow CN(X) \subseteq CN(Y)$
- Coherence $CN(\emptyset) \neq L$

Very recently, (64) has extended the Abstract Logic presented in (33) in order to be able to deal with inconsistencies. Its approach is simple and mainly consists in redefining the notion of entailment. In (64) , the notion of *Consistency* is defined as follows:

Definition 10 *Consistency*
Let $X \subset L$. X is consistent in logic (L, CN) if and only if $CN(X) \neq L$.

This definition of consistency says that X is consistent if and only if its set of consequences is not the set of all well formed formulas.

A second notion that is presented is the notion of *Options*. An option O is any set of elements of L that is consistent and closed, i.e. $O = CN(O)$. The general

framework for reasoning about inconsistency is defined as a triple $< Opt(L), \succeq, \rightsquigarrow >$ where $Opt(L)$ is the set of all options that can be built from (L, CN), $\succeq$ is a partial or total preorder between options and $\rightsquigarrow$ is an entailment mechanism.

Let $O^{\succeq}$ be the set of all preferred options of L, based on any order that a user can define for $\succeq$, such as set inclusion. The entailment mechanism can be defined as one of the two following criteria:

- Universal criterion $L \rightsquigarrow \psi$ iff $\psi \in O_i$, $\forall O_i \in O^{\succeq}$. ψ is called a universal consequence of L
- Argumentative criterion $L \rightsquigarrow \psi$ iff $\exists O_i \in O^{\succeq}$ such that $\psi \in O_i$, and $\nexists O_i \in O^{\succeq}$ such that $\neg\psi \in O_i$. ψ is called an argumentative consequence of L

It can be shown that the set of formula, which are universal or argumentative consequence of L, is an option of L, and thus is consistent.

It is to notice that this logical framework verifies the *principle of explosion*, thus it is not classified as a *paraconsistent logic*. However, the newly defined notion of entailment warrants to produce only consistent consequences from set of *options* that are by definition consistent.

Such a general framework can be applied to handle inconsistencies in an inconsistent knowledge base. Given K an inconsistent knowledge base, an option O can handle K if there is a subset K' of K such that $O = CN(K')$. Consequently, K' is consistent as well as the set $CN(K')$ of its consequences. Moreover , an *optimal* option for K can be chosen from the set $O^{\succeq}$, taking into account the order of options defined by user, to choose the most preferred K' from K.

It is worth noticing that (40) proposed a very similar approach to the one of (64). (40) considers OWL and its underlying description logics. Similarly to (64) , it also redefines the notion of entailment for an *inconsistency reasoner*. An *inconsistency reasoner* is denoted by the symbol $|\approx$, instead of the classical symbol $\models$. In this work, given an inconsistent set of formulas Σ, *soundness* and *meaningfulness* of an inconsistency reasoner are defined as follows :

Definition 11 *Soundness*
An inconsistency reasoner $\not\approx$ *is sound if the formulas that follow from an inconsistent theory* Σ *follow from a consistent sub-theory of* Σ *using classical reasoning, namely, the following condition holds:*

$\Sigma \not\approx \phi \Longrightarrow (\exists \Sigma' \subseteq \Sigma)(\Sigma' \not\models \bot$ *and* $\Sigma' \models \phi)$

Definition 12 *Meaningfulness*
An answer given by an inconsistency reasoner is meaningful iff it is consistent and sound. Namely, it requires not only the soundness condition, but also the following condition:

$\Sigma \not\approx \phi \Longrightarrow \Sigma \not\approx \neg\phi$

An inconsistency reasoner is said to be meaningful iff all of the answers are meaningful.

It is to note that the *soundness* and *meaningfulness* properties of an inconsistency reasoner in (40) make it equivalent to the notion of *argumentative criterion* of entailment in (64) .

One of the important problems that these two works have to solve is to find the *optimal options* for (64) , or to find the subset Σ' that is consistent for (40) . Given Σ the inconsistent set of formulas, both works propose a method that starts from a subset of Σ and incrementally adds new formulas to this subset, preserving its consistency. There is a difference between the two works, concerning this procedure. In (64) , the subset is extended with the consequences of the formulas belonging to it, and by adding new formulas from Σ. In (40) , the subset is extended in each step by adding new formulas from Σ then by checking if this subset can answer if a formula ϕ can be consistently entailed from it.

We illustrate the *Linear Extension Strategy* in (40) in the figure 3.2. The scenario is carried out as follows. Given a query $\Sigma \not\approx \phi$, the initial consistent subset Σ' is set. Then the selection function is called to return a consistent subset Σ'' , which extends Σ', i.e., $\Sigma' \subseteq \Sigma'' \subseteq \Sigma$ for the linear extension strategy. If the selection function cannot find a consistent superset of Σ', the inconsistency reasoner returns the answer undetermined (i.e., unknown) to the query. If the set Σ'' exists, a classical reasoner is used to check if $\Sigma'' \models \phi$ holds. If the answer

is *yes* then $\Sigma \not\approx \phi$ holds. If the answer is *no* then the inconsistency reasoner further checks the negation of the query $\Sigma'' \models \neg\phi$. If the answer is *yes* then $\Sigma \not\approx \neg\phi$ holds, otherwise the current result is undetermined and the whole process is repeated by calling the selection function for the next consistent subset of Σ which extends Σ''.

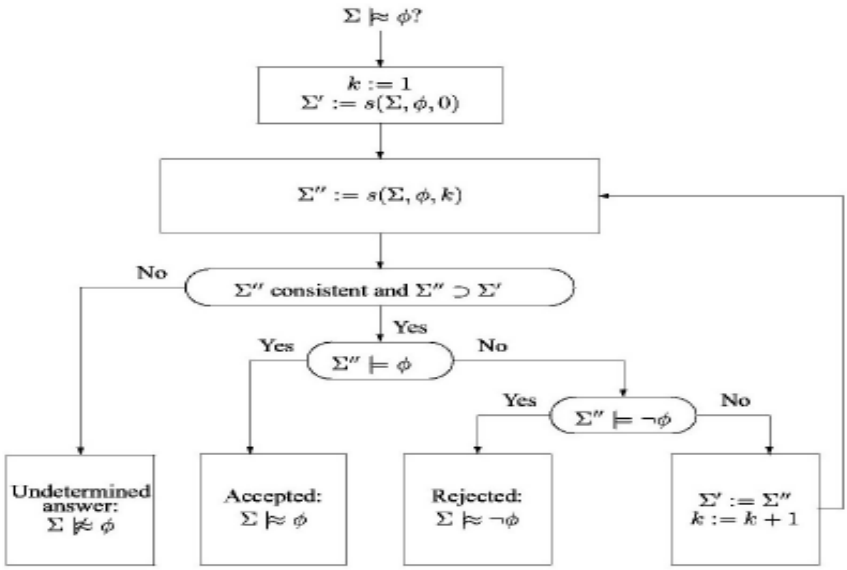

Figure 3.2: Linear Extension Strategy used in (40)

3.2.2.3 The $K45_n^A$ epistemic logic

In (25; 26), Calvanese and al. have considered the problem of inconsistency in peer-to-peer data integration systems. In such systems, each peer models an autonomous system that exports data in terms of its own schema, and data inter-operation is achieved by means of mappings among the peer schemas.

Two types of inconsistency are distinguished in such systems: local inconsistency and peer-to-peer inconsistency. Local inconsistency of a peer means that the data stored at the peer contradicts the peer schema. On the other hand, peer-to-peer inconsistency raises inside a peer P_i when the data coming from another peer P_j (through a mapping) contradicts the local data of P_i (or contradicts the data coming to P_i from another peer P_k).

Such a peer-to-peer data integration system can be formalized by means of the language of $K45_n^A$. This logic is extended from the logic $K45_n$ that is itself extended from $K45$ (44). Like $K45$, $K45_n$ is based on first-order logic, but has modal operators expressing the knowledge of a peer about a formula.

While $K45_n$ is not able to deal with inconsistency, its extension $K45_n^A$ is. $K45_n^A$ is a non-monotonic multi-modal epistemic logic. In comparison to $K45_n$, $K45_n^A$ has modal operators representing so-called *assumptions* of a peer about the truth of formulas. It is only when the *assumption* that a peer data does not contradict the schema of this peer is justified (no local inconsistency) that the data of this peer can be sent to the other peers as answers for their queries.

For peer-to-peer inconsistency inside a peer, under the semantics of $K45_n^A$, two contradictory data (among which at least one coming from an other peer) can only co-exist if the contradictory fragment of information between them is not taken into account. For example, consider a peer schema with the axiom "at a given moment, a person lives in only one place and holds only one citizenship." Suppose that this peer receives from two other peers two contradictory data : "John is Canadian and lives in Roma" and "John is Italian and lives in Roma". The contradictory fragment of data is on the citizenship of John. Under the semantics of $K45_n^A$, this fragment is discarded so that a query about the citizenship of John will return the empty set. In an other hand, the place where John lives is the same in these two data: Roma will be returned for a query about the place where John lives.

As we have seen, many logical frameworks have been proposed to deal with inconsistencies. They differ mainly on the kind of logics on which they are based. We summarize and compare the logical frameworks presented above in the table 3.1.

3.2.3 Other techniques for dealing with inconsistencies

We present in this sub-section some other techniques for drawing plausible conclusions in presence of inconsistencies. These techniques are based more or less on a logical framework (as it is always the case when it comes to reasoning), but

Approach	has been applied in	has important properties
APC paraconsistent logic	Single DB	PC-based Interpretation redefinition Implication redefinition Four-valued
LFI1 paraconsistent logic	Single DB	PC-based Implication redefinition Entailment redefinition Three-valued
Preservationist logic	Single KB	PC-based Coherence level preserving Dilution preserving
Abstract logic	Single KB	Entailment redefinition
DDL with holes	Distributed ontologies	Description Logic extension Distributed interpretation with holes Distributed entailment
$K45_n^A$ logic	P2P data integration systems	Epistemic Logic

Table 3.1: Summary on the logical frameworks presented in the sections 3.2.1 and 3.2.2

the influence of the logical base is not as clear and important as in the works that we have just presented above.

3.2.3.1 Source reliability-based technique

In (28), the problem of inconsistency in an integrated data source is considered. The integrated data source is formed by integrating different local sources. These sources are ordered. The order depends on the application: the order can represent the relative reliability of agents, in the case of agents who observe the real world and send their beliefs; it can represent the "age" of the agent in the sense that the more recent an agent is, the more important it is assumed to be; finally, the order can represent any arbitrary choice of the person who collects the information.

A very simple idea is that the more important (according to the order) an agent is, the more we want to keep the information it provides. For example, consider two agents a_i and a_j, and assume that a_i is more reliable than a_j. When integrating a_i and a_j, an information A that can be deduced from a_i will be kept as a new axiom for the integrated source, while $\neg A$ that can be deduced from a_j is not accepted.

3.2.3.2 Majority technique

The problem of inconsistency in an integrated data source is also considered in (29). The technique that is used to handle inconsistency is very simple, it counts the number of times that a formula and its negation appear in the sources. For example, we want to know if the axiom q is true in an integrated database. If q appears in some sources and its negation does not appear in any source then the answer is *yes*. An other possibility to have a positive answer is *yes by majority*, expressing that the number of times that q appears in the sources is greater than that of its negation, which itself is superior to 0. If q and $\neg q$ appear the same number of times, superior to 0, the answer is *Balanced Inconsistency*. Finally, the answer *Undetermined* is returned when neither q nor $\neg q$ appear in any source.

3.2.3.3 Change detection for compiled knowledge technique

(63) considers the problem of inconsistency in modular ontologies. Modular ontologies are distributed ontologies, each ontology is considered as a module. While it is classical that a module consists of its intensional knowledge (its concept definitions and relations), it is to note that a module in (63) consists also of an extensional part. An extensional part of a module contains external concept definition. An external concept definition of a module M is an axiom of the form $C \equiv M' : Q$, where M' is an other module and Q is a query over the vocabulary of M'.

With this structure, while reasoning inside a module in order to imply new axioms, it is not only necessary to reason in the internal part of this module, but also in the external part which requires a reasoning in the corresponding external module. The idea is to do this external reasoning only once, then to import all

the deduced knowledge in the external module into the local one. The imported knowledge is also called compiled knowledge. These knowledge can be used in further reasoning inside the local module.

However, if there is a change (addition or suppression or modification) in the corresponding external source, the compiled knowledge may become inconsistent or out of date. Therefore, the definition of "consistency" in this case is not the same as we have seen in the works presented above. An algorithm has been proposed for each module to detect changes in external modules from which it has compiled knowledge. The changes are classified as "harmful" or "harmless". This algorithms is based on the Distributed Description Logic (without holes) that we have presented above.

3.3 Summary

There are two main approaches for dealing with inconsistencies in the literature.

1. The first approach consists in repairing the data source. This approach does not preserve the structure of the data source. It is thus suitable to be used only in a centralized context, such as a single database or a single knowledge base. Actually, in a typical Semantic Web setting, one would be importing ontologies from other sources, making it impossible to repair them, and the scale of the combined ontologies may be too large to make repair effective. Moreover, when there are multiple data sources, if there is no explicit order between the sources then they should be treated equally.

2. The second approach is suitable for both centralized or decentralized context, because it preserves the structure of the data source. This approach is mostly based on redefining some notions in logics, such as implication or entailment in order to avoid wrong conclusions.

In the next chapter, we will present in detail the inconsistency problem that we have considered in SomeWhere and our solution for it. As we are interested in dealing with inconsistency in a totally decentralized network, the first approach

consisting of restoring the consistency for a single data source can not be applied, as discussed above.

We will therefore adopt the second approach for dealing with inconsistencies. We accept the presence of inconsistencies in the SomeWhere network and propose a way to draw only well-founded conclusions. Given an inconsistent set Σ of formulas and a query q, our approach is in somehow similar to the one of (64) and (40). We define well-founded logical consequences of a set of formulas Σ as those which can be classically deduced from a consistent subset Σ' of Σ. Our approach can be distinguished from those of (64) and (40) on the following two points:

- Firstly, given an inconsistent set Σ of formulas, we accept both A and its negation $\neg A$ as well-founded consequences of Σ, as long as each of them can be classically deduced from a consistent subset Σ' of Σ.
- Secondly, the inconsistent set Σ is not known in our framework. Σ is the union of all the peer theories which are distributed over a SomeWhere network. There is no centralized control so that one can know the whole theory. Therefore, our algorithm to compute well-founded consequences of a query w.r.t to Σ must run in a fully decentralized manner.

Chapter 4

DEALING WITH INCONSISTENCIES IN SOMEWHERE

In the SOMEWHERE peer-to-peer inference system, one of the assumptions that have been made is that the global SOMEWHERE theory is consistent so that query answering in SOMEWHERE produces meaningful answers. However, this assumption may not be relevant considering the decentralized and multi-authored nature of the peer theories. In this case, query answering in SOMEWHERE may produce answers that are meaningless.

Our choice is to compute only *well-founded* consequences of a formula, i.e., those that are consequences of the formula w.r.t. to a consistent subset of the global theory. For this, the problem is first to *detect* inconsistencies, second to *reason* in spite of them in a satisfactory way. In addition, these two tasks have to be done in a *decentralized* manner, because the peer theories are distributed and the global theory of SOMEWHERE is not known to any peer. We have developed the two corresponding decentralized algorithms P2P-NG and WF-DECA for these tasks.

Let us consider a SOMEWHERE P2PIS $\mathcal{P} = \{P_i\}_{i=1..n}$, where each P_i is a peer

with its own theory, also denoted by P_i. In each peer theory P_i, we distinguish the set M_i of mappings that the peer P_i has with other peers from the set O_i containing clauses involving only its local variables. In addition, we assume that each mapping of P_i has a unique identifier prefixed by P_i. The *global theory* $\mathcal{T}(\mathcal{P})$ of $\mathcal{P}$ is: $\bigcup_{i=1..n} P_i$, which is also equal to $\mathcal{M} \cup \mathcal{O}$ where $\mathcal{M} = \bigcup_{i=1..n} M_i$ and $\mathcal{O} = \bigcup_{i=1..n} O_i$.

We recall the semantics of such a P2PIS, using the notations we have just presented above and we make explicit the notion of consistency of it.

Definition 13 *Semantics and consistency of a P2PIS*
Let $\mathcal{P} = \{P_i\}_{i=1..n}$ be a P2PIS, an interpretation *I of $\mathcal{P}$ is an assignment of the variables of $\mathcal{T}(\mathcal{P})$ to true or $false$. I is a* model of a clause *c if and only if one of the literals of c is evaluated to true in I. I is a* model of a set of clauses *iff it is a model of all the clauses of the set. $\mathcal{P}$ is* consistent *if and only if $\mathcal{T}(\mathcal{P})$ has a model.*

Considering the decentralized and multi-authored nature of the peer theories, even if each local theory is consistent, the global theory may be inconsistent. We present an example of an inconsistent P2PIS in Figure 4.1.

- P_1 can be asked by researchers for choosing where to submit their results (demos or papers). For instance, part of its knowledge can model that: $PODS06$ is open for submission; submitting to $PODS06$ entails submitting to $PODS$; only theoretical results are submitted to $PODS$; a demo cannot be submitted to JAIR.

- P_2 distinguishes proceedings from journals and knows that: submitting to PODS entails submitting to a conference with proceedings; submitting to JAIR entails submitting to a journal; a same result cannot be submitted in a conference and in a journal; patented results cannot be submitted to a journal.

- P_3 has some knowledge about research valorization policy: software should be patented or presented as demos; theoretical results should be submitted to journals.

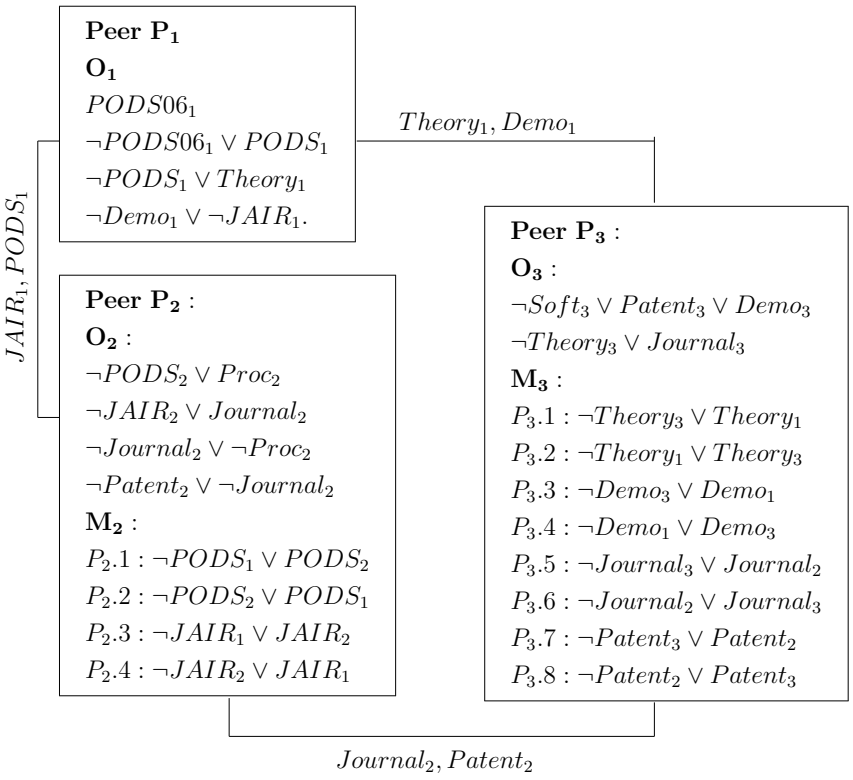

Figure 4.1: Exemple d'un rseau SomeWhere

- The knowledge expressed separately by P_1, P_2 and P_3 using their respective vocabularies can be respectively modeled by the set of clauses O_1, O_2 and O_3.
- The set M_2 of mappings stored at P_2 states the equivalence between $PODS_1$ and $PODS_2$ (reps. $JAIR_1$ and $JAIR_2$) through the mappings identified by $P_2.1$, $P_2.2$, $P_2.3$ and $P_2.4$. The set M_3 of mappings stored at P_3 also establishes equivalences between variables of P_3 and variables of the two other peers. Note however that mappings clauses do not necessarily result from equivalences.

In this example, the set of the clauses $\{\neg Journal_3 \vee Journal_2$, $\neg Theory_3 \vee Journal_3$, $\neg Theory_1 \vee Theory_3$, $\neg PODS_1 \vee Theory_1$, $\neg PODS06_1 \vee PODS_1$, $PODS06_1$, $\neg Journal_2 \vee \neg Proc_2$, $\neg PODS_2 \vee Proc_2$, $\neg PODS_1 \vee PODS_2\}$ is

inconsistent. Among these clauses, there are 3 mappings : $\neg Journal_3 \vee Journal_2$ (P_3.5) , $\neg Theory_1 \vee Theory_3$ (P_3.2) and $\neg PODS_1 \vee PODS_2$ (P_2.1).

Intuitively, the inconsistencies are due to the different viewpoints of different peers. However, given the lack of a centralized control, all peers should be treated equally. It would be unfair to exclude any peer out of the network, hoping of restoring the consistency for the P2PIS, or to refuse the join of a new peer just because the resulting P2PIS becomes inconsistent.

Our choice is to accept the presence of inconsistency. The problem is first to *detect* inconsistencies, second to *reason* in spite of them in a satisfactory way. Our approach is to compute only *well-founded* consequences of a formula, i.e., those that are consequences of the formula w.r.t. to a consistent subset of the global theory.

Such an approach is not novel in the centralized case, such as (59) that tried to point out the faulty axioms in a TBox, or (64) and (40) that defined a new notion of entailment based on the classical one w.r.t. a consistent subset of the whole inconsistent theory. However, in our decentralized case, such an approach raises new algorithmic issues because both the computation and the storage of *nogoods* (accounting for inconsistencies) are distributed. As the reasoning is distributed, the set of the formulas involved in a proof is also distributed. Thus, one has to be able to check the consistency of distributed sets of formulas, w.r.t. distributed sets of nogoods.

We assume each local theory to be consistent. Therefore, the possible inconsistencies result from the interaction between local theories, and are caused by *mappings*. Before adding a mapping, a peer checks whether this mapping (possibly with other mappings) can be the cause of some inconsistency, i.e., if the empty clause can be produced as one of its consequences. In that case, the peer stores locally as a *nogood* the set of mapping identifiers involved in the corresponding proof. At reasoning time, the concerned distributed nogoods must be collected to check whether the proof under construction is well-founded. This requires collecting for each result the mapping support of its derivations together

with the nogoods stored at each peer visited by the reasoning. The only results that are returned are those with a mapping support not including a nogood.

4.1 Peer-to-peer detecting inconsistencies and nogoods

As outlined in the introduction, even if each P_i is locally consistent, this is not necessarily the case for $\mathcal{T}(\mathcal{P})$. We assume that the causes of inconsistencies are only due to mappings and we define a **nogood** as follows:

Definition 14 *nogood*
A ***nogood*** ng *is a set of mappings such that* $\mathcal{O} \cup ng$ *is inconsistent.*

Note that in a nogood ng there necessarily exists some mapping m from which one can derive the empty clause (i.e., with a proof rooted in m). We exploit this property in the decentralized algorithm P2P-NG to detect nogoods accounting for inconsistencies.

4.1.1 The P2P-NG algorithm

P2P-NG runs at each peer and is used before adding a new mapping m to a peer P to check if the join of this mapping cause inconsistency, i.e. to check whether the propagation of m into the existing P2PIS produces the empty clause. If that is the case, P stores locally as new nogoods the mappings (including m) involved in the corresponding derivations.

A derivation of a clause c (possibly $\square$) is a proof of c using a complete strategy of resolution, i.e. a sequence of applications of the resolution rule which given two clauses $e \vee c_1$ and $\neg e \vee c_2$ produces the resolvent clause $c_1 \vee c_2$ (in which redundant literals are removed). A derivation rooted in m is a derivation the first application of resolution rule applies to m. In our setting, the input set of clauses are either reduced to a local theory of a given peer, or a union of local theories and mappings.

We call the set of mappings used in a derivation its **mapping support**. The P2P-NG algorithm computes the (possibly empty) set of mapping support of the derivations of the empty clause rooted in m, starting at the peer P. Since mappings have a unique identifier, mapping support can efficiently be encoded as sets of mapping identifiers (and similarly for nogoods computed from mapping support).

As the DeCA algorithm is used in SomeWhere for consequence finding, P2P-NG is an adaptation of DeCA to find **all** the mapping support of the empty clause. It follows the same split-recombine strategy as DeCA but it has the following significant differences:

- The stopping conditions **must** be changed. The ones in the original DeCA algorithm were designed for the computation of *proper prime* implicates only. Here we need to find all the possible ways of deriving the empty clause, we cannot stop the reasoning as soon as we produce the empty clause, or as soon as we find a unit clause in a local peer which is the same as the literal under processing.

- Because we are only looking for $\square$ as a consequence, locally produced consequences c such that $L(c) \neq \square$ can be filtered out.

- While DeCA would return $\{\square\}$ as its result if there exists a derivation of the empty clause, P2P-NG returns as many mapping support as different ways of deriving $\square$.

We use the following notations:

- For a clause c, $Resolvent{+}SMS(c, P)$ computes the set of pairs $(r\ SMS(r, P))$ such that r is a local consequence of c w.r.t. P and $SMS(r, P)$ is its corresponding set of local mapping support.

- For any clause r that is a local consequence of a clause c w.r.t. a peer P, $L(r)$ denotes the disjunction of literals of r the variables of which are not shared in P, and $S(r)$ denotes the disjunction of literals of r the variables of which are shared in P, i.e. $r = L(r) \vee S(r)$.

- For a literal q, $\bar{q}$ denotes its complementary literal, $ACQ(q, P)$ denotes the set of peers that share the variable q with P.
- A history $hist$ is a sequence of tuples (l, P, c) where l is a literal, P a peer, and c a clause which is a consequence of l on the peer P. A history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, which either has produced locally the clause c_0 in P_0 (in that case, c_0 may have been splitted into its different literals among which l_1 is propagated in P_1), or not (in that case l_0 is simply propagated from P_0 to P_1 and $l_0 = c_0 = l_1$). For every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1}.
- $\otimes$ is the distribution union operator on sets of sets: $SS_1 \otimes \cdots \otimes SS_n = \{S_1 \cup \cdots \cup S_n \,|S_1 \in SS_1, \ldots, S_n \in SS_n\}$. If $L = \{l_1, \ldots, l_p\}$, $\otimes_{l \in L} SS_l$ denotes $SS_{l_1} \otimes \cdots \otimes SS_{l_p}$.

Each literal resulting from the splitting of a clause (Line 11) of the P2P-NG$_l$ algorithm) is processed independently by the P2P-NG$_l$ algorithm. P2P-NG$_l(q, SP, hist)$ checks whether there exists a derivation of $\square$ rooted in the literal q, starting with the computation of local consequences of q, and then recursively following the acquaintances of the visited peers. To ensure termination, it is necessary to keep track of the literals already processed by peers. This is done thanks to $hist$, where $hist$ is the history of the reasoning branch ending up to the propagation of the literal q in SP, which is the set of acquaintances of the last peer added to the history.

Before adding a new mapping m to its local theory, each peer P first computes P2P-NG(m, P) and stores locally all the nogoods $\{m\} \cup ms$ (see theorem 3), such that $ms \in$ P2P-NG(m, P) is minimal (for set inclusion).

Illustrative example (continued)

In the example of Figure 4.1, let us suppose that the different peers join in the following order: P_1, then P_2, then P_3 and that their respective mappings are added according to their numbering. At the join of P_2, the successive adding of the 4

Algoritm 5: Detection of the nogoods caused by adding a mapping

P2P-NG(m, P)

(1) LOCAL$(P) \leftarrow Resolvent{+}SMS(m, P)$
(2) RESULT $\leftarrow \emptyset$
(3) **foreach** $(c\ sms) \in$ LOCAL(P) s.t. $S(c) \neq \square$ and $L(c) = \square$
(4) **foreach** literal $q \in S(c)$
(5) NOGOODS$(q) \leftarrow$ P2P-NG$_l(q,$ ACQ$(q, P), \emptyset)$
(6) **if** for every $q \in S(c)$ NOGOODS$(q) \neq \emptyset$
(7) UNIONCOMB $\leftarrow sms \otimes (\otimes_{q \in S(c)}$NOGOODS$(q))$
(8) RESULT $\leftarrow$ RESULT $\cup$ UNIONCOMB
(9) **return** RESULT

P2P-NG$_l(q, SP, hist)$

(1) **if** for every $P \in SP$, $(q, P, \_) \in hist$
(2) **return** $\emptyset$
(3) **else**
(4) RESULT $\leftarrow \emptyset$
(5) **if** $(\bar{q}, \_, \_) \in hist$
(6) RESULT $\leftarrow$ RESULT $\cup \{\emptyset\}$
(7) **foreach** $P \in SP$
(8) LOCAL$(P) \leftarrow \{(q\ \{\emptyset\})\} \cup Resolvent{+}SMS(q, P)$
(9) RESULT $\leftarrow$ RESULT $\cup \bigcup_{P \in SP} SMS(\square, P)$
(10) **foreach** $P \in SP$ and $(c\ sms) \in$ LOCAL(P) s.t. $S(c) \neq \square$ and $L(c) = \square$
(11) **foreach** literal $l \in S(c)$
(12) SMS$(l) \leftarrow$ P2P-NG$_l(l,$ ACQ$(l, P), [(q, P, c)|hist])$
(13) **if** for every $l \in S(c)$, SMS$(l) \neq \emptyset$
(14) UNIONCOMB $\leftarrow sms \otimes (\otimes_{l \in S(c)}SMS(l))$
(15) RESULT $\leftarrow$ RESULT $\cup$ UNIONCOMB
(16) **return** RESULT

mappings causes no inconsistency. When P_3 joins, the 4 first mappings cause no inconsistency. Let us focus on the 5th one. P2P-NG$(\neg Journal_3 \vee Journal_2, P_3)$ is triggered where P_3 is the theory containing the clauses of P_3 that are not mappings, and the 4 first mappings (which have been added since they have

been checked as not deriving inconsistencies). $\neg Theory_1 \vee Journal_2$ is produced locally at P_3 as a local consequence of $\neg Journal_3 \vee Journal_2$, with a set of local mapping support equal to $\{\{P_3.2\}\}$. Then, $\neg Theory_1 \vee Journal_2$ is split (Line 11 of P2P-NG$_l$): $\neg Theory_1$ is processed by P_1, while $Journal_2$ is processed by P_2. The propagation of $\neg Theory_1$ produces $\square$ as a local consequence in P_1 with a local set of mapping support equal to $\{\emptyset\}$. Thus $\{\emptyset\}$ is returned to P_3 as the set of mapping support of the derivation of $\square$ from $Theory_1$. The propagation of $Journal_2$ produces $\neg PODS_1$ as a local consequence in P_2, with a local set of mapping support equal to $\{\{P_2.1\}\}$, as well as $\neg Patent_2$ with a local set of mapping support equal to $\{\emptyset\}$. $\neg PODS_1$ is in turn propagated in P_1, where it produces $\square$ as a local consequence with a local set of mapping support equal to $\{\emptyset\}$. It is transmitted back to P_2 which, after combination of $\{\emptyset\}$ and $\{\{P_2.1\}\}$ (Line 14) of P2P-NG$_l(Journal_2, \{P_2\}, \emptyset)$), transmits back to P_3 the set of mapping support $\{\{P_2.1\}\}$ for the derivation $\square$ from $Journal_2$. By combination (Line 7) P2P-NG$(\neg Journal_3 \vee Journal_2, P_3)$ returns $\{\{P_3.2, P_2.1\}\}$. The nogood $\{P_3.5, P_3.2, P_2.1\}$ is thus obtained and stored at P_3. No other nogood is obtained from the last mappings of P_3.

4.1.2 Termination, soundness and completeness of P2P-NG

Theorem 2 Termination
P2P-NG *terminates.*

Proof: At each recursive call, a new triple (q, P, c) is added to the history. If the algorithm did not terminate, the history would be infinite, which is not possible since the number of peers, literals and clauses within a P2PIS is finite. $\square$.

Theorem 3 states that the result of P2P-NG(m, P) can be used to characterize the nogoods involving the mapping m (by assimilating mappings to their corresponding index). The proof of Theorem 3 relies on Lemma 1.

Theorem 3 Soundness
Let m be a mapping and P be a peer such that P2P-NG$(m, P) \neq \emptyset$. $\forall ms \in$ P2P-NG(m, P), $ms \cup \{m\}$ *is a nogood.*

PROOF:
Suppose that P2P-NG$(m, P) \neq \emptyset$, let us denote the set of results of P2P-NG(m, P) by $\mathcal{A}$. For every $ms \in \mathcal{A}$, we will prove that ms is the mapping support of a derivation of $\square$ rooted in m.

Because P2P-NG$(m, P) \neq \emptyset$, the conditions at lines (3) and (6) are satisfied and each ms of $\mathcal{A}$ is found at line (7):

- (3) is satisfied means that there is some clause c that is in LOCAL(P) such that $S(c) \neq \square$ and $L(c) = \square$,
- for every such a clause c, sms denotes the set of mapping support of the derivation of c rooted in m,
- for every such clause c, (6) is satisfied means that in line (5), the result of P2P-NG$_l(q,$ ACQ$(q, P), \emptyset)$ (which is NOGOODS(q)), is $\neq \emptyset$, for each $q \in S(c)$. Because of Lemma 1, for every $q \in S(c)$, every $ms_q \in$ NOGOODS(q) is the mapping support of a derivation of $\square$ rooted in q. Therefore, in line (7), $\otimes_{q \in S(c)}$NOGOODS(q) consists of sets of mappings, each of them is the mapping support of a derivation of $\square$ rooted in $S(c)$.

  Each set ms of mappings of $\mathcal{A}$ (that is added into UNIONCOMB at line (7)) consists of the mapping support of a derivation of $\square$ rooted in $S(c)$ (corresponding to one of the sets of $\otimes_{q \in S(c)}$NOGOODS(q)) and the mapping support of a derivation of c rooted in m (corresponding to one of the set of the mappings in sms). Therefore, ms is the mapping support of a derivation of $\square$, rooted in m.

As every $ms \in \mathcal{A}$ is the mapping support of a derivation of $\square$ rooted in m, $ms \cup \{m\} \cup \mathcal{O}$ is inconsistent and $ms \cup \{m\}$ is a nogood, by definition. $\square$.

Lemma 1 *If* P2P-NG$_l(q, SP, hist) \neq \emptyset$ *then* $\forall ms \in$ P2P-NG$_l(q, SP, hist)$, ms *is the mapping support of a derivation of* $\square$ *rooted in* q.

PROOF: We prove by induction on number rc of recursive calls required by P2P-NG$_l(q, SP, hist)$ to terminate.

Let us denote the set of results of P2P-NG$_l(q, SP, hist)$ by $\mathcal{A}$. For every $ms \in \mathcal{A}$, we will prove that ms is the mapping support of a derivation of $\square$ rooted in q.

- $rc = 0$: Every ms of $\mathcal{A}$ is found at Line (6) or Line (9) . Each ms correspond to the mapping support of a derivation of $\square$ rooted in q, because it uses either $\bar{q}$ in the history, or clauses that are local to a peer P of SP, for deleting q and deduce $\square$.

- Suppose the induction hypothesis true for $rc \leq p$, and let SP be a set of peers of a P2PIS such that P2P-NG$_l(q, SP, hist)$ requires $p + 1$ recursive calls to terminate. Because there is at least one recursive call, the conditions in the line (10) and (13) are satisfied.

  The result $\mathcal{A}$ of P2P-NG$_l(q, SP, hist)$ contains:

  - the sets of mappings found at Line (6)and Line (9) , which, by the same argument as in the base case, are sets of mapping support of derivations of $\square$ rooted in q.
  - and the sets of mappings found at line (15).

- (10) is satisfied means that there is some $P \in SP$ such that $(c\ sms) \in$ LOCAL(P), $S(c) \neq \square$, $L(c) = \square$,

- for every such a peer P and a clause c, sms denotes the set of mapping support of c rooted in q,

- for every such a peer P and clause c, (13) is satisfied means that the result of P2P-NG$_l$ $(l, \text{CQ}(l, P), [(q, P, c)|hist])$ which is $SMS(l)$, is $\neq \emptyset$, for each $l \in S(c)$. According to the induction hypothesis (the number of recursive calls for P2P-NG$_l$ to terminate is less than or equal to p), every $ms_l \in SMS(l)$ is the mapping support of a derivation of $\square$ rooted in l. Therefore, in line (14), $\otimes_{l \in S(c)} SMS(l)$ consists of sets of mappings, each of them is the mapping support of a derivation of $\square$ rooted in $S(c)$,

Each set ms of mappings of $\mathcal{A}$ (that is added into UNIONCOMB at line (15)) consists of the mapping support of a derivation of $\square$ rooted in $S(c)$ (corresponding to one of the sets of $\otimes_{l \in S(c)} SMS(l)$) and the mapping support of a derivation of c rooted in q (corresponding to one of the set of mappings in sms). Therefore, ms is the mapping support of a derivation of $\square$, rooted in q.

$\square$.

Theorem 4 states that the P2P-NG algorithm is complete, i.e., it enables to find the mapping support of all the irredundant derivations of the empty clause from a given mapping added to a P2PIS. The proof of Theorem 4 relies on Lemma 2.

Definition 15 *Irredundant derivation*
A derivation is irredundant if it does not involve two identical applications of the resolution rule.

Theorem 4 Completeness *Let $\mathcal{P}$ be a P2PIS and m a mapping of a given peer P of $\mathcal{P}$. Let ms be a mapping support of an irredundant derivation of $\square$ rooted in m. It will be returned by* P2P-NG(m, P).

PROOF: Since we suppose that each local theory is consistent, the mapping support ms is not empty. Suppose that each local consequence c of m obtained by a derivation within P includes a literal the variable of which is not shared. This literal has not been deleted by a resolution within P and it cannot be deleted by a resolution with clauses outside P since its variable is not shared. This contradicts the existence of a derivation of $\square$ rooted in m with a non empty mapping support. Therefore, the derivation of $\square$ rooted in m having ms as its mapping support involves at least one local consequence c of m in LOCAL(P) such that $S(c) \neq \square$ and $L(c) = \square$. From Lemma 2, we can infer that ms is the union of the local mapping support of c and of mapping support of irredundant derivations of $\square$ rooted in q for every literal q in c.

To finish the proof, we have to prove that P2P-NG$_l(q,$ ACQ$(q, P), \emptyset)$ returns all the mapping support of irredundant derivations of $\square$ rooted in q. For doing

so, we prove by induction on number rc of recursive calls required by P2P-NG$_l(q, SP, hist)$ to terminate that P2P-NG$_l(q, SP, hist)$ returns the set of mapping support of all the irredundant derivations of $\square$ rooted in q and using clauses of $\mathcal{P}(hist)$ where $\mathcal{P}(hist) = \mathcal{P} \cup \{l_i | (l_i, \_, \_) \in hist\}$.

- $rc = 0$: If the condition of Line (1) is satisfied, it does not exist irredundant derivations of $\square$ rooted in q using clauses of $\mathcal{P}(hist)$: all the derivations of $\square$ that can be produced from the last occurrence of q and starting from clauses of SP contain applications of the resolution rule that are identical to the ones that were produced from the occurrence of q in the history. Let suppose that there exists irredundant derivations of $\square$ rooted in q using clauses of $\mathcal{P}(hist)$. Since in this case the condition of Line (1) is not satisfied, we are in the case where for each $P \in SP$ and for each $c \in \text{LOCAL}(P)$ the condition $S(c) \neq \square$ and $L(c) = \square$ is not satisfied, i.e., $S(c) = \square$ or $L(c) \neq \square$. Therefore, all the irredundant derivations of $\square$ rooted in q, and using clauses of $\mathcal{P}(hist)$, either involve $\bar{q}$ which is in the history, or are derivations involving only local clauses of a peer P of SP. If $\bar{q}$ is in the history, then the corresponding irredundant derivation of $\square$ has a mapping support which is empty, which will be returned in Line (6) of the algorithm. All the other irredundant local derivations will be found by *Resolvent+SMS*(q,P) and the algorithm will return the set of their mapping support in Line (9) of P2P-NG$_l$.

- Suppose the induction hypothesis true for $rc \leq p$, and let SP be a set of peers of a P2PIS such that P2P-NG$_l(q, SP, hist)$ requires $p + 1$ recursive calls to terminate. Since there is at least one recursive call, the condition of Line (1)is not satisfied.

  The results returned by the algorithm at Line (6) and Line (9) correspond to the mapping support of the irredundant derivation of $\square$ rooted in q and using either $\bar{q}$ in the history or clauses that are local to a peer P of SP.

  Consider now an irredundant derivation d of $\square$ rooted in q and using a mapping outside SP. Such a derivation necessarily requires (i) either that the variable of q is shared (ii)or involves a local consequence c obtained by

a derivation rooted in q and using clauses of a given P of SP such that $S(c) \neq \square$ and $L(c) = \square$. $S(c) \neq \square$ is required in order to have external clauses from SP in the derivation. $L(c) = \square$ is required for getting the empty clause as result of the derivation.

(i) In the first case, q is involved in an iteration of the loop of Line (11). According to the induction hypothesis (the number of recursive calls to obtain SMS(q) in Line (12) is less than or equal to p) SMS(q) includes all the mapping support of irredundant derivations of $\square$ rooted in q and using clauses of $\mathcal{P}(hist) \cup \{q\}$. Therefore, SMS($q$) includes the mapping support of the derivation d.

(ii) In the second case, the mapping support of the irredundant derivation d is the union of the mapping support of the local derivation of c and of the mapping support of the (irredundant) derivation d' of $\square$ corresponding to the part of the derivation d rooted in c. c is involved in an iteration of the loop of Line (11). According to the induction hypothesis (the number of recursive calls to obtain SMS(l) in Line (12)is less than or equal to p), for every $l \in S(c)$, SMS(l) includes all the mapping support of irredundant derivations of $\square$ rooted in l and using clauses of $\mathcal{P}(hist) \cup \{q\}$. According to Lemma 2, the mapping support of d' is the union of mapping support of irredundant derivations of $\square$ rooted in l and using clauses of $\mathcal{P}(hist) \cup \{q\}$. It will belong to $(\otimes_{l \in S(c)}$SMS$(l))$ and therefore, the mapping support of the derivation d will be returned by Line (15).

$\square$.

Lemma 2 *Let ms be the mapping support of an irredundant derivation of $\square$ rooted in a clause c: $c_1 \vee \ldots \vee c_n$ where every c_i is a clause (which can be a unit clause or not) such that there is no literal common to c_i and c_j for $i \neq j$. There exists $ms_1, \ldots, ms_n$ where, for every i, ms_i is a mapping support of an irredundant derivation of $\square$ rooted in c_i, such that: $ms = ms_1 \cup \ldots \cup ms_n$.*

PROOF:

We prove the lemma by induction on the length of the (irredundant) derivation of $\square$.

-If the length of the derivation is 2: $c = l_1 \vee l_2$, and the first step of the derivation uses $\bar{l}_1$ (or $\bar{l}_2$) and the second step uses $\bar{l}_2$ (or $\bar{l}_1$). The mapping support ms of the derivation of $\square$ is the set of mappings contained in $\{\bar{l}_1, \bar{l}_2\}$. Suppose that both $\bar{l}_1$ and $\bar{l}_2$ are mappings (the other cases are similar). There is a derivation of $\square$ rooted in l_1 using $\bar{l}_1$ (i.e., with a mapping support ms_1: $\{\bar{l}_1\}$) and there is a derivation of $\square$ rooted in l_2 using $\bar{l}_2$ (i.e., with a mapping support ms_2:$\{\bar{l}_2\}$). We have: $ms = ms_1 \cup ms_2$.

- Suppose the property true for irredundant derivations of $\square$ of length less than p and consider a derivation d of $\square$ of length p and rooted in $c = c_1 \vee c_2 \ldots \vee c_n$. Let ms be its mapping support. Without loss of generality, let $c_1 = l_1 \vee rc_1$ and let $c' = \bar{l}_1 \vee rc'$ be the clause involved in the first step of the derivation. Let MSS be the mapping support of the derivation of $\square$ rooted in the resolvent $rc_1 \vee c_2 \vee \ldots \vee c_n \vee rc'$, which has a length of $p - 1$.

- If c' is a mapping identified by $P.i$, we have: $MS = MSS \cup \{P.i\}$, otherwise $MS = MSS$.

- By induction hypothesis, there exists $MS_{rc_1}, MS_2, \ldots, MS_n$ and MS' such that MS_{rc_1} is a mapping support of an irredundant derivation of $\square$ rooted in rc_1, for every $i \in [2..n]$ MS_i is a mapping support of a irredundant derivation of $\square$ rooted in c_i, and MS' is a mapping support of an irredundant derivation of $\square$ rooted in rc', and $MSS = MS_{rc_1} \cup MS_2 \cup \ldots \cup MS_n \cup MS'$.

- Let us consider the (irredundant) derivation of $\square$ rooted in l_1 the first step of which involves the clause c', and the others steps involve the clauses in d which are used to eliminate the literals of c'. Let ms_{l_1} be its mapping support. We have: $ms_{l_1} = ms'$ if c' is not a mapping, $ms_{l_1} = ms' \cup \{P.i\}$ if c' is a mapping identified by $P.i$.

- If there exists an irredundant derivation of $\square$ rooted in rc_1 and an irredundant derivation of $\square$ rooted in l_1, there exists an irredundant derivation of $\square$ rooted in $rc_1 \vee l_1$ using the union of the clauses involved in the two derivations.

- Let $ms_1 = ms_{l_1} \cup ms_{rc_1}$. Since ms_{l_1} is a mapping support of an irredundant derivation of $\square$ rooted in l_1 and ms_{rc_1} is a mapping support of an irredundant derivation of $\square$ rooted in rc_1, ms_1 is a mapping support of an irredundant derivation of $\square$ rooted in c_1.
- Therefore, we have: $ms = ms_1 \cup ms_2 \cup \ldots \cup ms_n$ where for every i, ms_i is a mapping support of an irredundant derivation of $\square$ rooted in c_i.

$\square$.

Corollary 1 results directly from Theorem 4. It guarantees that all the minimal nogoods are computed and stored in the P2PIS. It is the key for proving that the reasoning algorithm presented in the next section is well-founded.

Corollary 1 *Let ng a nogood; let m a mapping of ng such that, among the mappings of ng, m is the last mapping added into the P2PIS; let P the peer stores this mapping m: ng is stored at P*

4.2 Peer-to-peer well-founded reasoning

In this section, we present our algorithm WF-DeCA that exploits the results of P2P-NG to compute *well-founded* consequences of a query w.r.t. an inconsistent P2PIS. First, we have to define the notion of *well-founded* consequences.

Definition 16 *P2P well-founded implicate*
Let $\mathcal{P}$ be an inconsistent P2PIS: r is a well-founded *implicate of c w.r.t. $\mathcal{P}$ if r is an implicate of c w.r.t. a* consistent subset *of $\mathcal{T}(\mathcal{P})$.*

4.2.1 The WF-DeCA algorithm

The WF-DeCA(q, P) algorithm computes well-founded consequences of the (unit) clause q, starting at the peer P. This algorithm extends the original DeCA consequence finding algorithm by computing the set of mapping support of the derivations for each consequence, and by collecting the nogoods encountered during the reasoning. Because of the split/recombination technique used by the algorithm,

mapping support of derivations are only known after the recombination step, and the set of possibly relevant nogoods must be available at this step: if some mapping support includes a nogood, it is discarded; consequences that get an empty set of mapping support after nogoods filtering are discarded as well.

We use the following notations :

- $LocalConsSSNG(q, P)$ is a local procedure that computes the set of triples (c sms sng) such that c is a local consequence of q w.r.t. P, sms is its corresponding set of local mapping support, and sng is the set of nogoods stored at the peer P that contain a mapping m of some local mapping support ms of c.
- $\uplus$ denotes the *merged union* of sets of consequences, i.e. the union of sets of triples of the form (c sms sng), where triples corresponding to a same consequence c are merged together, by computing the union of their respective sms and sng.
- $\circledcirc$ is the distribution union operator on sets of triples of the form (c sms sng): $S_1 \circledcirc \cdots \circledcirc S_n = \{(c_1 \vee \cdots \vee c_n \; sms_1 \otimes \cdots \otimes sms_n \; sng_1 \cup \cdots \cup sng_n) / (c_1 \; sms_1 \; sng_1) \in S_1, \ldots, (c_n \; sms_n \; sng_n) \in S_n\}$.

Illustrative example (continued)
Let us illustrate the behavior of WF-DeCA($Soft_3, P_3$), assuming that the only target variables are $PODS_1$ and $JAIR_1$. $Patent_2 \vee Demo_1$ is the only clause produced locally on P_3 with a local part of which (i.e. $\square$) in $\mathcal{T}arget(P_3)$. Its local sms is $\{\{P_3.7, P_3.3\}\}$. The only nogood stored at P_3 contains neither $P_3.7$ nor $P_3.3$. The corresponding sng returned by $LocalConsSSNG(Soft, P_3)$ is thus empty. $Patent_2 \vee Demo_1$ is then split. When $Patent_2$ is transmitted to P_2, $\neg JAIR_1$ is the only clause produced locally with a local part (i.e. $\square$) in $\mathcal{T}arget(P_2)$. Its local sms is $\{\{P_2.3\}\}$ and its sng is empty. The further propagation of $\neg JAIR_1$ returns an empty result. So the triple ($\neg JAIR_1$ $\{\{P_2.3\}\}$ $\emptyset$) is sent back to P_3 as a consequence of $Patent_2$. When $Demo_1$ is transmitted to P_1 the only clause produced locally is $\neg JAIR_1$, which is in $\mathcal{T}arget(P_1)$. Its local sms is empty, as

Algoritm 6: Well Founded Distributed Consequence Finding Algorithm WF-DeCA(q, P)

(1) WF-DeCAH$(q, \{P\}, \emptyset)$

WF-DeCAH$(q, SP, hist)$

(1) **if** for every $P \in SP$, $(q, P, \_) \in hist$
(2) **return** $\emptyset$
(3) **else if** $(\bar{q}, \_, \_) \in hist$
(4) **return** $\{(\Box\ \{\emptyset\}\ \emptyset)\}$
(5) **else**
(6) RESULT $\leftarrow \emptyset$
(7) **foreach** $P \in SP$
(8) LOCAL$(P) \leftarrow \{(q\ \{\emptyset\}\ \emptyset)\} \uplus LocalConsSSNG(q, P)$
(9) **foreach** $P \in SP$ and $(c\ sms\ sng) \in$ LOCAL(P) such that $L(c) \in \mathcal{T}arget(P))$
(10) **if** $S(c) \in \mathcal{T}arget(P)$
(11) RESULT $\leftarrow$ RESULT $\uplus \{(c\ sms\ sng)\}$
(12) **if** $S(c) \neq \Box$
(13) **foreach** literal $l \in S(c)$
(14) ANSWER$(l) \leftarrow$
(15) WF-DeCAH$(l,$ ACQ$(l, P), [(q, P, c)|hist])$
(16) **if** for every $l \in S(c)$ ANSWER$(l) \neq \emptyset$
(17) UNIONCOMB $\leftarrow$
(18) $\{(L(c)\ sms\ sng)\} \otimes (\otimes_{l \in S(c)}$ANSWER$(l))$
(19) **foreach** $(c\ sms\ sng) \in$ UNIONCOMB
(20) $nsms \leftarrow \{ms \in sms / \forall ng \in sng, ng \not\subseteq ms\}$
(21) **if** $nsms \neq \emptyset$
(22) RESULT $\leftarrow$ RESULT $\uplus \{(c\ nsms\ sng)\}$
(23) **return** RESULT

well as its *sng*. So the triple $(\neg JAIR_1\ \{\emptyset\}\ \emptyset)$ is sent back to P_3 as a consequence of $Demo_1$. P_3 then combines these two triples obtained from P_2 and P_1 giving $(\neg JAIR_1\ \ \{\{P2.3\}\}\ \emptyset)$, which is the only final consequence of $Soft_3$ being in the target language. Since the corresponding *sng* is empty, $\neg JAIR_1$ is trivially a well-founded consequence.

4.2.2 Termination and Soundness of WF-DeCA

Theorem 5 states that the WF-DeCA(q, P) algorithm terminates and returns only well-founded consequences. Its proof relies on showing that each triple $(r\ sms_r\ sng_r)$ returned by WF-DeCA$(q, SP, hist)$ is such that either r is a local consequence of q w.r.t. some peer $P \in SP$, or for every $ms_r \in sms_r$, $\mathcal{O} \cup ms_r$ is consistent (using Corollary 1) and r is the result of a derivation rooted in q that only uses clauses that do not containt any nogood stored in the P2PIS.

Theorem 5 *Let P be a peer of a P2PIS $\mathcal{P}$ and q a literal belonging to the vocabulary of P.* WF-DeCA(q, P) *terminates and for all triples $(r\ sms_r\ sng_r)$ returned by* WF-DeCA(q, P)*, r is a well-founded consequence of q w.r.t. $\mathcal{P}$.*

Proof:
Termination At each recursive call, a *new* triple (sl, P, c) is added to the history. If the algorithm did not terminate, the history would be infinite, which is not possible since the number of peers, literals and clauses within a P2PIS is finite.

Soundness By definition, r is a well-founded consequence of q w.r.t. $\mathcal{P}$ if it is a consequence of q w.r.t. a consistent subset of $\mathcal{P}$. Let $(r\ sms_r\ sng_r)$ a triple returned by WF-DeCA(q, P).

- If $(r\ sms_r\ sng_r)$ is found without any recursive call of WF-DeCAH then r is a local consequence of q and P. r is by definition a well-founded consequence.

- If $(r\ sms_r\ sng_r)$ is returned after at least one recursive call of WF-DeCAH then it is found at line (17) in the form $(r\ bsms_r\ sng_r)$ such that sms_r contains the mapping support that are not filtered out from $bsms_r$ by the filter from the line (19) to (22). By applying Lemma 3, we know that for each $bms_r \in bsms_r$, all the nogoods that are contained in bms_r are collected in sng_r. Therefore, after the filter from the line (19) to (22), each ms_r in sms_r is a mapping support such that there is no nogood stored in the P2PIS that is contained in it. Consequently, we are sure that r is deduced from q w.r.t. a consistent subset of $\mathcal{P}$. r is thus well-founded.

□.

Lemma 3 *Let P be a peer of a P2PIS $\mathcal{P}$ and q a literal belonging to the vocabulary of P. Let $(r\ sms_r\ sng_r)$ be a triple found at line (8) (in* LOCAL(P)*) or at line (17) (in* UNIONCOMB*) by* WF-DECAH$(q, P, \emptyset)$*. For each $ms_r \in sms_r$, all the nogoods stored in the P2PIS that are contained in ms_r are collected in sng_r.*

PROOF:

- If the triple $(r\ sms_r\ sng_r)$ is found at line (8) (in LOCAL(P)), all the mappings support $ms_r \in sms_r$ contain local mappings to P.
  For each $ms_r \in sms_r$, suppose that there exists a nogood ng contained in ms_r. Since ms_r is local to P, all the mappings in ng are local to P too. Therefore, ng is necessarily stored at P. Consequently, ng has been collected by the procedure $LocalConsSSNG(q, P)$, and is then in sng_r.

- If the triple $(r\ sms_r\ sng_r)$ is found at line (17) (in UNIONCOMB) then r is found by at least one recursive call of WF-DECAH . r is thus of the form $L(c) \vee A_1 \vee ... \vee A_k$ where:

  - $L(c)$ is the local part of a clause c found in the form $(c\ sms_c\ sng_c)$ by $LocalConsSSNG(q, P)$ when WF-DECAH is called the first time, i.e. WF-DECAH$(q, SP, \emptyset)$. We also have that each $ms_c \in sms_c$ contains only local mappings to P.
  - for each i=[1..k], A_i is returned by WF-DECAH$(l_i, ACQ(l_i, P), [(q, P, c))$ in the form $(A_i\ sms_{A_i}\ sng_{A_i})$ as a consequence of l_i, where l_i is a shared literal of the clause c.

  By definition of the operator $\oslash$, each ms_r in sms_r is of the form $ms_c \cup ms_{A_1}\ ...\ \cup ms_{A_k}$ where ms_c is a mapping support in sms_c and for each i=[1..k], ms_{A_i} is a mapping support in sms_{A_i}.

  Suppose that there exists a nogood ng contained in ms_r. Let m a mapping of ng such that, among the mappings of ng, m is the last mapping added into the P2PIS. Either m is in ms_c, or m is in one of the ms_{A_i}s.

– If m is in ms_c, it is thus stored at the peer P. By Corollary 1, ng is also stored at P and has been collected by $LocalConsSSNG(q, P)$ and is then in sng_c. By definition of the operator $\oslash$, $sng_r = sng_c \cup sng_{A_1} \ldots \cup sng_{A_k}$, therefore ng is in sng_r

– If m is in one of the ms_{A_i}s then let P' the peer that stores m. By Corollary 1, ng is also stored at P'. Because m is used, the reasoning must have passed by P', therefore ng has also been collected by the procedure $LocalConsSSNG$ running at P'. ng is thus in sng_{A_i}. By definition of the operator $\oslash$, $sng_r = sng_c \cup sng_{A_1} \ldots \cup sng_{A_k}$, therefore ng is in sng_r

□.

4.3 Implementation of P2P-NG and WF-DeCA and experimentations

In the following, we describe the adaptations/modifications that have been performed on SomeWhere in order to implement the P2P-NG and WF-DeCA algorithms. For a better understanding, we first present a brief overview of some particularities of the SomeWhere architecture. Then we present DeCAbL, (DeCA bounded length) an adaptation of the original DeCA algorithm. DeCAbL is a specialized version of DeCA that computes only the proper implicates (of a given formula) the length of which is bounded by some value, given as a parameter. This procedure has its own value since most of the time, the user is not necessarily interested in the set of all consequents, because this set can really be huge, but only in a limited number of those (e.g. the shortest ones). Knowing in advance the maximal size of the consequents to be produced, makes it possible to optimize DeCA and, as experimental results will show it, to improve considerably its performance. But the DeCAbL algorithm is also closely related to our concern, since it can easily be adapted in order to implement P2P-NG. It thus can be considered as a fruitful side effect of our work. The last subsection is then devoted to the presentation of the technical adaptations that have

been performed to achieve the implementation of both P2P-NG and WF-DeCA algorithms in SomeWhere.

4.3.1 An overview of the SomeWhere architecture

Setting up a SomeWhere network requires launching a software component called a SomeWhere *platform*. By the term SomeWhere *platform*, we refer to:

- a pair $(host, port)$, representing respectively the IP address and the port of the computer on which the platform runs
- and all the functionalities that a peer should be able to do, i.e. to launch, host its own theory, accept queries, run the DeCA algorithm and interact with other peers,...

The DeCA algorithm that has been implemented is the message-passing version presented in Section 2.2.

SomeWhere supports the classical paradigm of peer-to-peer architectures. In this case, each peer is managed by one instance of the SomeWhere *platform* on one computer. The resulting architecture is described in Figure 4.2, where a peer is represented by one smiley and a SomeWhere platform is figured by a black rectangle.

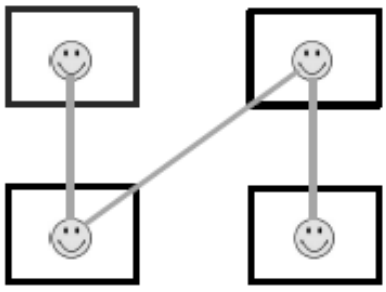

Figure 4.2: Architecture N peers - N SomeWhere platforms

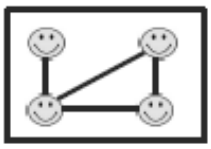

Figure 4.3: Architecture N peers - 1 SomeWhere platform

The SomeWhere *platform* can also be run in a mode where it can "hosts" several peers. In such a configuration, the platform acts as a *virtual* layer that manages internally the communications between the peers hosted by the platform. This mode is represented in Figure 4.3.

Because one platform has a single network address (i.e. one pair $(host, port)$), the peers inside a SomeWhere platform are referenced by their name. Note that even if several peers are hosted on the same SomeWhere platform, no centralization is made in any way: each clausal theory of each peer remains completely independent and the behavior of DeCA through the *virtual* network of peers is rigorously the same, as in the previous configuration.

The main advantage of this mode is that a single computer can be used. It can also saves a significant amount of time devoted to network communications between the peers. Moreover, it makes things more convenient, when data has to be collected from the different peers, to generate reports during experimentations. The main drawback is a single machine has to share its computing power, to support simultaneously the reasoning tasks of all the peers, as well as the communications between them. Depending on the number of peers and the complexity of the reasoning tasks, this can lead to a heavy load for a single machine.

Eventually, for large scale experimentations with a limited number of computers, The SomeWhere platforms is implemented in such a way that a same computer can run simultaneously several SomeWhere platforms, each of them hosting several peers and being associated with a different port number. The difference with the previous case is that communications between peers hosted by different platforms are routed through the network layer, exactly in the same way, as when the different platforms are running on different machines. This mode is

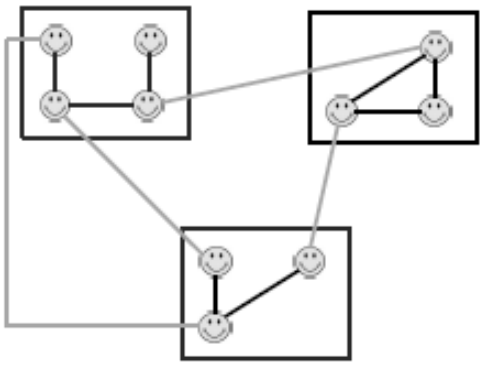

Figure 4.4: Architecture N peers - M SomeWhere platforms ($M < N$)

represented in Figure 4.4. Of course it is also possible to run different platforms on different machines, each of them hosting different peers. The different usages of the SomeWhere platform give a lot of flexibility when conducting experiments. Depending of what is expected to be measured.

4.3.2 DeCAbL : Decentralized bounded length consequence finding algorithm

As a preliminary step towards the implementation of P2P-NG and WF-DeCA, we first have designed and implemented an optimization of DeCA called DeCAbL. In comparison to DeCA, DeCAbL has a supplementary parameter, called lb, and only computes proper consequences of a query, whose length is at most lb.

Bounding the size of produced consequents is an interesting feature because in many cases, the number of these is so great, that it is difficult for the user to grasp all the information that is obtained. But most of the time, the user is only interested by the most informative consequents, i.e. those that correspond to the shortest clauses. Bounding the maximal size of the consequents to be produced, allows for an optimization of DeCA that basically amounts to cut all reasoning branches, as soon as we know that they can only contribute to produce consequents, the length of which would exceed the fixed bound. Experimental results

with this procedure, presented in the following, clearly show this can considerably improve the performance of the consequence finding task. The DeCAbL algorithm thus clearly has its own interest but, as we will see in the next section, it also has strong connections with P2P-NG.

The content of DeCAbL algorithm is described in Algorithm 7. It is very close to that of DeCA presented in Section 2.2 and we keep the same notation conventions. In addition, when *hist* represent a history list and m a clause, we use newLits$(m, hist)$ to denote the number of literals of m that do not appear in any clause c such that $(\_, \_, c)$ is an element of *hist*. The main difference with DeCA is that before returning a consequence m, DeCAbL checks the condition newLits$(m, hist) \leq lb$. The consequence in question is returned only if this condition is satisfied. A final filter on the length of the consequences found at line (16) of DeCAbL makes sure of its correctness.

Because DeCAbL is close to DeCA, the implementation of DeCAbL does not require to modify or extend the architecture of the SomeWhere platform itself, we still can benefit from the flexibility of the N peers - M SomeWhere platforms ($M < N$). We only have to enrich the SomeWhere platform with the functionalities of DeCAbL .

Completeness of DeCAbL

Theorem 6 states that DeCAbL$(q, SP, hist, lb)$ is complete, in the sense that every proper consequence m such that newLits$(m, hist) \leq lb$ is returned. The proof of this theorem is based on the completeness of DeCA . As we know that DeCA is complete, we prove that all consequences of DeCA$(q, SP, hist)$ that satisfy the condition newLits$(m, hist) \leq lb$ are returned by DeCAbL$(q, SP, hist, lb)$. Note that when the parameter *hist* is $\emptyset$, the condition newLits$(m, hist) \leq lb$ is equal to $\mid m \mid \leq lb$, where $\mid m \mid$ represents the length of m. In practice, when we ask a query to a P2PIS, the initial value of the *hist* parameter is $\emptyset$.

Theorem 6 *If* $m \in$ DeCA$(q, SP, hist_0)$ *and* newLits$(m, hist_0) \leq lb_0$ *then* $m \in$ DeCAbL$(q, SP, hist_0, lb_0)$

Algoritm 7: Decentralized bounded length consequence finding algorithm
DeCAbL$(q, SP, hist, lb)$

(1) **if** there exists $P \in SP$ s.t $q \in P$ or if for every $P \in SP, (q, P, \_) \in hist$ **return** $\emptyset$
(2) **else if** $(\overline{q}, \_, \_) \in hist$ **return**$\{\Box\}$
(3) **else** for every $P \in SP$, LOCAL$(P) \leftarrow \{q\} \cup (\cup_{P \in SP} Resolvent(q, P))$
(4) **if** there exists $P \in SP$ s.t. $\Box \in$ LOCAL(P) **return** $\{\Box\}$
(5) **else** for every $P \in SP$, LOCAL$(P) \leftarrow \{\, c \in$ LOCAL$(P) \mid L(c) \in \mathcal{T}arget(SP)$ and NEWLITS$(L(c), hist) \leq lb \,\}$
(6) **if** for every $P \in SP$ and $c \in$ LOCAL(P), $S(c) = \Box$, **return** $\bigcup_{P \in SP}$LOCAL(P)
(7) **else**
(8) LOCAL $\leftarrow \bigcup_{P \in SP}$LOCAL$(P)$
(9) RESULT $\leftarrow$ LOCAL
(10) **foreach** $P \in SP$ and $c \in$ LOCAL(P) s.t $S(c) \neq \Box$
(11) **if** $\neg q \vee c \in P$ then $P \leftarrow P \setminus \{\neg q \vee c\}$
(12) $nlb = lb$ - NEWLITS$(L(c), hist)$
(13) **foreach** literal $l \in S(c)$
(14) ANSWER(l) $\leftarrow$ DeCAbL$(l,$ ACQ$(l, P), [(q, P, c)|hist], nlb)$
(15) DISJCOMB $\leftarrow (\otimes_{l \in S(c)}$ ANSWER$(l)) \otimes \{L(c)\}$
(16) DISJCOMB $\leftarrow \{t \in$ DISJCOMB s.t NEWLITS$(t, hist) \leq lb \,\}$
(17) RESULT $\leftarrow$ RESULT $\cup$ DISJCOMB
(18) **return** RESULT

Proof: The proof proceeds by induction on the maximum number rc of recursive calls of DeCA$(q, SP, hist_0)$ to obtain m. If m is returned by DeCA$(q, SP, hist_0)$ after rc recursive calls, and if NEWLITS$(m, hist_0) \leq lb_0$, then m is returned by DeCAbL$(q, SP, hist_0, lb_0)$ after rc recursive calls.

- $rc = 0$: either one of the conditions of Line (1) Line (2), Line (4) or Line (6) is satisfied in the DeCA algorithm.

  If m is obtained from line (1) or (2) or (4) of DeCA then m is obtained by the same line of DeCAbL because they are the same code from line (1) to line (4) in the two algorithms.

If m is obtained from line (6) of DeCA , it means in line (3) of DeCA there is a peer $P \in SP$ such that $m \in Resolvent(q, P)$. Because the code from line (1) to (4) are the same in the two algorithms, in line (3) of DeCAbL we also have $m \in Resolvent(q, P)$.

m is returned from line (6) of DeCA means that it is not filtered out in line (5), so m satisfies the condition of this line. As we have the same code in both algorithms from line (1) to (4), in line (5) of DeCAbL , m satisfies the same condition as in DeCA . Moreover, as we have newLits$(m, hist_0)$ $\leq lb_0$ and because the length of m is at least equal to the length of $L(m)$, we have newLits$(L(m), hist_0) \leq lb_0$. Therefore, m satisfies both conditions in line (5) of DeCAbL , it is not filtered out in line (5) of DeCAbL.

In line (5) of DeCAbL , for each $P \in SP$, the consequences in local(P) are filtered out using 2 conditions, one of them corresponds to the unique condition in line (5) of DeCA . Therefore, each reduced local(P) in line (5) of DeCAbL has no more clauses than the corresponding reduced local(P) in line (5) of DeCA . So, the condition in line (6) of DeCAbL must be satisfied. Therefore, m is returned by DeCAbL .

- Suppose the induction hypothesis true for $rc \leq p$, and let SP be a set of peers such that DeCA$(q, SP, hist_0)$ requires at most $p + 1$ recursive calls to find m. Since there is at least one recursive call, none of the conditions of Line (1), Line (2), Line(4) of DeCA is satisfied. By the same argument as in the base case, none of the conditions of Line (1), Line (2), Line(4) of DeCAbL is satisfied.

  Moreover, because m is obtained after $p + 1$ recursive calls, it can not be obtained at line (6) of DeCA$(q, SP, hist_0)$. It means m is obtained in line (14), after being formed in line (13). So, the form of m is $L(c) \vee A_1 ... \vee A_k$ where $L(c)$ is the local part of a clause c in $Resolvent(q, P)$, P is a peer in SP when DeCA is called the first time, i.e. DeCA$(q, SP, hist_0)$, and for each i=[1..k], A_i is returned by DeCA $(l_i, ACQ(l_i, P), [(q, P, c)|hist_0])$ after at most p recursive calls, where l_i is a shared literal of the clause c.

In DeCAbL , by using the same argument as in the base case, such a clause c is also obtained in line (3), is not filtered out in line (5), and because its shared part $S(c)$ is not equal to $\Box$, DeCAbL$(q, SP, hist_0, lb_0)$ does not terminate at line (6). This clause c will be split, into the $L(c)$ part and l_i, i=[1..k]. Each l_i is propagated by a recursive call. We will prove that for each i=[1..k], we have A_i as a consequence of l_i by applying the induction hypothesis. Then by the recombination of $L(c)$ and the A_is, we form m.

In order to apply the induction hypothesis for inferring that A_i is also obtained after p recursive calls of DeCAbL , as one of the results of DeCAbL$(l_i, ACQ(l_i, P), [(q, P, c)|hist_0], lb_1)$, where $lb_1 = lb_0$ - newLits$(L(c), hist_0)$, we need to prove that newLits$(Ai, [(q, P, c)|hist_0]) <= lb_1$.

Let $\mathcal{C}$ be the set of clauses of $hist_0$, i.e. for every clause cc, $cc \in \mathcal{C}$ if and only if $[\_\,, \_\,, cc] \in hist_0$.

Let $L^m_{\mathcal{C}}$ be the set of literals of m which do not appear in any clause of $\mathcal{C}$. Let $\mid L^m_{\mathcal{C}} \mid$ denotes the cardinality of $L^m_{\mathcal{C}}$. By definition, $\mid L^m_{\mathcal{C}} \mid =$ newLits$(m, hist_0) \leq lb_0$.

Let $L^{L(c)}_{\mathcal{C}}$ be the set of literals of $L(c)$ which do not appear in any clause of $\mathcal{C}$. Let $\mid L^{L(c)}_{\mathcal{C}} \mid$ denote the cardinality of $L^{L(c)}_{\mathcal{C}}$. By definition, $\mid L^{L(c)}_{\mathcal{C}} \mid =$ newLits$(L(c), hist_0)$.

In each A_i, let $L^{A_i}_{\mathcal{C} \cup L(c)}$ be the set of literals of A_i which do not appear in any clause of $\mathcal{C} \cup L(c)$. Let $\mid L^{A_i}_{\mathcal{C} \cup L(c)} \mid$ denotes the cardinality of $L^{A_i}_{\mathcal{C} \cup L(c)}$.

Because $m = L(c) \vee A_1 ... \vee A_k$, so for each A_i, i = [1..k], $\mid L^m_{\mathcal{C}} \mid \geq \mid L^{A_i}_{\mathcal{C} \cup L(c)} \mid + \mid L^{L(c)}_{\mathcal{C}} \mid$.

Therefore, for each A_i, $\mid L^{A_i}_{\mathcal{C} \cup L(c)} \mid \leq \mid L^m_{\mathcal{C}} \mid - \mid L^{L(c)}_{\mathcal{C}} \mid$.

In other terms, we have $\mid L^{A_i}_{\mathcal{C} \cup L(c)} \mid \leq$ newLits$(m, hist_0)$ - newLits$(L(c), hist_0)$ $\leq lb_0$ - newLits$(L(c), hist_0) = lb_1$ (*).

For each A_i, newLits$(Ai, [(q, P, c)|hist_0])$ is the set of literals that are in A_i but not in $\mathcal{C}$ and not in $L(c)$ and not in $S(c)$. This set is not bigger than the set $L^{A_i}_{\mathcal{C}\cup L(c)}$. We have newLits$(Ai, [(q, P, c)|hist_0]) \leq \mid L^{A_i}_{\mathcal{C}\cup L(c)} \mid$ (**).

From (*) and (**), we have newLits$(Ai, [(q, P, c)|hist_0]) \leq lb_1$.

Therefore, the induction hypothesis applies and we can infer that A_i is obtained after p recursive calls of DeCAbL, as result of DeCAbL$(l_i, ACQ(l_i, P), [(q, P, c)|hist_0], lb_1)$.

Then, m is obtained in Line (15) of DeCAbL and returned in Line (16) since it satisfies the condition newLits$(m, hist_0) \leq lb_0$.

□.

4.3.3 Experimentations with DeCAbL

In this section, we report the results of the experimentations of DeCAbL that we have performed. In those experiments, we have used all the flexibility of the SomeWhere platform to consider networks of peer theories hosted by different SomeWhere platforms, running on different computers. Our main goal is to compare the performance of DeCAbL with that of DeCA. We have designed three data sets, corresponding to three different P2PIS, in which each theory is made of synthetic data.

4.3.3.1 Data set 1: a consistent P2PIS

The P2PIS corresponding to the first data set consists of a chain of 150 peers P_i, $i = [0..149]$ such that every peer has exactly one mapping with its predecessor and with its successor in the chain (if it exists).

All peer theories have the same structure, with one local clause and two mappings, except P_0 who has only one mapping connecting it to P_1, and P_{149} who has only one mapping connecting it to P_{148}. Each peer has 3 local variables, A_i, B_i and C_j, all of which are declared to be in the in the so-called Target language of P_i (see Definition 3, section 2.1.2).

For every $j = [1..148]$, the theory P_j is the following :

$$\boxed{\begin{array}{l} PeerP_j \\ Local\ clauses \\ A_j \vee C_j \\ Mapping\ clauses \\ A_j \vee B_j \vee \neg A_{j+1} \\ A_{j-1} \vee B_{j-1} \vee \neg A_j \end{array}}.$$

The theory P_0 has only one mapping:

$$\boxed{\begin{array}{l} PeerP_0 \\ Local\ clauses \\ A_0 \vee C_0 \\ Mapping\ clauses \\ A_0 \vee B_0 \vee \neg A_1 \end{array}}.$$

The theory P_{149} has only one mapping:

$$\boxed{\begin{array}{l} PeerP_0 \\ Local\ clauses \\ A_{149} \vee C_{149} \\ Mapping\ clauses \\ A_{148} \vee B_{148} \vee \neg A_{149} \end{array}}.$$

We have created this kind of P2PIS because we want to have the following characteristics in our P2PIS:

- Firstly, we know that in this P2PIS, there is no inconsistency.
- Secondly, if we ask the query $\neg A_0$ to the peer P_0, the reasoning will be *linear*. In this case, we will have locally in P_0 two consequences : C_0 and $B_0 \vee \neg A_1$. While the clause C_0 does not contain any shared variable, the other clause has $\neg A_1$ as a shared variable. Therefore, we propagate $\neg A_1$ to P_1 as a sub-query. In P_1, the same results will be produced, and a sub-query $\neg A_2$ will be propagated to P_2, etc. We can assure and verify that the reasoning will finish at the peer P_{149}.
- Thirdly, we recall that in practice, the parameter *hist* is always set to $\emptyset$ for the first query. Then, if we ask the query $\neg A_0$ to the peer P_0 with the parameter *lb* set to a value l, the reasoning just needs to go from P_0 up to

P_{l-1}. It is due to the fact that, in each peer P_i, we can decrease by one the new bound for the sub-query being propagated to the next peer, because for the clause $c = B_i \vee \neg A_{i+1}$, $L(c) = B_i$ always contain a new literal that has not appeared in *hist*.

With this characteristic, we can easily compare how the processing time evolves according to the value of the parameter *lb*, and compare it with the case where no bound is set at all, when the query $\neg A_0$ is asked to the peer P_0,

In these experiment, the 150 peers are hosted by 5 SomeWhere platforms, each of them thus hosting 30 theories and running on a different machine. The figure 4.5 shows the result of this experimentation.

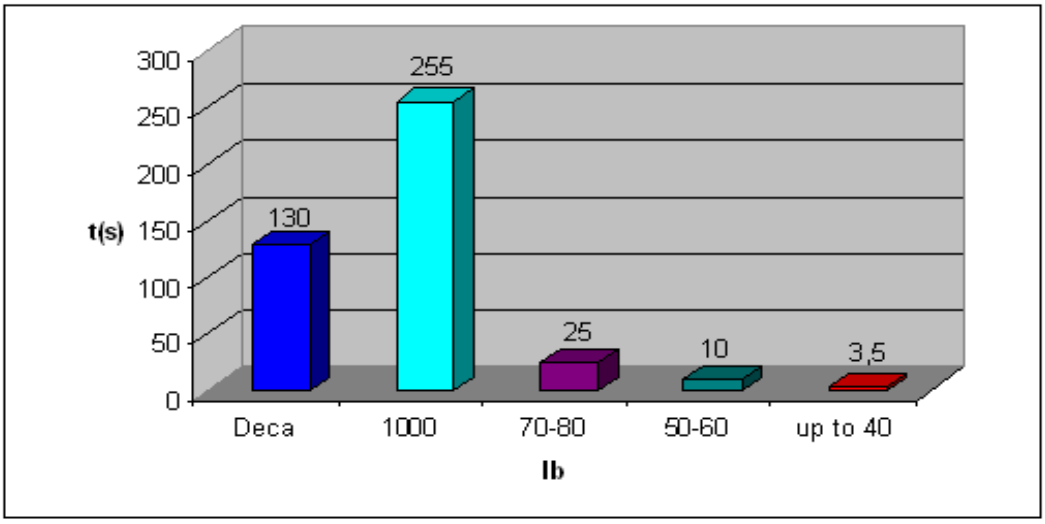

Figure 4.5: Query time comparison between DeCA and DeCAbL with varied parameter *lb*

The first remark from these results is that if we disable the parameter *lb* of DeCAbL by setting for it a very great value (1000 in the figure above), then DeCAbL is much slower than DeCA . This is actually understandable, because DeCAbL must do more things than DeCA , such as the computation of the new value of *lb* for each sub-query, the computation of newLits$(L(c), hist)$ and

newLits$(m, hist)$ that consists of a set inclusion test of each literal of $L(c)$ or of m with respect to $hist$.

The second point observed from the figure 4.5 is that we gain very much in term of time when we set lb to a value which is smaller than the maximum possible length of the consequences. In this P2PIS, the maximum possible length of a consequence is 150, corresponding to the reasoning branch going from P_0 to P_{149}. We can see that even if when the bound is between 70 and 80, which is not very small w.r.t. 150, the query finishes in 25 seconds, more than five time faster than DeCA . Especially, if we only want to verify if the P2PIS is consistent, by setting lb to 0, then we obtain the result after just 0.7 second. Using DeCA the same query would take 130 seconds.

It is worth noticing that when extending such a P2PIS to 200 peers (i.e. 50 more peers in the chain), DeCA takes much longer : 310 seconds to complete (i.e. 180 more seconds). This point confirms that the further the reasoning goes, the longer the query time. The explication is : not only due to the fact that the reasoning propagates farther in the network, but also to the fact that more recombinations occur, when a consequence is sent back to the previous peer in the same branch of reasoning. Fixing a bound for the size of returned of consequents thus not only prevent the propagation of the reasoning in the network but also saves all recombinations that would occur otherwise.

4.3.3.2 Data set 2: an inconsistent P2PIS

The P2PIS generated in this section consist of 100 peers, $\{P_i\}$, $i = [0..99]$. Each peer has 4 local variables A_i, B_i, C_i and D_i, all of them are in the Target language of P_1.

The theory P_0 :

$PeerP_0$
Local clauses
$\neg A_0 \vee D_0$
$\neg D_0 \vee C_0$
$\neg D_0 \vee B_0$
Mapping clauses
$\neg A_0 \vee A_1$
$\neg A_0 \vee \neg A_{99}$

.

For every $j = [1..9]$, the theory P_j is the following :

$PeerP_j$
Local clauses
$\neg A_j \vee D_j$
$\neg D_j \vee C_j$
$\neg D_j \vee B_j$
Mapping clauses
$\neg A_j \vee A_{j+1}$
$\neg A_{j-1} \vee A_j$

.

The theory P_{10} :

$PeerP_{10}$
Local clauses
$\neg A_{10} \vee D_{10}$
$\neg D_{10} \vee C_{10}$
Mapping clauses
$\neg D_{10} \vee B_{10} \vee D_{11}$
$\neg A_{10} \vee A_{11}$
$\neg A_9 \vee A_{10}$

.

For every $j = [11..98]$, the theory P_j is the following :

$PeerP_{10}$
Local clauses
$\neg D_j \vee C_j$
Mapping clauses
$\neg D_j \vee B_j \vee D_{j+1}$
$\neg A_j \vee A_{j+1}$
$\neg D_{j-1} \vee B_{j-1} \vee D_j$
$\neg A_{j-1} \vee A_j$

.

The theory P_{99} :

$PeerP_{99}$
Local clauses
$\neg D_{99} \vee C_{99}$
Mapping clauses
$\neg D_{98} \vee B_{98} \vee D_{99}$
$\neg A_{98} \vee A_{99}$
$\neg A_0 \vee \neg A_{99}$

.

This P2PIS has the following characteristics:

- It is inconsistent if we add the new clause A_0 to P_0. The inconsistency is caused by the set of mappings of length 2. The idea is that if we ask the query A_0 to P_0, then one of the consequence inside P_0 is A_1. A_1 is shared with P_1 and thus is asked to P_1. In P_1, the same result is obtained and the reasoning forwards to P_2 and so on, up to P_{99} with the sub-query A_{98}. In the peer P_{99}, we find A_{99} as a consequence of A_{98}. Then, from A_{99}, using the mapping $\neg A_0 \vee \neg A_{99}$, we find $\neg A_0$. Because the initial query was A_0, the P2PIS is inconsistent.

- If we ask the query A_0 to the peer P_0, there is only one branch of reasoning that goes from P_0 to P_{10}, with a sub-query A_{10}. Then, from P_{10}, we have two branches of reasoning.

  The first one goes to P_{11} with the sub-query D_{11}, and so on, up to P_{99}. This branch of reasoning uses mappings of length 3 at each peer it passes. Note that this set of mappings has the same characteristics as those of the P2PIS in the previous experimentation, i.e. it is consistent and linear.

  The second branch of reasoning goes from P_{10} to P_{11} with the sub-query A_{11}. This branch corresponds to the one that we have just discussed above, i.e. it uses the set of mappings of length 2 and it is inconsistent.

As in the previous experimentation, the 100 theories are handled by 5 SomeWhere platforms, each SomeWhere hosts 20 theories. The 5 SomeWhere platforms run on 5 different machines.

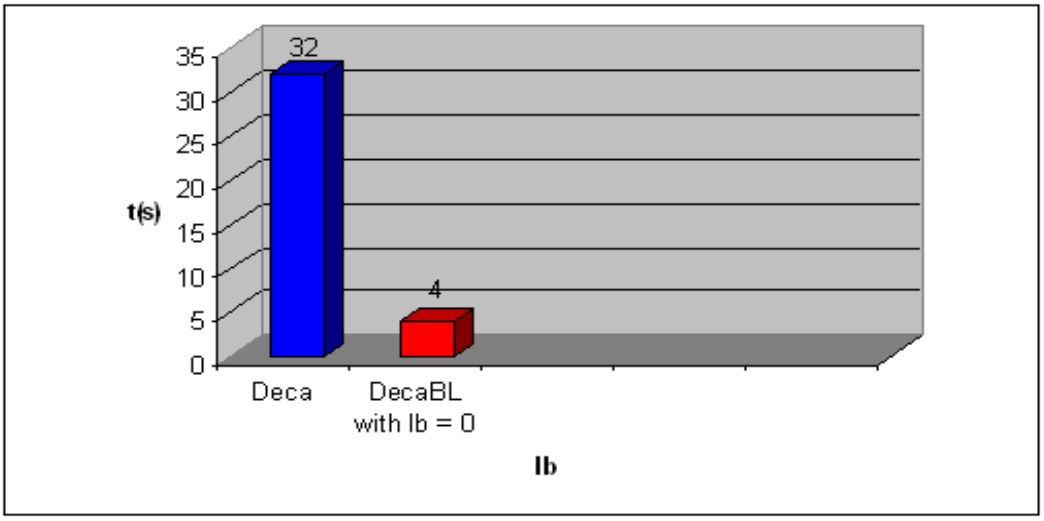

Figure 4.6: Query time comparison between DeCA and DeCAbL with parameter $lb = 0$ for a test of inconsistency

Again our goal is to compare the processing time of DeCA and DeCAbL. But since we know here that adding A_0 is inconsistent with the global theory, the only interesting consequent is in fact the empty clause. Therefore we only compare DeCA to DeCAbL with $lb = 0$.
The result presented on figure 4.6 consolidates the observations made in the previous experimentation. With DeCA, it takes 32 seconds to complete the query processing while with DeCAbL the query processing terminates after just 4 seconds. This gain can also be explained by the fact that when we only look for $\square$, the linear reasoning starting from P_{10} but not leading to $\square$ is not initiated by DeCAbL. It is because at P_{10}, when the local reasoner produces $B_{10} \vee D_{11}$ as a consequence of A_{10}, the condition of $\text{NEWLITS}(\{B_{10}\}, hist_{10}) \leq lb = 0$ is not satisfied, therefore the sub-query D_{11} is not propagated to P_{11}, avoiding this branch of reasoning. In contrast, this has to be executed with DeCA, accounting for the long reasoning time of the inconsistency test.

4.3.3.3 Data set 3: a random P2PIS

For the third data set, we have generated a P2PIS with the following characteristics (corresponding to the class *hard* in (13)) :

- the network is made of 1000 randomly generated peers, that are randomly linked each other, but in such a way that the graph of connected peers has the "small world" topotogy characterizing many social networks.
- The theory of peer is randomly generated on a vocabulary of 70 local variables, among which 40 are declared in its Target language. Each peer has 40 local clauses of length 2, concerning only its local variables.
- Each peer is connected to 10 different neighbors, and shares 3 of its local variables with each of its neighbor. All the mappings between the peers are of length 3.
- Because the P2PIS is generated in a random manner, we can not know in advance if it is consistent or not.

We have deployed this P2PIS on a cluster of 22 machines, each machine running one SomeWhere platform hosting about 48 peers. Our goal with this experiment is to compare the performance of DeCAbL and DeCA with respect to an inconsistency test. For this, we have generated 1000 different unit clause queries. For each query, we have compared DeCA, and DeCAbL with the parameter lb set to 0. Since on such networks, some queries may take a long time to terminate, we also have set up a time-out value of 45 seconds for every query: a query that takes more than 45 seconds to finish is called a time-out query. The figure 4.7 shows the result of this experimentation.

As we can see in Figure 4.7, when using DeCA, about 34% of the 1000 queries terminate before the time-out. This number augments to 40% when using DeCAbL . This shows that even in a randomly generated P2PIS, we gain in term of query time with DeCAbL . However, the gain in this case is less important than in the two previous experimentations. The explication is that we can not assure a query with DeCAbL to have less reasoning branches, and each reasoning branch to stop sooner than with DeCA .

Let us consider a peer P_i with two mappings of length 3. The first mapping concerns two local variables and one shared variable, $m_1 = A_i \vee B_i \vee C_j$. The second mapping concerns one local variable and two shared variable, $m_2 = A_i \vee$

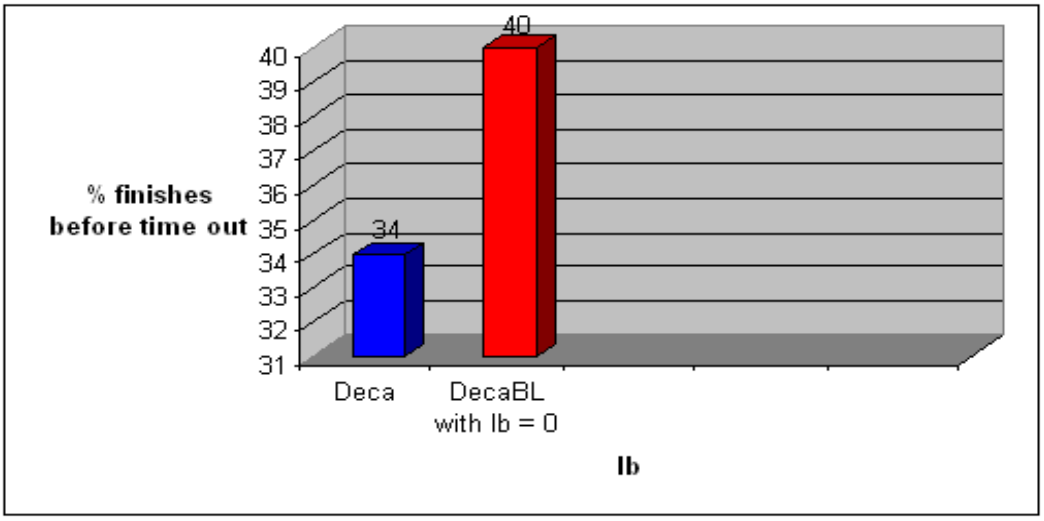

Figure 4.7: Comparison between DeCA and DeCAbL with parameter $lb = 0$ on the percentage of queries that finish before time out in a random P2PIS

$B_j \vee C_j$. Suppose that the query $\neg A_i$ is asked to P_i. Two local consequences are produced : $c_1 = B_i \vee C_j$ and $c_2 = B_j \vee C_j$. If newLits$(L(c_1), hist) > lb = 0$, the shared variable C_j is not asked to P_j. In contrast, newLits$(L(c_2), hist) = 0$, therefore both B_j and C_j are asked to P_j, initializing two branches of reasoning. In a similar way, each of them may also initialize two sub-branches in P_j, and so on. A sub-branch of reasoning can (possibly) stop only if it is solved with a mapping of the form of m_1, i.e. two local variables and one shared variable. We generated and carried on the two previous experimentations in such a way that this condition was guaranteed. However, in this experimentation, because we do not control the generation of the P2PIS, this point is not ensured.

An other observation from this experimentation is that in both the set of non time-out queries, a majority of them finishes very quickly, less then 2 seconds. A small part of them, corresponding to a portion of non time-out queries when using DeCAbL , finishes after 12 seconds. This point shows that the non time-out queries stop quickly because either they do not produce shared variables that need to reason further in other peers, or they are solved fairly quickly with mappings of the form of m_1, as discussed above.

The experimentations that we have presented in this section show that DeCAbL is a good optimization of DeCA in term of query time.

4.3.4 Implementation of P2P-NG and WF-DeCA

We now focus on the adaptations that have been performed on the SomeWhere platform in order to implement P2P-NG and WF-DeCA.

As mentioned earlier, DeCAbL can be closely related to P2P-NG since in the particular case where the bound lb is fixed to 0, both algorithms only focus on possible derivations of the empty clause. The main difference between the two algorithms is that while DeCAbL will output either an empty set or a set containing only the empty clause, P2P-NG ouputs either an empty set or a set containing the mapping supports of all possible derivations of the empty clause. This is essentially the point where the two algorithm differ. While DeCAbL uses *Resolvent* to perform its local reasoning, P2P-NG uses $Resolvent + SMS$. The latter computes local non only local consequents but also their minimal sets of support. Also, when recombining the results obtained from recursive calls, sets of mapping support obtained so far and coming from the different recursive calls have to be distributed over each other, as well as over the mapping support of the local part of the consequent.

Note that in practice, despite the fact that the management of mapping supports necessarily induces a little overhead (with respect to DeCAbL), because similar pruning can be achieved as for the case of DeCAbL when $lb = 0$, the observed performances of P2P-NG remain very close to those of DeCAbL.

In order to use the P2P-NG algorithm in an appropriate way, the SomeWhere platform has been enriched with additional functionality. Particularly, any peer hosted by a platform has been given the possibility to add new mappings with existing peers of a P2PIS. This was not possible in the original version of SomeWhere, where each peer could only be launched with a static theory and set of mappings. This has required to reconsider a number of implementation details. For instance, adding a new mapping may cause a variable of distant peer to change its status from non-shared to shared variable. The communication

protocol between peers has thus been enriched with new messages, in order to be able to warn distant peers involved in a mapping so that they can register such changes of status.

Peers have been enriched as well with a local nogood database. When adding a new mapping m to the peer P and when the result of P2P-NG(m, P) is not empty, a new nogood $ms \cup \{m\}$ is inserted in the database of P, for each mapping support ms that is returned by P2P-NG. The global launch process of a P2PIS has also been revised in such a way that each peer, when launched, successively adds its mapping using the previously described insertion procedure.

The implementation of WF-DeCA can be seen as a variation on the implementation of both DeCAbL and P2P-NG. From the former, we have kept the idea of bounding the size of produced implicates, since it still has its own interest to focus on the production of such consequents. From a theoretical point of view, since the number of literals of the global theory of the P2PIS is finite, setting a very large bound still allows for the production of all implicates of a query. Of course, we have learned that controlling the size of the produced implicate has also some overhead and, if we really want to produce all implicates, it could be significant. But from a practical point of view it is much more convenient to have this possibility. Anyway, it would be very easy to adapt the current version of WF-DeCA to get rid of this control on the size of produced implicates.

From P2P-NG, WF-DeCA keeps the fact that when deriving a consequent all mapping supports have to be produced as well. But in addition, it also has to retrieve, from the local nogood databases, the nogood that possibly overlap with existing mapping support. From a technical point of view, content of answer messages has been extended in order to incorporate sets of mapping supports as well as sets of nogoods. Then the recombination operator has been adapted in order to work on triples of the form $(c\ sms\ sng)$.

4.4 Conclusion and discussion

In this chapter, we have considered the problem of **dealing with inconsistencies in** SomeWhere . We have proposed two algorithms: P2P-NG to detect and store causes of inconsistencies, WF-DeCA to reason and return only well-founded answers to queries. We have proved the soundness and termination of both algorithms, as well as the completeness of P2P-NG, which is key for the soundness of WF-DeCA.

We have also described how we have implemented P2P-NG and WF-DeCA in the SomeWhere platform. Finally, we have described an adaptation of DeCA (DeCAbL) that produces only consequences whose length is bound by an input parameter. We have explained how the promising results on the performance gain of DeCAbL can be transposed to P2P-NG and WF-DeCA.

As stated earlier, our approach is in somehow similar to (64) and (40). Briefly, we also use a new notion of logical consequences that is based on the classical one, i.e. a consequence of q and Σ is **well-founded** if it can be classically deduced from q and a consistent subset Σ' of Σ. We detail here differences between our approaches.

In (64) and (40), the inconsistent set Σ is known, it is a centralized knowledge base. Therefore, it is not difficult to compute consistent subsets Σ' of Σ as detailed in Section 3.2.2.2. In contrast, the inconsistent set Σ is not known in our framework. Σ is the union of all the peer theories which are decentralized over a SomeWhere network. There is no centralized control and no knowledge of the whole theory. Therefore, we can not use the same procedures as in (64) and (40) to find Σ'. To solve the problem, our approach is based on the decentralized computation of the nogoods, which are exploited to check whether subset of Σ supporting the derivation of a consequence are consistent.

It is also important to notice that the storage of nogoods is completely distributed. We cannot know all the nogoods stored in our network. However, while processing a query, we must be sure of collecting all the nogoods that would invalidate the production of a consequence. Theorem 5 makes sure of this for WF-DeCA.

In our approach, for every query, each consequence is well-founded w.r.t. a possibly different consistent subset Σ'. In (64) and (40), once found, Σ' is fixed and all the consequences are well-founded w.r.t. this Σ'. With respect to a modification on the whole theory Σ (a new mapping is added between two peers in SomeWhere, or a new axiom is added to the centralized knowledge base in (64) and (40)), this point has some impacts. In our context, we must detect if the new mapping creates new nogoods and store them while in (64) and (40), we must re-verify the consistency of the union of the found and fixed Σ' with the new axiom.

An other difference between our approach and that of (64) and (40) is that given a query q, we accept to obtain both A and its negation $\neg A$ as long as each of them can be deduced from a consistent subset Σ' of Σ and q. This is due to the fact that the definition of our notion of logical consequences corresponds only to the **soundness** property, but not to the **meaningfulness** property as defined in (64) and (40)

Compared to DeCA, the main property of WF-DeCA is produce well-founded answers **only**. Figure 4.8 shows a case where DeCA produces an answer derived from an inconsistent P2PIS. That answer, which is not well-founded, is not produced by WF-DeCA when applied to the same P2PIS.

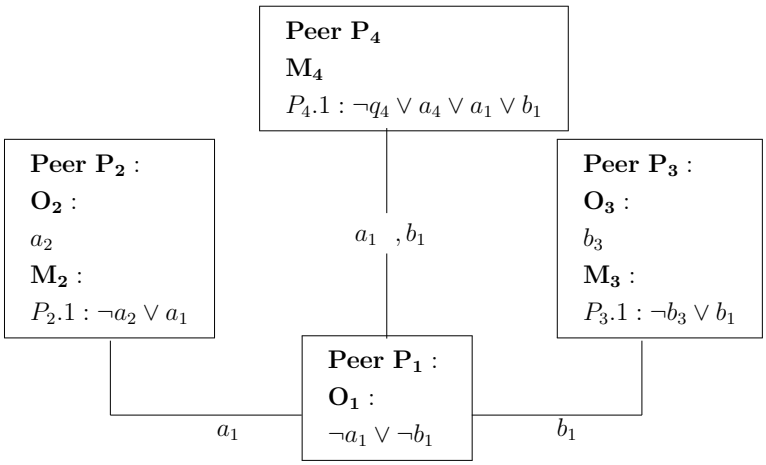

Figure 4.8: Example of an inconsistent P2PIS (edges labeled by shared variables)

The simple P2PIS illustrated in the figure 4.8 is inconsistent due to the set of clauses $\{\neg a_1 \vee \neg b_1, \neg a_2 \vee a_1, a_2, \neg b_3 \vee b_1, b_3\}$. Among these clauses, there are the mappings $P_2.1$ and $P_3.1$. Suppose that mappings are added by the peers with the order : $P_2.1, P_3.1, P_4.1$. It can be verified that the addition of $P_2.1$ does not cause any inconsistency. When $P_3.1$ is added into P_3, a nogood $ng = \{P_2.1, P_3.1\}$ is detected and stored at P_3. The addition of $P_4.1$ does not cause any new nogood.

Illustration of WF-DeCA on a query

Let us consider the query q_4 that is provided to P_4. A local consequence of the query is $a_4 \vee a_1 \vee b_1$ with the mapping support $P_4.1$. Because a_1 and b_1 are shared with P_1, each of them is sent to P_1, initializing two reasoning branches which are processed in parallel.

- When a_1 is propagated to P_1, by resolution with $\neg a_1 \vee \neg b_1$, a local consequence is $\neg b_1$, which is shared with P_3 and therefore propagated to P_3. Inside P_3, the empty clause $\square$ is deduced with the mapping support $P_3.1$ after resolution with b_3. The nogood ng stored at P_3 is also collected because $P_3.1$ is included in ng. $(\square \ \{\{P_3.1\}\} \ \{ng\})$ is thus sent back to P_1 as a consequence of $\neg b_1$, then to P_4 as a consequence of a_1 to be recombined with a_4 and each of the consequences of b_1.

- When b_1 is propagated to P_1, by resolution with $\neg a_1 \vee \neg b_1$, a local consequence is $\neg a_1$, which is shared with P_2 and therefore propagated to P_2. Inside P_2, the empty clause $\square$ is deduced with the mapping support $P_2.1$, after resolution with a_2. $(\square \ \{\{P_2.1\}\} \ \emptyset)$ is thus sent back to P_1 as a consequence of $\neg a_1$, then to P_4 as a consequence of b_1 to be recombined with a_4 and each of the consequences of a_1.

By recombining $(a_4 \ \{\{P_4.1\}\} \ \emptyset)$ with $(\square \ \{\{P_3.1\}\} \ \{ng\})$ and $(\square \ \{\{P_2.1\}\} \ \emptyset)$, we obtain $(a_4 \ \{\{P_4.1, P_3.1, P_2.1\}\} \ \{ng\})$. The only mapping support of the consequence a_4 contains the nogood ng (which is $\{P_2.1, P_3.1\}$). Therefore, $(a_4 \ \{\{P_4.1, P_3.1, P_2.1\}\} \ \{ng\})$ is discarded by WF-DeCA. Actually, a_4 has been deduced using all the clauses identified above that are responsible for the inconsistency, and thus is not well-founded. Note that the DeCA algorithm,

since it does not manage mapping supports as well as nogoods, is not able to detect that this consequent is not well-founded.

As stated by Theorem 5 and illustrated in the previous example, WF-DeCA guarantees to produce well-founded answers only. It is not however guaranteed to be complete. The following example shows a case in which WF-DeCA does not produce an answer which is yet well-founded.

Let us consider the P2PIS illustrated in the figure 4.9. It is inconsistent due to the set of clauses $\{\neg a_1 \vee \neg b_1, \neg a_2 \vee a_1, a_2, \neg b_3 \vee b_1, b_3\}$. Among these clauses, there are the mappings $P_1.1$ and $P_3.1$. Suppose that mappings are added by the peers with the order : $P_2.1, P_1.1, P_1.2, P_3.1, P_4.1$. It can be verified that the additions of $P_2.1$, $P_1.1$, $P_1.2$ do not cause any inconsistency. When $P_3.1$ is added into P_3, a nogood $ng = \{P_1.1, P_3.1\}$ is detected and stored at P_3. The addition of $P_4.1$ does not cause any new nogood.

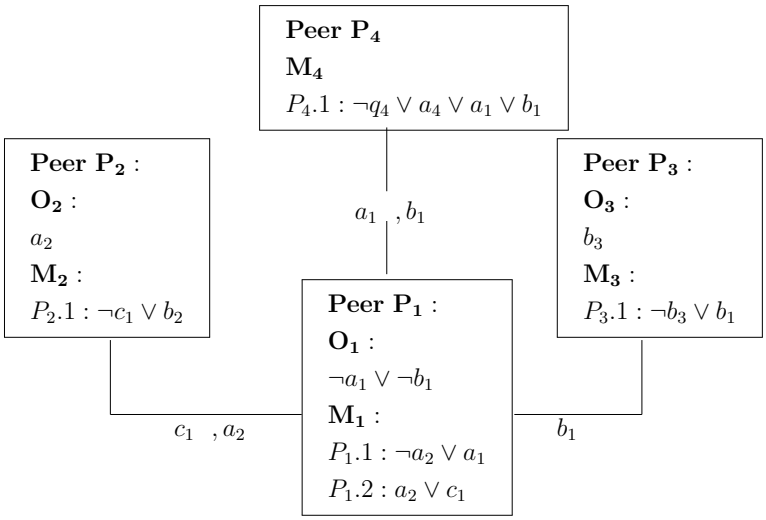

Figure 4.9: Example of an inconsistent P2PIS (edges labeled by shared variables)

For the non-completeness of WF-DeCA, let us consider the query q_4 that is provided to P_4. A local consequence of the query is $a_4 \vee a_1 \vee b_1$ with the mapping support $P_4.1$. Because a_1 and b_1 are shared with P_1, each of them is sent to P_1, initializing two reasoning branches which are processed in parallel.

- When a_1 is propagated to P_1, by resolution with $\neg a_1 \vee \neg b_1$, a local consequence is $\neg b_1$, which is shared with P_3 and therefore propagated to P_3. Inside P_3, the empty clause $\square$ is deduced with the mapping support $P_3.1$ after resolution with b_3. The nogood ng stored at P_3 is also collected because $P_3.1$ is included in ng. $(\square \ \{\{P_3.1\}\} \ \{ng\})$ is thus sent back to P_1 as a consequence of $\neg b_1$, then to P_4 as a consequence of a_1 to be recombined with a_4 and each of the consequences of b_1.

- When b_1 is propagated to P_1, by resolution with $\neg a_1 \vee \neg b_1$, a local consequence is $\neg a_1$. By resolution $\neg a_1$ with the mapping $P_1.1$ we produce $\neg a_2$, which produces c_1 by resolution with the mapping $P_1.2$. c_1 is shared with P_2 and therefore propagated to P_2 and produces b_2 using the mapping $P_2.1$. $(b_2 \ \{\{P_2.1\}\} \ \emptyset)$ is thus returned to P_1 as a consequence of c_1. Because c_1 is a consequence of $\neg a_2$ which is itself a consequence of $\neg a_1$, b_2 is a consequence of $\neg a_1$ with a mapping support equal to $\{\{P_2.1, P_1.1, P_1.2\}\}$. Finally, because $\neg a_1$ is produced from b_1 coming from P_4, $(b_2 \ \{\{P_2.1, P_1.1, P_1.2\}\} \ \emptyset)$ is thus sent back to P_1 as a consequence of b_1 to be recombined with a_4 and each of the consequences of a_1.

By recombining $(a_4 \ \{\{P_4.1\}\} \ \emptyset)$ with $(\square \ \{\{P_3.1\}\} \ \{ng\})$ and $(b_2 \ \{\{P_2.1, P_1.1, P_1.2\}\} \ \emptyset)$, we obtain $(a_4 \vee b_2 \ \{\{P_4.1, P_3.1, P_2.1, P_1.1, P_1.2\}\} \ \{ng\})$. The only mapping support of the consequence $a_4 \vee b_2$ contains the nogood ng (which is $\{P_1.1, P_3.1\}$). Therefore, $(a_4 \vee b_2 \ \{\{P_4.1, P_3.1, P_2.1, P_1.1, P_1.2\}\} \ \{ng\})$ is discarded by WF-DECA. However, for the production of this clause, we has not used the clause a_2, which is one of the clauses identified above as accounting for the inconsistency. Therefore, $a_4 \vee b_2$ can be derived from a consistent set of clauses and thus is well-founded according to Definition 16.

Part II

TRUST FOR SEMANTIC PEER-TO-PEER SYSTEMS

INTRODUCTION

Trust is an important issue in peer-to-peer and social networks, in which some (possibly malicious) peers can provide wrong, virus-contaminated or unsatisfying data as answers to a query.

In this second part of our thesis, we explain the model of trust that we propose for semantic peer-to-peer systems. This model is based on the Bayesian approach to statistics and relied on a strong mathematical setting with a clear semantic. Our approach provides a useful framework for ranking answers of a query according to their trust level. The benefit may be high when the number of answers is important. Our approach distinguishes from the existing works by the fact that the trust is context-dependent since, for a given peer, it may vary depending on that peer's classes.

This part is structured as follows :

- In Chapter 5, we present a state of the art on the approaches for trust and reputation computation for peer-to-peer and social networks,
- In Chapter 6, we present our probabilistic trust model for semantic peer-to-peer systems.

  We formally define *trust*, provide a simple theorem to estimate trust and a Bayesian statistics-based proof of this theorem. Our model considers direct experiences of the peer estimating trust towards the classes involved in the semantic annotations of the resources returned as answers to previous queries. In case it does not have direct experiences relevant to the classes for which trust is estimated, we present a simple strategy for it to collect experiences of other peers that are relevant to compute the trust in question.

  We also present how this model of trust can be applied into the SomeOWL semantic peer-to-peer system (presented in Section 2.3). We design a greedy

strategy for SomeOWL that can collect experiences of other peers into the trust computing peer as soon as these information are needed.

Finally, we conclude this part by discussing about our model. Although our model has not been designed to cope with the problem concerning malicious peers, we also discuss how it can deal with this problem.

Chapter 5

STATE OF THE ART ON TRUST AND REPUTATION COMPUTATION MODELS FOR PEER-TO-PEER AND SOCIAL NETWORKS

Online marketplaces like Ebay and Amazon, peer-to-peer file sharing systems such as KaZaa or BitTorrent, or social network services such as Facebook or MySpace have been changing radically the way people interact and exchange information. Actually, e-commerce applications have made buying and selling easier : more potential buyers for selling ; more choices and more competitive prices for buying. Distributed file sharing systems have made information searching and retrieving simple than ever: just say what you are looking for and click the 'submit' button to receive answers. They are very nice and useful applications. However, the question of their trustworthiness is a major issue.

In this chapter we will present representative works that have proposed trust and reputation computation models for online social networks, peer-to-peer networks and information searching and retrieving technique such as Google. We divide the works into four categories. For a better understanding of our classification,

we first distinguish **trust** from **reputation**.

In the Longman dictionary, **trust** is defined as a **strong belief** in the honesty, goodness of someone or something, while **reputation** is defined as the **opinion** that people have about someone or something because of what has happened in the past. In our point of view, the two definitions have a strict relation. Actually, the 'strong belief' in the above definition of trust may be a belief of an individual A in the target person or object. In this case we say that the target person or object is trustworthy for A. This 'strong belief' may also be the belief of a set of individuals $\mathcal{A}$, in this case we say that the target person or object is trustworthy for the whole community $\mathcal{A}$, or that he has a good reputation in $\mathcal{A}$, because the individuals in $\mathcal{A}$ have a good opinion about him.

It makes sense to talk about reputation in contexts where people have the same interest, when there is a consensus about the honesty, goodness of a person. For example, in an online marketplace, buying and selling safely is one of the same interests of the majority of participants. People have thus a consensus of what is a honest user : one that pays correctly when buying, and provides correctly goods when selling. There is also a consensus of what is a cheater : one that does not pay correctly when buying, or that does not provide correctly goods when selling.

It makes sense to talk about trust in contexts where people may not have the same interest, when there may not be a consensus about the honesty, goodness of a person, but where the point of view of each individual about the honesty or goodness of that person counts. For example, there is not an absolute consensus about the quality of laptops of different marks. Peter trust the Asus brand because Asus laptops have a good autonomy although their design is not beautiful. In contrast, his sister has a different point of view, she likes beautiful laptops, so she trusts the Apple brand.

Now that the possible ambiguity in the sense of the terms trust and reputation has been explicated, we present our classification for the works that are being presented. All these works use (satisfaction or complaint) feedbacks on past interactions between agents, users or sources, to compute trust and reputation. Firstly, we distinguish works that proposed a computation model for reputation

from those that proposed a computation model for trust. After that, in each category, we classify a work as 'probabilistic' or 'non-probabilistic' based on the manner that it uses to aggregate feedbacks.

5.1 Models of reputation

In this section, we will present some representative computation models of reputation. In these models, the points of view of individuals knowing the target individual (whose reputation is to be computed) are aggregated. The final outputs express the viewpoint of the whole community towards the target individual.

5.1.1 Non probabilistic models of reputation

In online marketplaces

Reputation systems have been used intensively in online marketplaces such as Ebay (3), the largest online auctioning system, or Amazon (2), the largest online market. In these systems, one can buy and sell many things. In general, a buyer and a seller do not know each other, it is thus questionable for each of them to decide to interact with the other. In order to promote good behaviors of participants, these systems provide very simple methods to compute the reputation of sellers, in Amazon, and of both sellers and bidders, in Ebay. After each transaction, in Amazon, the seller files a feedback about the buyer. In Ebay, both the seller and the bidder make a feedback about each other. A feedback may be positive, negative or neutral. Feedbacks concerning a same user are centralized into a server.

In Amazon, the reputation system computes the reputation of a seller and shows this score, which is the percentage of positive, neutral and negative feedbacks.
In Ebay, the reputation system computes 2 notes for both sellers and bidders. For every user, the first note is the difference between the number of his total positive feedbacks and the number of his total negative feedbacks. The second note is the percentage of his total positive feedbacks w.r.t. his total feedbacks.

In both systems, the reputation of a user may be based on the whole life-time feedbacks concerning this user, or only based on those gathered during the last 3 or 6 months. This is an important point because we can see if there is a change in the behavior of a user. The reason is that a user can build his reputations in a long time by behaving correctly in all transactions concerning small value articles. Then, when having a good reputation, he may begin cheating in transactions concerning valuable articles. With these characteristics, the reputation systems of both Ebay and Amazon are very successful. These online marketplaces have a very big community of participants.

In peer-to-peer setting

While the methods used in Ebay and Amazon rely on a central server, (10) is one of the first approaches for computing reputation of peers in a peer-to-peer setting, such as the P-Grid peer-to-peer network (9) (presented in Chapter 1). In this setting, when two peers P and Q interact to exchange information, each of them may file a feedback about the other. The only type of feedback is complaint (a negative feedback). Each feedback is duplicated. All the feedbacks and their replicas are stored in a distributed manner, using a distributed hash table in the overlay network of P-Grid.

One possible situation in this setting is that, after a transaction, not only the honest peer P can make a complaint about the cheating peer Q, but Q can also make a complaint about P in order to hide his bad behavior. So, a third person R cannot know which one between these two peers is the cheater. However, when Q continues his misbehavior with other peers, the situation becomes easier for R to understand. It will see that Q complains about many peers and all of them complain about Q. It is thus very probable that Q is the cheater. This idea is expressed by the following formula:

$$rep(P) = \mid c(P,Q) \mid \text{x} \mid c(Q,P) \mid$$

where $\mid c(P,Q) \mid$ is the total number of complaints made by P about all the other peers Q ; $\mid c(Q,P) \mid$ is the total number of complaints made by all the other peers Q towards P. High values of $rep(P)$ mean that P is not trustworthy.

When a peer R assesses the reputation of a peer P, using the above formula, it collects all the complaints concerning P by searching the distributed hash table in the overlay network of P-Grid. No central server is thus necessary for this task.

In this setting, an other possible problem is that complaints used for computing the reputation of a peer may be misreported by peers that store them. This problem is dealt with using the replicas that have been stored for each original complaint. Complaints and replicas that are retrieved from a peer, and that differ much from those retrieved from the other peers, will be ignored.

In our viewpoint, the formula used to compute reputation of peers in this model has some flaws. Firstly, the threshold defining that $rep(P)$ is high must be set in an ad-hoc way. We do not know the value at which $rep(P)$ should be considered as 'high' in order to avoid to interact with P. Secondly, the interpretation of $rep(P)$ when it is equal to 0 is not clear. In this case, we do not know if it means that P has never interacted with other peers , or that it has interacted with other peers, but no complaint has been made against it so far.

5.1.2 Probabilistic models of reputation

We present in this section some models of reputation based on probabilistic approaches. We are aware of three main kinds of techniques: those that use the PageRank algorithm, those that use EigenTrust-based algorithms and those that are based on maximum likelihood estimation technique.

5.1.2.1 The PageRank algorithm

The PageRank algorithm has been presented first by (21; 54). It is a probabilistic algorithm used to compute the global reputation of web pages on the Internet. It is also applied to rank the results of document search in Google (6).

In this approach, the reputation of a web page is considered as its rank. It is supposed that a page has a high rank if it is referred by other pages that themselves are highly ranked. The idea is that the more a page is referred, the better its reputation. However, a page that is referred by many pages whose ranks are

not high may not be a good page. On the other hand, a page that has few references to it may still be a very good page, if the pages referring to it are highly ranked. For example, a page that has only one reference to it, the one from a very important page such as www.yahoo.com is still considered more important than an other page which has many references to it but all from unknown pages. In other terms, the rank of a page is divided evenly among its forward links to contribute to the ranks of the pages it points to. Figure 5.1 illustrates this idea.

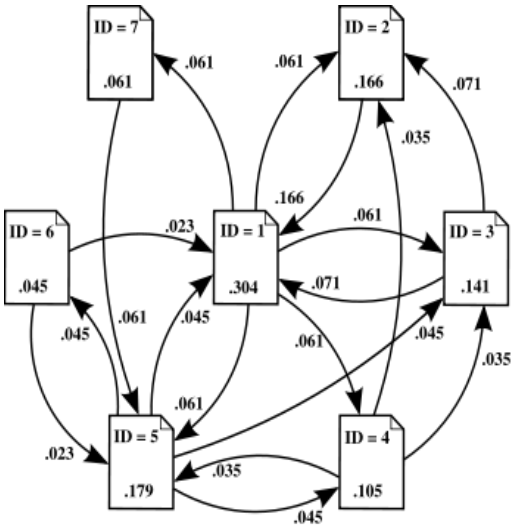

Figure 5.1: How PageRank works

The rank $rank(p)$ of a page p is defined by the following formula, expressing the idea presented above:

$$rank(p) = d.\frac{1}{T} + (1-d)\sum_{i=1}^{k}\frac{rank(p_i)}{C(p_i)}$$

where $d \in [0,1]$, T is the total number of pages, $p_1, ...p_k$ are the pages referring to p, each has rank $rank(p_i)$ and $C(p_i)$ is the number of links out of p_i.

This equation is recursive but it may be computed by starting with any set of ranks and iterating the computation until it converges. In order to know which

page points to which pages, Google has developed a Web page crawler. The pages that are crawled by the web crawler are stored in a single database inside a server. This database is very big, containing more than 24 million pages by the time of 1998. Then, the pages in this database are indexed using an efficient indexing method. When the page indexing step finishes, the computation of PageRank is executed inside the server, using all the informations about pages that have been collected. This computation is thus done in a non-distributed manner.

Given a page p, the probabilistic interpretation of $rank(p)$ is the following. Let us consider a Web surfer who wandering the Web. Suppose that he is on the page p'. At each step, the surfer may jump to one of the page referred by p' with probability $(1 - d)$ or may either jump to any random page on the Web with probability d. The probability that the random surfer visits a page p from p' is thus equal to its rank $rank(p)$.

The algorithm of PageRank has also been used in (53) to compute reputation of users in the Semantic Web, such as the Advogato Web community (1). In (53), more than 3 millions profiles of users of this community and the local trust assertions towards their neighbors have been centralized. From this huge data, subsets of about 572 users, each with in average 5 neighbors, have been used to test the convergence rate of the algorithm. It has been shown that the algorithm converges fairly fast, after about 40 iterations.

In the next sub-section we will present the EigenTrust algorithm that is used to compute reputation of peers in a peer-to-peer network. The principal formula of the EigenTrust algorithm is similar to the formula used for computing the *rank* of a web page in PageRank. The difference with the EigenTrust algorithm is that it does not rely on a central server and uses trust propagation between peers instead of collecting web pages links using a web crawler.

5.1.2.2 The EigenTrust Algorithm

In (42), Hector Garcia-Molina et al. have presented the EigenTrust algorithm that permits a peer to use trust propagation in order to compute the global reputation of peers in a peer-to-peer system. In EigenTrust, a peer stores locally its trust values towards its acquaintances (peers that it has interacted with). Each local

trust value c_{ij}, computed by P_i for an other peer P_j, is the difference between the number of direct positive feedbacks and the number of direct negative feedbacks that P_i has about P_j.

In (42), three versions of the EigenTrust algorithm have been presented :

- The basic version of the EigenTrust algorithm is called BasicEigenTrust. It is basic because it ignores the distributed nature of peer-to-peer networks : it is supposed that some central server knows all the c_{ij} values and performs the peers's reputation computation.

  In this version of the EigenTrust algorithm, the global reputation of a peer P_j is obtained by iterating the process of weighting the local trust values c_{ij} assigned to P_j by other peers P_i, by using the global reputations of the P_is. The final result of the peers's reputation computation is a vector t that contains the global reputation values of all the peers in the network. In other terms, $t[k]$ represents the global reputation of the peer P_k. We describe how the vector t is computed in the following.

  The BasicEigenTrust algorithm
  Given e the m-vector representing a uniform probability distribution over all m peers in the network; C^T the transposed matrix of the matrix C of values c_{ij} :

  1. t^0 is initialized to be equal to e
  2. repeat $t^{k+1} = C^T t^k$ until t converges

  It has been proven that if C is irreducible and aperiodic, then the BasicEigenTrust algorithm converges.

- An other version of the EigenTrust algorithm is called DistributedEigenTrust. In this version of EigenTrust, the distributed nature of peer-to-peer networks is taken into account : no central server is used to collect all the c_{ij} values, i.e. they are stored locally in each peer P_i assigning c_{ij} to a peer P_j. In addition, all the peers in the network cooperate to compute and

store the global trust vector. This version of EigenTrust can also handle the problem of malicious peers : it prevents those peers from cooperating to get high global reputation by exaggerating their local trust values towards their accomplices, and giving low local trust values to all other peers.

In order to neutralize malicious peers, it is supposed that there are some peers in the network that are pre-trusted: they are trusted by all peers. Let $\mathcal{P}$ be the set of these pre-trusted peers, a distribution vector p is defined over $\mathcal{P}$, such that $p_i = \frac{1}{|\mathcal{P}|}$ if the peer $P_i \in \mathcal{P}$; $p_i = 0$ otherwise.

- This vector p is used in place of the start vector e in the BasicEigenTrust algorithm. In the presence of malicious peers, it has been proven, by using the start vector p, the algorithm converges faster.
- In addition, the main formula in the BasicEigenTrust algorithm is also modified into $t^{k+1} = (1-a)C^T t^k + ap$, where $0 < a < 1$. This new formula means that a peer has a probability a to interact with one of the pre-trusted peers, avoiding the effect of collaborative malicious peers.

The above formula is similar to the formula of the PageRank algorithm. However, the DistributedEigenTrust algorithm does not rely on any central server to compute the vector t. Every peer P_i will compute **locally** its own global reputation value, using the formula: $t_i^{k+1} = (1-a)(c_{1i}t_1^k + ... + c_{mi}t_m^k) + ap_i$. The peers communicate with each other to compute its global reputation value simultaneously.

Given a peer P_i, let A_i be the set of peers which have downloaded files from P_i, let B_i be the set of peers from which P_i has downloaded files. The DistributedEigenTrust algorithm that is executed in every peer P_i is as follows :

The DistributedEigenTrust algorithm

1. query all peers $P_j \in A_i$ for $t_j^0 = p_j$

2. repeat
   compute $t_i^{k+1} = (1-a)(c_{1i}t_1^k + ... + c_{mi}t_m^k) + ap_i$
   send $c_{ij}t_i^{k+1}$ to all peers $P_j \in B_i$
   compute the difference between t_i^{k+1} and t_i^k
   wait for all peers $P_j \in A_i$ to return $c_{ji}t_j^{k+1}$
   until t_i converges

- The last version of the EigenTrust algorithm is called SecureEigenTrust. This version solves a possible problem in the DistributedEigenTrust version, where a peer computes and stores its own global reputation value. With this setting, a malicious peer can easily manipulate its reputation value, report false trust values and thus subvert the system.

  The SecureEigenTrust version proposes a solution to this problem. In this version, the global reputation value of a peer will not be computed by itself, but by its *score managers*. A score manager of a peer P_i is another peer that computes and stores the global reputation value of P_i. To assign score managers to a peer P_i, a distributed hash table (DHT) such as Chord (62) is used. The unique identifier of P_i, such as its IP and TCP port is hashed into a point in the DHT hash space. The DHT hash space is a logical coordination space that is partitioned dynamically among the peers in the system such that every peer covers a region. Once the unique identifier of P_i is hashed, the peers that cover the corresponding region in the DHT hash space will become P_i's score managers.

  With this new setting, when a peer needs to know the reputation value of an other peer, it will compute the location of that peer's score managers and query all of them for the value it is searching. A majority vote will be used in case of significant differences in the reported values from the score managers. This vote helps to avoid using a faulty reputation value reported by a malicious peers.

The EigentTrust algorithm differentiates from the PageRank algorithm only by taking into account the distributed nature of peer-to-peer networks. The probabilistic interpretation of the EigentTrust algorithm's outputs is also similar to

that of the PageRank algorithm. For completeness, Richardson et al (48) use the DistributedEigenTrust algorithm to compute the reputation of statements (logical assertions) on the semantic web. Nedjl et al (51) have proposed to personalize the DistributedEigenTrust version in order to compute trust value of peers instead of their reputations. This method will be presented in the section 5.2.2.1.

5.1.2.3 Maximum likelihood estimation technique

In (32), a different probabilistic way to compute the reputation of peers has been proposed. It is based on the technique of maximum likelihood estimation. In this work, the reputation of a peer P_j is considered as its innate probability μ_j of performing honestly in its interactions with other peers.

Assume that peer P_j has interacted with peers P_1, P_2, ... P_n. Let x_1, x_2, ..., x_n be the feedbacks about P_j that the peers P_1, P_2, ... P_n have made in the corresponding transactions. The value of each feedback is either 1 denoting a positive feedback, or 0 denoting a negative feedback. These feedbacks are stored using the P-Grid structure (9).

Now suppose that a peer P_i wants to know the reputation of the peer P_j. It will search the P-Grid structure for the feedbacks about P_j. It is interesting to notice that when a peer P_k among P_1, P_2, ... P_n makes a feedback about the behavior of P_j, it may lie with a probability l_k. When retrieving a feedback Y_k made by P_k about P_j, the probability that P_i observes the value y_k is computed using:

$$Pr[Y_k = y_k] = \begin{cases} l_k(1-\mu_j) + (1-l_k)\mu_j & \text{if } y_k = 1 \\ l_k\mu_j + (1-l_k)(1-\mu_j) & \text{if } y_k = 0 \end{cases}$$

Given a random sample of independent reports y_1, y_2, ..., y_n, the likelihood function of this sample is: $L(\mu_j) = P[Y_1 = y_1]P[Y_2 = y_2]...P[Y_n = y_n]$
The maximum likelihood estimation procedure amounts to find a value of μ_j that maximizes the likelihood function. This number is the maximum likelihood estimate of the unknown probability, that is considered as the global reputation value of the peer P_j.

Although the value μ_j is estimated by a peer P_i, its value has a global meaning, not depending on the peer P_i. It is because every peer P_i computing μ_j is supposed to trust its own feedback (i.e. to put 1 as the value of $P[Y_i = y_i]$). Therefore, μ_j is depended on the feedbacks that have been made about P_j by all the peers.

5.2 Models of trust

In this section, we present some representative computation models of trust. In these models, the output does not express the aggregated viewpoint of a community towards a target individual B, but expresses the point of view of a given individual A computing these outputs. However, to the best of our knowledge, there is no approach that uses only the direct experiences of A about B to compute the trust of A for B. Actually, they also use witnesses's experiences about B but the set of witnesses is dependent on A.

5.2.1 Non probabilistic models of trust

In online societies

One of the first works that treated the problem of trust in online societies is (58). This work has proposed a trust model for gregarious societies, i.e. where people tend to gather into groups. The trust for an individual b belonging to a group B, when computed by an individual a belonging to a group A, is the result of an intuitive aggregation of all the knowledge of A about B. This aggregation combines (1) the local trust of a for b, $R(a, b)$, (2) the trust of a for the group B, $R(a, B)$, (3) the trust of the group A for b, $R(A, b)$ and (4) the trust of the group A for the group B, $R(A, B)$. These values are defined as follows:

1. $R(a, b)$ is the sum of each a's feedback about b weighted by a time-dependent function that gives higher values for more recent feedbacks. A feedback can take a value among {-1,0,+1}, corresponding respectively to negative, neutral and positive feedbacks.

2. $R(a, B)$ is the sum of local trust of a towards each agent b_i of B.

3. $R(A, b)$ is the sum of local trust of each a_i of A for b, weighted by the importance of each a_i in A.

4. $R(A, B)$ is the sum of local trust of each a_i of A for each b_i of B, weighted by the importance of each a_i in A.

The aggregation of these four values to compute the trust of a for b is made in an ad-hoc way:
$trust(a, b) = R(a, b)\,\xi(a, b) + R(a, B)\,\xi(a, B) + R(A, b)\,\xi(A, b) + R(A, B)\,\xi(A, B)$
where the $\xi(.,.)$ are weights reflecting the importance of each source of opinion and are chosen such that $\xi(a, b) + \xi(a, B) + \xi(A, b) + \xi(A, B) = 1$.

Although an individual a in A, when computing its trust for an individual b in B, has to take into account the feedbacks of the other individuals in A, the output trust value is still relative to a and different from the trust value of b computed by another individual in A. The factor (1) warrants this property. Moreover, due to the factors (2) and (3), it is easy to see that if an individual a' belongs to a different group A', the trust of b computed by a is different to the one computed by a'.

Massa et al (47) also proposed an ad-hoc way for an individual to compute the trust towards others in an online social network. It is supposed that each individual u has a local trust value for each of the people he knows, for example $localTrust(u, v)$. Such a local trust has a value between 0 and 1. Now, let a peer P be the individual that computes its trust towards others, the following notations are used:

- $users[0] = P$
- $users[1]$ = the set of users that P knows
- for $i = 2, ...k$; $users[i]$ = the set of users known by $users[i-1]$ and have not yet been in any $users[j]$, $j = 0, .., i-1$

By using the following algorithm, P can compute its trust value for the users that are in $users[i]$, $i = 1, ..., k$. The trust value for a user in $users[i]$ depends

only on the local trust of $users[i-1]$ for him, weighted by the trust of P for the $users[i-1]$.

Algorithm used in (47)

$i = 0$
$trust(P) = 1$
while $(i \leq k)$
  i ++
  for each individual u in $users[i]$
    $trust(u) = \frac{\Sigma_{v=pred(u)}(trust(v)*localTrust(v,u))}{\Sigma_{v=pred(u)}(trust(v))}$

In the above algorithm, $pred(u)$ denotes the set of predecessors of u, for each user v among them $trust(v) \geq 0.6$. It means that when computing its trust value for a user u in $users[i]$, P uses the local trust value of a user v in $pred(u)$ towards u only if P's trust value for v is greater than 0.6. This choice is intuitive and ad-hoc. Moreover, the meaning of the trust values computed by P for other peers is not clear.

In the Semantic Web

A simple formula to compute trust of users in the FOAF (4) network has been proposed in (34). In the FOAF network, each user is identified by his email address. Users store in their profile a list of users that they know, called their acquaintances. (34) extends the profile of each user, permitting for a user to state its trust towards each of its acquaintances. This local trust takes a value from 1 to 9, represents different level of trust, from 'Distrust absolutely' to 'Trust absolutely'.

In (34), the formula that is used by a user i to compute its trust towards a user s that it does not know directly makes use of the local trust value of the users along the paths from i to s. Suppose that i has n neighbors with paths to s, this formula is as follows:

$$t_{is} = \frac{\Sigma_{j=0}^{n} \begin{cases} (t_{js} * t_{ij}) & \text{if } t_{ij} >= t_{js} \\ (t_{ij}^2) & \text{if } t_{ij} < t_{js} \end{cases}}{\Sigma_{j=0}^{n} t_{ij}}$$

The above formula is hard to interpret. One of its properties is that for all the users j in the path from i to s, t_{is} will be less than t_{ij}.

In peer-to-peer setting

The work in (30) has proposed an approach based on votes for choosing resources to download in Gnutella-like P2P networks. Recall that in Gnutella-like systems, when a peer P searches for a resource, it broadcasts a Query message containing the search keywords. to all its neighbors. Neighbors of P forward this Query message to theirs neighbors that have not yet been asked. By this way, the Query message is broadcasted in the network, creating searching branches rooted in the peer P. When a peer receives a Query message for which it has a 'match' (it has the requested resource), it responds with a QueryHit message that will be transmitted back to P along the corresponding searching branch.

It is supposed in this approach that every peer, as well as every resource, has a unique identifier (ID). Each peer keeps a binary feedback database towards resources that it has downloaded, and a binary feedback database towards the peers having provided with resources to it. When a peer P receives a list of resources for its search, it chooses one resource among them that it thinks most satisfying its request. When choosing a resource, P also knows the other peers that offer the same resource. Before downloading, P would like to know which one is the best and if the resource is of good quality. For this, it asks other peers to vote on the resource and the providers in question, by broadcasting a Poll message on the Gnutella network. When receiving the Poll message, a peer checks its feedback databases to answer the vote request of P. After reception of the votes, P observes the results and freely decide whether to trust the resource in question and chooses the best provider in his point of view.

It is easy to see that the approach used in this work is very intuitive and ad-hoc. Moreover, even if P asks all the other peers in the network to vote, the final judgment to trust a resource and a provider or not is only up to P.

5.2.2 Probabilistic models of trust

We present in this section the most representative models of trust whose outputs have a probabilistic meaning. We will discuss briefly a work that adapts the EigenTrust algorithm (presented in details in section 5.1.2.2) then we compare those that have adopted the Bayesian approach to statistics to build their trust model.

5.2.2.1 Personalized EigenTrust algorithm for trust computation

Nedjl et al (51) have proposed a slight modification for the DistributedEigenTrust version of the EigenTrust algorithm. In the DistributedEigenTrust algorithm, it is supposed that there is a set of pre-trusted peers that have globally good reputation. The computation of reputation of peers in a peer-to-peer network makes use of this set of pre-trusted peers. In (51), a peer can choose among the set of pre-trusted peers a subset that it trusts the most. The computation of trust values that takes place at each peer will make use of this peer's choice of pre-trusted peers.

With this modification, when the DistributedEigenTrust algorithm converges, we obtain *personalized global reputation values* of peers. The personalized global reputation of a peer A, that is computed by a peer B, expresses the trust that B has for A, given B's own point of view when choosing the peers to be pre-trusted. If B had chosen an other subset of pre-trusted peers then the personalized global reputation value of A would have been different.

5.2.2.2 Bayesian approach to statistics for trust estimation

In a peer-to-peer (or multi agents) network, one of the roles of every peer is to provide services to others. A service may be a resource (movies, documents, ...) for other peers to download, or simply a recommendation to peers that will provide the service in question. In this context, it is often supposed that for each transaction, a peer either behaves correctly by providing a service of good quality, or misbehaves if its service is bad.

It is interesting to define the quality criteria of a service. In fact, the criteria to determine the quality of a service should only be decided by every peer receiving this service, because every peer is not necessarily as indulgent as another. Consequently, for a given peer P_k, each other peer P_i thinks differently about P_k's willingness of behaving correctly. Let us denote this willingness expressed in P_i's point of view by θ_{ik}, $0 \leq \theta_{ik} \leq 1$.

In reality, such a parameter θ_{ik} is unknown to P_i. The Bayesian approach to statistics can be used in order for P_i to estimate this parameter. This approach consists mainly in two steps :

1. P_i assigns to θ_{ik} a supposed initial value. This value is often 0.5, expressing the fact that when P_i has no interaction with P_k yet, it thinks that there is 50% chance that P_k will behave correctly.

2. After observing the performances of P_k in some transactions, P_i recomputes this parameter, using the Bayes formula (39). The performance of P_k in each of these transactions is recorded by P_i in the form of positive and negative feedbacks, corresponding respectively to good and bad services made by P_k.

We deliberately do not detail this approach right now, but we will do it later in the next chapter. In the following, we present some works that have adopted the Bayesian approach to statistics to build models of trust in peer-to-peer and mobile ad-hoc networks.

The works in (22; 23) are situated in the context of mobile ad-hoc networks. The goal of the trust model in (22; 23) is to permit every peer to choose the most trustworthy peers, according to its viewpoint, in order to determine a routing path for packets of information. A special point in the model of (22; 23) is that when used to compute trust, less recent feedbacks have less weight than recent ones so that they have less impact in the estimated trust.

Some other works that have used the Bayesian approach to statistics are (49; 66). In all these works, apart from its direct feedbacks, an agent may also use feedbacks of third party about the agent for which trust is being computed. Using third

party's feedbacks may be appropriate when the agent estimating its trust towards another agent has no, or enough, direct experiences with the latter.

Aggregation the feedbacks from different sources is a difficult problem and leads easily to an ah-hoc formulation. In our point of view, (66) proposed the most sophisticated method in order to combine these feedbacks. First, the trust computing agent P_i calculates the probability that a feedback provider P_j will provide an accurate feedback. This is done based on the trust in the recommendation service of P_i towards P_j. Second, based on this probability, P_i reduces the distance between P_j's opinion (expressed via its feedbacks) on the target agent P_k and the prior belief that P_i has on P_k. Once all the opinions collected have been adjusted in this way, P_i has a collection of positive and negative feedbacks about P_k. The feedbacks are aggregated using the same method as for estimating trust, i.e. Bayesian approach to statistics to estimate θ_{ik}.

Another problem, in case P_i misses direct experiences about P_k, is to choose a set of agents S to which P_i asks for their feedbacks about P_k. In (49), S is the set of agents on the path from P_i to P_k. In (22), S is the set of neighbors of P_i. In most cases, the set S is determined by P_i, making the estimated value θ_{ik} relative on P_i. However, if S is the set of all agents, such as the case of (41) then this value θ_{ik} is the same for all peers P_i. (41) assumes the existence of a central server to collect the feedbacks about all peers. However, such an assumption is not appropriate considering the distributed nature of peer-to-peer or multi agents systems.

5.3 Summary

Trust and reputation modeling is an intensively studied subject for online social networks, peer-to-peer and multi agents systems. Although trust and reputation are closely related, in some context it is appropriate to distinguish them.

Several trust and reputation models have been designed. They differ on the interpretation of their outputs : a clear mathematical semantics for probabilistic approaches, as opposed to ad-hoc models. Some algorithms may be used to compute either trust or reputation, depending on the way feedbacks are used,

Comparing factors	Approaches
Reputation models	Ebay(3); Amazon(2); (10); EigenTrust (42); Maximum likelihood technique (32); PageRank (21; 54); Bayesian approach (work (41))
Trust models	personalized EigenTrust (51); (58); (30); Bayesian approach (works (22; 23; 49; 66)); (47)
Probabilistic models	PageRank (21; 54), EigenTrust (42), Maximum likelihood technique (32), Bayesian approach
Non-probabilistic models	Ebay(3), Amazon(2), (10)

Table 5.1: Classification of the works presented in this chapter

such as the EigenTrust algorithm or the Bayesian approach to statistics technique. Table 5.1 summarizes and compares the models that we have presented.

In the models that we have presented, feedbacks are collected in two ways. The first one is to use a central server to store all feedbacks. The second one is to permit agents to request other agents for their feedbacks, and receive these feedbacks either by using a DHT-based or by using a Gnutella-based routing of messages communication. In both ways, an agent has first to determine the target agent for which feedbacks are needed, then to ask for the feedbacks concerning the target agent. We summarize the possible ways for feedbacks collection in Table 5.2.

Comparing factors	Approaches
Use of central server	Ebay(3); Amazon(2); PageRank (21; 54); personalized EigenTrust (51); Bayesian approach (work (41))
Gnutella-based routing	(58) ; (30) ; (47); Bayesian approach (works (22; 23; 49; 66))
DHT-based routing	(10) ; EigenTrust (42) ; Maximum likelihood technique (32)

Table 5.2: Summary on the possible ways for collection of feedbacks

Outline of our approach

In the next chapter, we will present our **probabilistic** model of **trust** for semantic peer-to-peer systems. We have adopted the Bayesian approach to statistics in order to build a trust model whose outputs are mathematically founded.

Our model has two important points with respect to the models presented in the section 5.2.2.2. Firstly, our model takes into account the notion of context for trust. In fact, a peer may be trustworthy in some contexts, but untrustworthy in others. This applies for every peer evaluating the trustworthiness of that peer. In our model, a context is a set of concept names, used by one (or some) peer(s), justifying a resource when it is returned as an answer of a search query. Secondly, when applying the model of trust into the SomeOWL semantic peer-to-peer system (presented in Section 2.3), we propose a new strategy to propagate feedbacks. This strategy is called **greedy**, permitting to collect feedbacks concerning answers of a search **during** the search process. Relevant feedbacks for each answer of a search are available at the querying peer **as soon as** the answer is returned. This permits the peer to use these feedbacks immediately, saving time for it by avoiding to wait for answers first then to ask for feedbacks after.

Chapter 6

A PROBABILISTIC TRUST MODEL FOR SEMANTIC PEER-TO-PEER SYSTEMS

In a semantic peer-to-peer system, each peer is free to use its own ontology to annotate the resources that it stores locally and that it agrees to share with others. In such a context, no user imposes to others his own ontology but logical mappings between ontologies make possible the creation of a network of people in which personalized semantic marking up of data cohabits nicely with a collaborative exchange of data. The mappings are exploited during information retrieval or query answering for query reformulation between peers.

Semantic peer-to-peer systems provide a support for the emergence of open and decentralized electronic social networks, in which no central or external authority can control the reliability of the participating peers. Some of the peers may have (accidentally or deliberately) annotated some (or all) of the resources they have in an inappropriate way or resource content may evolve over time in such a way that prior annotations are not suitable anymore. As a consequence, a user of such semantic peer-to-peer systems is not always satisfied with the answers returned to his queries. The proposal of an adequate model to assess the level of confidence that a peer may have in a given answer is thus an important issue.

We have studied and designed a model that can respond to that need. Our model

is based on the Bayesian approach to statistics and focuses on trust (as opposed to reputation), because in a decentralized context different peers may correspond to different points of view. Our model differs from other works that have adopted the same Bayesian approach by the following points.

Firstly, we argue for a finer grained context sensitive approach to trust. Actually, in semantic peer-to-peer systems, the result of a query is a set of resources (e.g., documents) that have been semantically annotated by one or several peers. We consider that such semantic annotations constitute some form of logical justifications for the returned resources. Therefore, instead of defining trust at the level of peers, we handle trust at the level of semantic annotations of resources. The trust in resources justified by some annotation L, from the viewpoint of a peer P , is defined as the probability that a resource annotated by L satisfies P. This trust can be estimated using the Bayesian approach to statistics from the evolving number of resources annotated by L and observed by P, that satisfy or do not satisfy it.

Secondly, our model computes trust by using direct experiences between peers. When this information is not available or not sufficient, we propose two simple mechanisms of trust propagation in order for a peer to take advantage of the experiences of others. One of these mechanism is designed especially for the SomeOWL semantic peer-to-peer system.

6.1 Preliminary : some useful notions of probability and statistics

Because our model trust is based on the Bayesian approach to statistics we first briefly recall in this section some basic notions of probability and statistics that will be used in the following.

Random Variables

Consider an experiment for which the set of all possible outcomes is denoted by S. A random variable X is a real-valued function that is defined on S, assigning a real

number $X(s)$ to each possible outcome $s \in S$. For example, in the experiment of tossing a pair coin, all the possible outcomes are 'head', 'tail'. It can be modeled by the random variable defined by $X(head) = 1$ and $X(tail) = 0$. A random variable is said to be a discrete random variable if its set of possible values is countable. Otherwise it is said to be a continuous random variable.

Discrete random variables and probability distribution

A discrete random variable X has an associated *probability distribution*, defined as a function $f : \mathbb{R} \mapsto [0, 1]$ which specifies the probability of X taking each of its different possible values: $f(x) = Pr(X = x)$. If x is not a possible value for X, then $f(x) = 0$. If $\Sigma = \{x_1, x_2, ...\}$ denotes the set of all possible values for the variable X, the probability distribution f verifies the property :

$$\sum_{x \in \Sigma} f(x) = 1$$

Example : A random variable X that can take only possible values either 0 or 1 is called a **Bernoulli random variable**. If p denotes the probability $Pr(X = 1)$ (and thus $Pr(X = 0) = 1 - p$), the associated probability distribution of X is a **Bernoulli distribution** with parameter p defined as :
$f_p(x) = \begin{cases} p^x(1-p)^{1-x} & \text{for} \quad x = 0, 1, \\ 0 & \text{otherwise.} \end{cases}$

Continuous random variables and probability density function

In contrast with the discrete case, a continuous random variable X can take any value in an interval, a semi-interval, or in the whole set $\mathbb{R}$. The continuous distribution of X can be characterized by its associated *probability density function* $f : \mathbb{R} \mapsto [0, 1]$. This function can be used to model the probability of X being in some interval $[a, b]$:

$$Pr(a \leq X \leq b) = \int_a^b f(x).dx$$

The probability density function f must satisfy two the properties : $\int_{-\infty}^{+\infty} f(x).dx = 1$ and $f(x) \geq 0$, for all x.

Example : The **beta distribution** is a continuous probability distribution for random variables defined on the interval $[0, 1]$. It has two positive parameters α and β. The probability density function $f_{\alpha,\beta}$ of the beta distribution is defined by :

$$f_{\alpha,\beta}(x) = \begin{cases} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} & \text{for} \quad 0 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

where $\Gamma(x) = \int_{-\infty}^{x} t^{x-1}e^{-t}dt$. The Γ function may be viewed as a generalization of the factorial function, since for positive integers we have $\Gamma(n) = (n-1)!$.
It may be noticed that in the particular case where both α and β are equal to 1, the beta distribution corresponds to the uniform distribution.

Independent random variables

It is said that n random variables $X_1, ..., X_n$ are independent if, for every n sets $A_1, ..., A_n$ of real numbers, $Pr(X_1 \in A_1, X_2 \in A_2, \ldots, X_n \in A_n) = Pr(X_1 \in A_1).Pr(X_2 \in A_2).\ldots.Pr(X_n \in A_n)$.

Expectation of a random variable

Given a random variable X, the **expected value** (or mean value) of X, denoted by $E(X)$ characterizes, if it exists, the average value that may be expected from repeated random trials for X. In the case where X is a discrete random variable with possible values in Σ and f is its probability distribution, it is defined as : $E(X) = \sum_x xf(x)$ if $\sum_x |x|f(x) < \infty$ If X is a continuous random variable, the probability density function of which is f, it is defined by : $E(X) = \displaystyle\int_{-\infty}^{\infty} xf(x)dx$ if $\displaystyle\int_{-\infty}^{\infty} |x|f(x)dx < \infty$.

Example : the expected value of a random variable X that follows the beta distribution with parameters α and β is $E(X) = \frac{\alpha}{\alpha+\beta}$

Variance and standard derivation of a random variable

Suppose that X is a random variable with mean $\mu = E(X)$. The variance of X, $Var(X)$ is defined as : $Var(X) = E[(X-\mu)^2]$. The variance of a distribution

provides a measure of dispersion of the distribution around its mean μ. If the variance is small, the probability distribution is tightly concentrated around μ. If the variance is large, the probability distribution spreads widely around μ. The standard derivation is the root of the variance.

Example : the variance of a random variable X that follows the beta distribution with parameters α and β is $Var(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Conditional probability

Given two jointly distributed random variables X and Y, the conditional probability distribution of Y given X , written $Pr(Y|X)$, is the probability distribution of Y when X is known to be a particular value. For discrete random variables, the conditional probability function is defined by :

$$f_{Y|X}(y|x) = Pr(Y = y|X = x) = \frac{Pr(X=x \wedge Y=y)}{Pr(X=x)} \text{if} Pr(X = x) \neq 0$$

For continuous random variables, a similar definition may be proposed at the level of probability density functions.

Principle of the Bayesian approach

Conditional probabilities are extensively used in the context of the Bayesian approaches, to revise probability values as new evidence concerning the variables under consideration is obtained. If the variable of interest is Y and if the new obtained information can be modeled with what is known on some variable X, it is possible to update the information on Y, i.e. its so called *prior probability* $Pr(Y = y)$. The new information is taken into account by considering the *posterior probability* of Y, defined as $Pr(Y = y|X = x)$.

Bayesian inference principle consists in deriving the posterior probability of a random variable Y from its prior and its likelihood $Pr(X = x|Y = y)$, using the Bayes theorem:

$$Pr(Y = y|X = x) = \frac{Pr(X = x|Y = y).Pr(Y = y)}{Pr(X = x)}$$

A particular case where the principle of Bayesian inference can be used is the estimation of the value of the parameters of some probability distributions. In such a case, we are concerned by some random variable, known to follow some given probability distribution, but the parameters of which are unknown. In the Bayesian approach, such parameters are modeled as random variables. Possible initial knowledge on these parameters is then expressed by means of prior probability distributions. In the absence of any particular knowledge, uniform distribution can be used as prior distributions. From the data that may be observed and collected, one can then extract a random sample of values corresponding to similar random variables as the one under consideration (i.e. which have the same distribution). This information may then be used for computing a posterior distribution of the parameters being estimated.

Illustration : Let us consider a Bernoulli random variable X known to have a Bernoulli distribution with unknown parameter p. This parameter is modeled as a continuous random variable on $0 \leq p \leq 1$. In such a case, it is convenient to use a beta distribution to model the distribution of this random variable. As a matter of fact, the following theorem states that if the prior distribution of p is a beta distribution of parameter α and β, its posterior distribution after n independent observations, is itself a beta distribution whose parameters can be computed from α and β. Another convenient property of the beta distribution is that when $\alpha = \beta = 1$, it corresponds to a uniform distribution. It thus can be used with such values to model the absence of information in the prior.

Theorem 7 *Suppose that $X_1, ..., X_n$ form a random sample from a Bernoulli distribution, the parameter θ of which is unknown ($0 \leq \theta \leq 1$). If the prior distribution of θ is a beta distribution with given parameters α and β ($\alpha > 0$ and $\beta > 0$) the posterior distribution of θ given the $X_i = x_i (i = 1, \ldots, n)$ is a beta distribution with parameters $\alpha + \sum_{i=1}^{n} x_i$ and $\beta + \sum_{i=1}^{n} (1 - x_i)$.*

6.2 Local observations for trust estimation

In a semantic peer-to-peer systems each peer interacts with other peers of the network. The level of trust of a peer into the information provided by other peers may clearly be influenced by the quality of previous good or bad interactions with those peers. Memorizing the quality of such past interactions is thus essential in order to account for the evolution of trust over time. In this section we present the assumptions that we have made when designing our trust model and illustrate these assumptions by an example.

We assume that each resource r returned as an answer to some query is associated with a label $L(r) = C_1 \dots C_n$ corresponding to its logical justification (semantic annotation). $L(r)$ is a set of classes of the vocabularies of (possibly different) peers known to annotate the resource r and supposed to characterize a sufficient condition for r to be an answer. Any other resource annotated in the same way is thus equally supposed to be an alternative answer to the query. We also assume that the classes used in labels are independent in the sense that for any two classes of a justification, none of them is a subclass of the other. This assumption means that for a returned answer, the only classes that appear in its justifications are those corresponding to most specific classes of the network.

Finally we assume that the user, when querying a peer P_i , is randomly asked to evaluate some of the returned resources as *satisfying* or *not satisfying*. Each evaluation on a returned resource is an *observation* on its label. All the labels that have been observed so far by the user are stored in a local *observation database* O_i at P_i . An observation database is a (relational) table, every row of which has 3 fields, that correspond respectively to the name of a label (L), the number of satisfying observations on this label ($\#_i^+(L)$) and the number of non-satisfying observations on this label ($\#_i^-(L)$).

Note that, because labels correspond to sufficient conditions for resources being returned, any observation on a label L' such that $L \subseteq L'$ is also an observation that satisfies the condition L and therefore may be considered as relevant for L.

Definition 17 *Number of satisfying/non-satisfying observations*

Label (L)	$\#_1^+(L)$	$\#_1^-(L)$
$P_2 : MyActionFilms$	30	6
$P_2 : MyCartoons$	3	15
$P_4 : Western$	2	8
$P_5 : Italian \ \ P_4 : Western$	0	6
$P_6 : AnimalsDocum$	14	14
$P_7 : JeanRenoir$	22	11
$P_3 : Bollywood$	6	35

Table 6.1: Summary of Peter's observations at P_1

Let $\mathcal{O}_i$ be the observation database of a peer P_i , L be a label in $\mathcal{O}_i$ and $Rel(L)$ denote the set of label L' in $\mathcal{O}_i$ such that $L \subseteq L'$. The numbers of satisfying and non-satisfying observations made by P_i that are relevant to L are respectively denoted by:

$O_i^+(L) = \sum_{L' \in Rel(L)} \#_i^+(L')$
$O_i^-(L) = \sum_{L' \in Rel(L)} \#_i^-(L')$

These two numbers summarize the past experience of the peer P_i relevant to the label L, i.e. the evaluated resources justified by at least the classes of L.

For instance, suppose that Peter is the user querying the peer P_1 of a semantic peer-to-peer system. After a number of resources have been evaluated, Peters past experience may be summarized as in Table 6.1.

Among all the resources evaluated by Peter and semantically annotated with the class $MyActionFilms$ of the peer P_2 , 30 have been considered as satisfactory and 6 as not satisfactory. For the same peer P_2 , only 3 out of 18 evaluated resources tagged by $MyCartoons$ were positive. Similarly, all evaluated resources annotated with $Italian$ by P_5 and $Western$ by P_4 obtained negative feedbacks. Especially, the numbers of satisfying and non-satisfying observations relevant to $P_4 : Western$ are respectively 2 and 14, computed from those of $P_5 : Italian$ $P_4 : Western$ and $P_4 : Western$ itself.

From this observation database, one can clearly understand that for the next queries, given this past experience, Peter would intuitively prefer (or more trust) answers classified as $P_2 : MyActionFilms$ to those annotated by $P_2 : MyCartoons$ or by $P_3 : Bollywood$. This is because the proportion of positive observations obtained so far is higher for $P_2 : MyActionFilms$ than for the others.

Each peer P_i can progressively update its observation database $\mathcal{O}_i$, as new resources are evaluated, and refine the trust it has towards resources justified by the different observed labels. The level of trust on a label can vary according to the observations made on it so far.

6.3 Bayesian model and estimation of trust

In this section, we present a model of trust which takes into account the observations on the labels in the observation database of a peer to estimate the trust of that peer towards these labels.
Given a label L, let X_{iL} be the binary random variable defined on the set of resources annotated by L as follows:

$$X_{iL}(r) = \begin{cases} 1 & \text{if the resource } r \text{ is satisfying for } P_i \\ 0 & \text{otherwise} \end{cases}$$

We define the trust of a peer P_i towards a label L as the probability that the random variable X_{iL} is equal to 1, given the observations resulting from the past experience of the peer P_i .

Definition 18 *Trust of a peer towards a label*
Let $\mathcal{O}_i$ be the observation database of a peer P_i and L be a label, the trust $T(P_i, L)$ of P_i towards L is defined as follows:

$$T(P_i, L) = Pr(X_{iL} = 1|\mathcal{O}_i)$$

The following theorem provides a way to estimate the trust $T(P_i, L)$ and the associated error of estimation.

Theorem 8 *Let $\mathcal{O}_i$ be the observation database of a peer P_i and L be a label. When $O_i^+(L)$ satisfying and $O_i^-(L)$ non-satisfying observations relevant to L have been performed, $T(P_i, L)$ can be estimated to*

$$\frac{1 + O_i^+(L)}{2 + O_i^+(L) + O_i^-(L)}$$

with a standard deviation of

$$\sqrt{\frac{(1 + O_i^+(L)) \times (1 + O_i^-(L))}{(2 + O_i^+(L) + O_i^-(L))^2 \times (3 + O_i^+(L) + O_i^-(L))}}$$

Proof:
In our setting, the random variable X_{iL} has been defined in such a way that it follows a Bernoulli distribution the parameter of which is precisely the probability of satisfaction of P_i by resources annotated with L, i.e. $T(P_i, L)$ which is to be estimated.

In the Bayesian approach to statistics, such unknown parameter is modeled as a random variable. In our context, it is understandable to take a neutral viewpoint on $T(P_i, L)$ when P_i does not have any experience on L. We thus set the prior distribution of $T(P_i, L)$ to be the uniform distribution, which corresponds to the beta distribution of parameters 1 and 1. Now, given o_1, ..., o_n the n observations in $\mathcal{O}_i$ relevant to L, we can compute the posterior distribution of $T(P_i, L)$.

Because, peers are randomly asked to evaluate some of the resources they have obtained, the n observation o_1, ..., o_n may be considered as independent (the results of each observation does not influence the result of any other observation). and constitute a random sample from a Bernoulli distribution.
Therefore, Theorem 7 can be applied in our setting. We thus deduce that the posterior distribution of $T(P_i, L)$ given the observations o_1, ..., o_n is also a beta distribution with parameters $1 + \sum_{i=1}^{n} o_i = 1 + O_i^+(L)$ and $1 + (n - \sum_{i=1}^{n} o_i) = 1 + O_i^-(L)$.

Therefore, $T(P_i, L)$ can be estimated as the mean of that beta distribution which is $\frac{1+O_i^+(L)}{2+O_i^+(L)+O_i^-(L)}$.

Label (L)	$T(P_1, L)$	Standard deviation
$P_2 : MyActionFilms$	0.815	0.062
$P_2 : MyCartoons$	0.2	0.087
$P_4 : Western$	0.166	0.085
$P_5 : Italian$ $P_4 : Western$	0.125	0.11
$P_6 : AnimalsDocum$	0.5	0.089
$P_7 : JeanRenoir$	0.657	0.079
$P_3 : Bollywood$	0.162	0.055

Table 6.2: Estimated trust of P_1 towards the labels of Table 6.1

The precision of the estimate is the standard deviation around this value, which is $\sqrt{\frac{(1+O_i^+(L))\times(1+O_i^-(L))}{(2+O_i^+(L)+O_i^-(L))^2\times(3+O_i^+(L)+O_i^-(L))}}$ □.

By applying Theorem 8 to the observation database summarized in Table 6.1, we can characterize the estimations for the trust of the peer P_1 towards the labels observed so far by P_1. Table 6.2 describes the corresponding estimations with their associated standard deviation.

The model presented so far relies on direct interactions between peers. Trust towards a label is estimated based on the numbers of satisfying and non-satisfying observations relevant to that label. The standard deviation expresses the precision of the estimation. Note that the number of observation required to reach a given level of precision may easily be deduced from the formula characterizing the standard deviation. For instance, it can be shown that the standard deviation is less than 0.1 with as few as 22 direct observations between peers.

However, when very few or no observations relevant to the label under consideration have been performed, the estimated trust may be imprecise. In such a case, several choices are possible. One may either keep a strict point of view, and trust only labels on the basis of the direct experiences. Note that in the case where there is strictly no relevant observation, the estimate is 1/2, which reflects a neutral point of view. One may also, as often in real life, take some

advice from more experienced peers. In that case, solicited peers' observations may prove useful to compensate for the lack of local observations, provided that they use similar evaluation criteria.

Instead of propagating trust between peers, our approach consists in propagating the pairs of numbers used for computing trust. Propagating two numbers instead of one does not represent a significant overhead. Yet, it has the significant advantage of providing a well-founded way to compute a joint trust using the same Bayesian model. Instead of using an ad-hoc aggregation function for combining local coefficients of trust, the numbers $O^+_{i1}(L) \dots O^+_{il}(L)$ (respectively $O^-_{i1}(L) \dots O^-_{il}(L)$) coming from solicited peers $P_{i1} \dots P_{il}$ are cumulated to compute the joint trust towards L of the peers $P_{i1} \dots P_{il}$ by applying the formula of Theorem 8.

A simple strategy to propagate the observations to compute trust from the solicited peers consists in waiting for getting some answer justified by a label L, then check if the local observation database contains enough observations relevant to L and if this is not the case, ask one or several trusted neighbors for their numbers $O^+(L)$ and $O^-(L)$. Since it applies after obtaining the answers, such a strategy can be used as a post-precessing and does not require to change the query evaluation mechanism. As a consequence, it can be applied to different kinds of semantic peer-to-peer systems, provided they are able to associate answers to semantic annotations (e.g. labels). Furthermore, if enough local observations have been performed, it does not cost anything. For this reason, we call this approach the **lazy strategy**

6.4 Application to SomeOWL

In this section, we apply this model in SomeOWL (presented in Section 2.3). In particular, we present an alternative to the lazy strategy for observation propagation, called the **greedy strategy**. This strategy has been specifically designed for SomeOWL and can be implemented with a slight adaptation of SomeOWL.

SomeOWL has been deployed over the SomeWhere platform. The data stored at each peer are documents which are identified by Uniform Resource Identifiers (URIs). Each peer categorizes its documents by declaring the corresponding URIs as instances of one or several classes of its own taxonomy of classes, which also serves as a query interface (for other peers or users). Queries specify the classes of interest for another peer or a user. The answers are the documents whose URI is declared or inferred as an instance of those classes. The point is that some answers are stored in distant peers that must be retrieved and queried in their own vocabulary. Mappings between taxonomies using different names for their classes may require the initial query to be split and reformulated.

Query answering in SomeOWL is a two step process. The first step is the rewriting of the query, which returns a set of conjunctions of extensional classes (possibly from different peers) that logically entail the query. The second step consists then in fetching from the relevant peers the documents labeled by those classes. Therefore, the answers returned by SomeOWL are associated by construction to the rewriting whose evaluation has produced them, which corresponds exactly to the label required in our model of trust.

In order to deploy our trust model in SomeOWL , we have to equip each peer with a local database, for the accounting of evaluated answers, according to their label (i.e. the rewriting that produced them). From that local database, it is straightforward to plug the local computation of trust.

We also have to implement a strategy for getting additional observations from other peers when needed. Implementing the lazy strategy can easily be achieved by adding two new messages for asking and sending the required numbers $O^+(L)$ and $O^-(L)$ from trusted peers, after getting an answer labeled by L. The drawbacks of that strategy are that (1) each peer has to manage a list of trusted peers, (2) it requires peers to exchange additional messages after obtaining the answer and its label. It is however possible to avoid these points, using an alternative strategy.

The main idea of the greedy strategy is to encapsulate the information needed for trust computation into the messages that are exchanged during query processing.

With this approach, peers involved in the processing of a query, send back not only their respective answers, but also all the number of observations that they have performed on labels participating to the computation of the trust on the final label.

SomeOWL query rewriting is performed by the DeCA algorithm, through a propositional encoding of peer ontologies (presented in Section 2.3.2). With this encoding, a rewriting of a query in SomeOWL is found by DeCA in the form of a clause, the literals of which correspond to the classes of the rewriting. Each label thus corresponds to a clause and we assume that this is taken into account in the local observation database.

Because of the split/recombine strategy used by DeCA , during query rewriting, a clause that is sent back from one peer to an other may be later recombined with other consequents and eventually, constitute only sub-clause of a final consequent, returned as an answer to the query. The problem is that the final consequent clause is only known at the very end, after the last recombination. From the point of view of observations propagation , valuable observations are only those relevant to the final consequent. But since this consequent is not known at this stage the only thing that can be done is to propagate observations relevant to clauses that could correspond to such a final consequent. Therefore, when sending back a clause c from a peer, we propagate as well all the observations available at that peer that are relevant to clauses c' that are subsumed by c.

We have modified the message passing version of DeCA (presented in Section 2.2) in order, during query processing, to collect the appropriate observations on the peers involved in the reasoning. Intuitively, answers are returned as pairs of the form $(c\ data(c))$ where $data(c)$ is a small database containing only the observations relevant to clauses subsumed by c that have been collected so far on the peers that have contributed to the production of the clause c.

- In the ReceiveForthMessage procedure, when a peer P_i receives a query, it processes the query locally by the *Resolvent* procedure. We adapt this procedure so that each time a back message is returned, it contains

not only consequent clauses c but the pair $(c\ data(c))$ where $data(c) = \text{SELECT}(c, \mathcal{O}_i)$, defined as the set of rows in the local database $\mathcal{O}_i$ corresponding to clauses that are subsumed by c. Moreover, during the initialization of cache structures $CONS(l, hist)$ for every shared literel l of c, the pair $(l\ Select(l, \{\mathcal{O}_i\}))$ is added to the cache instead of just l.

- In the ReceiveBackMessage procedure, a peer P_i handles an answer message from a distant peer P_j containing a consequent clause r_k, of a literal l_k that has been asked to P_k. The consequent is returned as a pair $(r_k\ \ data(r_k))$ where $data(r_k)$ is the set of observations relevant to r_k, and collected during the production of r_k.

  This pair is first added to the cache $CONS(l_k, hist)$ where $hist$ is the history context of the query l_k. The literal l_k comes from a clause c of the form $L(c) \vee l_1 \vee \ldots \vee l_p$ where $L(c)$ and $S(c) = l_1 \vee \ldots \vee l_p$ corresponds to the subparts of c containing respectively the non-shared and shared literals of c, l_k being one of the shared literals of $S(c)$. The new obtained answer r_k has then to be recombined with those corresponding to the respective consequents of the l_j, for $(j \neq k)$. This can be performed by extending the disjunction operator $\vee$ so that it can operate on couples of the form $(r\ data(r))$, to produce a new couple of the same form, in the following way :

  $(r_1\ data_1) \vee (r_2\ data_2) \ldots (r_n\ data_n)$
  $= (r_1 \vee r_2 \vee \ldots \vee r_n\ \text{SELECT}((r_1 \vee r_2 \vee \ldots \vee r_n, data_1 \cup data_2 \cup \ldots \cup data_n)))$.

  This amounts to saying that when combining observations coming from different sources, we keep among observations from all the different sources, only those that are relevant to the recombined clause.

  In practice, we here have to recombine the local part of $L(c)$ with the respective consequents r_j of the literals l_j, for $j \neq k$. We thus retain the local observations $\text{SELECT}(L(c), \mathcal{O}_i)$ with those coming from the $data(r_j)$ associated respectively to each r_j.

  Eventually, the whole set of consequents obtained by recombination and their respective sets of relevant observations may be characterized by:
  $\oslash_{l \in S(c) \setminus \{l_k\}} \text{CONS}(l, hist) \oslash ((L(c)\ \text{SELECT}(L(c), \mathcal{O}_i)) \oslash (r_k\ data(r_k)))$.

This adaptation of DeCA permits a peer to collect the number of satisfying and the number of non-satisfying observations on the labels that are relevant to the label L of an answer. This happens during the query processing process. These information is used to compute the numbers $O^+(L)$ and $O^-(L)$ that are used in the formula as presented in Theorem 8. The set of peers that provide these numbers are those that have contributed to obtain a rewriting. Those peers may thus be considered as naturally relevant for obtaining appropriate observations. This set of peers are determined at query time and may vary according to the query and the returned rewritings for the query.

The main advantage of this procedure is that no extra-processing is needed as results are obtained. This can be particularly interesting for a new peer that has recently joined the semantic peer-to-peer network but still has few direct experience with other peers. On the other hand, the overhead due to the extra information that is encapsulated in the messages, may greatly vary in size, according to the size of the clause returned, as well as the size of the observation databases of the other peers.

6.5 Discussion and Conclusion

We have presented an approach for modeling trust in the context of semantic peer-to-peer systems. It allows each peer to evaluate the trust it has towards resources depending on their semantic annotations. Because it relies on few assumptions, it is applicable to a wide range of semantic peer-to-peer systems. The granularity of the trust considered here is at the level of semantic annotations of resources. Defining a more abstract peer-based notion of trust is however straightforward since it would just amount to aggregate the available information concerning all the classes of a same other peer.

The model that we have presented leads to a simple formula making easy and fast for a peer to compute trust. The output of the model has a mathematical meaning. In addition to the simplicity of the formula of Theorem 8 for computing trust, the advantage of the model is that it provides also a simple way to compute the minimum number of experiences that are required for guaranteeing a good

precision of the estimation of trust. Using our model, by defining a threshold for the precision of the estimation of trust, a peer can judge if an estimated trust is precise enough. If it is not the case, the peer has to collect observations from other peers to have a better trust estimation towards the label in question. We have presented a simple strategy for propagating observations between peers that can be applied to any kind of semantic peer-to-peer systems. For SomeOWL particularly, the peer may use the greedy strategy to collect faster observations possibly relevant for computing trust.

An other advantage of our model is that trust evolves depending on the knowledge of a peer on a given label. When the observation database in a peer is updated, the new information is taken into account in this way: the observations that were in the database are used to determine the parameters of the prior distribution of trust, and the new observations are used to compute the posterior distribution of trust.

One of the assumptions that we have made for the aggregation using the formula in Theorem 8 is that the peers use similar criteria while evaluating resources. Although that assumption is not realistic, it is important to note that third party's observations are only used for a peer to form his initial belief about a label. After having enough direct experiences, the peer uses only its experiences to estimate trust.

Finally, even if our model does not take explicitly into account malicious peers, we discuss how it copes with them. A peer can be malicious by providing to other peers virus-affected resources, or by simply lying when reporting its observations about others. It is usually assumed that malicious peers are a minority in a semantic peer-to-peer network. Then, when a peer downloads a virus-contaminated resource, based on the semantic annotation of the resource, it can know the peers that have provided it with this resource and identify them as malicious peers. A more difficult problem is dealing with liar peers that transmit wrong observations on other peers. By construction, our model favors direct observations, therefore avoiding liar peers when having enough direct feedbacks. When it is not the case,

a way to limit the influence of liar peers is to use a majority vote and to discard the peers the observations of which differ much from the majority.

Chapter 7

Conclusion and perspectives

In this thesis, we have contributed to improve the reliability of answers in semantic peer-to-peer networks in different ways.

We have first considered the problem of returning only well-founded answers of a query when the global network is inconsistent. Our approach consists in detecting, in a fully decentralized setting, all the causes of inconsistencies at each addition of a new mapping. These causes are characterized as nogoods (sets of mappings) and are stored in a distributed manner. At query time, soundness of query answering is guaranteed by avoiding these causes. Our algorithms have been implemented in the SomeWhere platform, where the problem of handling of inconsistency had not been considered before. We have adapted this platform for experimenting our algorithms, and we have obtained promising results. The contributions of this part have been published in (27).

We have also considered the problem of modeling and handling peers' trust into returned answers. In our model, trust has been defined at the level of semantic annotations of answers, which are combinations of peer classes justifying them. Based on a mathematical setting, this model has a clear semantics and trust towards answers can be estimated by peers with a good precision using few observations. When applying the model in the SomeOWL semantic peer-to-peer network, we have designed a strategy for observation propagation, that is more practical than the strategy usually used in the literature. While this model remains to be implemented, its scalability seems guaranteed by the simplicity of its

computational model. The contributions of this part has been published in (52).

Some of the perspectives of this thesis are the followings.

Firstly, in our approach, the soundness of query answering relies on the completeness of the inconsistency detection. Currently, for guaranteeing the answers' soundness, we separate query answering from the detection of inconsistencies : a mapping can only be used in a reasoning after the completion of the detection of inconsistencies that it may create. A possible improvement is to allow sound usage of this mapping in a reasoning before the completion of this detection. A possible way to do this is to adapt the inconsistency detection algorithm, so that, the completion of this process, for successive reasoning depths, is notified. The soundness of an answer obtained with a mapping support of size k is then guaranteed if such a notification has been obtained for the depth k.

Secondly, in its current state, our system only guarantee to return well-founded answers. In particular, we can obtain both A and $\neg A$ as consequences, as long as each of them can be deduced from a (different) consistent subset of the inconsistent global theory. In such a case, a natural question for a user is to know if there are good reasons to prefer one consequence over the other. It would be useful to make explicit logical arguments in favor of one of them. A possible solution is to exploit further the information gathered during the consequence finding process, i.e. the set of mapping supports and the set of associated nogoods. By this, we can build an argumentative approach for setting preferences on all well-founded consequences that are produced. For example, we can consider the length of proofs (mapping supports) and prefer the consequences supported by small ones. We can also consider the diversity of mapping supports, or the way they overlap with the corresponding sets of nogoods. A numerical approach for determining the preferred consequences can also be considered. One possibility could be to investigate an extension of our trust model to other observation criteria. For instance, it could be based on sets of mapping support.

A third perspective would be to exploit the values of trust that are computed for different purposes.

- For discovering new mappings in a semantic peer-to-peer network: if the combination of classes of different peers turns out to get high values of trust, it suggests that these classes could be close semantically. Potential mappings between these classes can thus be established.

- For the routing of answers. We can imagine that a peer would like to provide fast and more answers to peers that it trusts, and to limit its interaction with those it does not trust. This routing strategy can be easily implemented since our model of trust can be applied at the level of peers: a given peer can thus distinguish among the other peers those that it trusts the most. Then, it can set priorities in query answering threads to favor them. It can also make a list of peers to whom it avoids sending answers, and from whom it avoids receiving answers.

- For the routing of queries. This perspective is dual to the previous one. We can imagine that peers would like to query directly peers that trust them, in order for their queries to be treated fast, and to receive more answers. For this, the current model can be slightly adapted, so that a peer can know those that trust it. Then, it establishes direct mappings with them and can thus query them directly.

Finally, our model of trust could be extended in order to integrate time management. As a matter of fact, old observations may not still be relevant for trust estimation, neither for discovering new mappings, as discussed above. We can define a time-based function for the model, that gives less weight to old observations, and more weight to recent ones. The simplest way is to forget observations older than a predefined date, and to consider only more recent ones.

References

[1] Advogato. http://www.advogato.org/person. Cited in page(s) 120

[2] Amazon. http://www.amazon.com. Cited in page(s) 116, 132

[3] Ebay. http://www.ebay.com. Cited in page(s) 116, 132

[4] FOAF. http://www.foaf-project.org. Cited in page(s) 127

[5] Gnutella. http://www.gnutella.com. Cited in page(s) 2

[6] Google. http://www.google.com. Cited in page(s) 118

[7] RDFS. http://www.w3.org/TR/rdf-schema. Cited in page(s) 6

[8] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. Van Pelt. GridVine: Building internet-scale semantic overlay networks. In *International Semantic Web Conference (ISWC)*, 2004. Cited in page(s) 6

[9] Karl Aberer. P-Grid: A self-organizing access structure for P2P information systems. *International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science*, **2172** : pages 179–194, 2001. Cited in page(s) 2, 117, 124

[10] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, pages 310–317. ACM Press, 2001. Cited in page(s) 117, 132

[11] P. Adjiman. *Peer-to-peer reasoning in propositional logic: algortihms, scalability study and applications.* PhD Thesis, University Paris Sud 11, 2006. Cited in page(s) 24

[12] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Scalability study of peer-to-peer consequence finding. In *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2005. Cited in page(s) 5, 14

[13] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a P2P setting: Application to the semantic web. *Journal of Artificial Intelligence Research (JAIR)*, 2006. Cited in page(s) 4, 5, 6, 10, 14, 15, 31, 98

[14] P. Adjiman, François Goasdoué, and Marie-Christine Rousset. Somerdfs in the semantic web. *Journal on Data Semantics VIII, Volume 4380/2007, Lecture Notes in Computer Science*, pages 158-181, 2007. Cited in page(s) 6

[15] Martin Allen. Preservationist logics and nonmonotonic logic. In *Proceedings of the Student Session, First North American Summer School in Logic, Language, and Information (NASSLLI), Stanford University*, 2002. Cited in page(s) 52

[16] Martin Allen and R.E.Jennings. Resource-bounded reasoning and paraconsistency. In *Logical Consequence: Rival Approaches, Proceedings of Conference of the Society for Exact Philosophy (SEP)*, September 1999. Cited in page(s) 53

[17] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Symposium on Principles of Databse Systems. ACM Press*, pages 68–79, 1999. Cited in page(s) 41

[18] Ofer Arieli, Marc Denecker, Bert Van Nuffelen, and Maurice Bruynooghe. Repairing inconsistent databases: A model-theoretic approach and abductive reasoning. In Hendrik Decker, Jrgen Villadsen,

and Toshiharu Waragai, editors, *Paraconsistent Computational Logic*, **95** of *Datalogiske Skrifter*, pages 51–65. Roskilde University, Roskilde, Denmark, 2002. Cited in page(s) 43

[19] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. Cited in page(s) 6

[20] Alex Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, **1**:2003, 2003. Cited in page(s) 50, 51

[21] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, **30**(1–7): pages 107–117, 1998. Cited in page(s) 118, 132

[22] S. Buchegger and J. Le Boudec. The effect of rumor spreading in reputation systems for mobile ad-hoc networks, 2003. In *Proceedings, Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, 2003. Cited in page(s) 130, 131, 132

[23] Sonja Buchegger. A robust reputation system for p2p and mobile ad-hoc networks. *Second Workshop on the Economics of Peer-to-Peer Systems (P2PEcon), Harvard University*, 2004. Cited in page(s) 130, 132

[24] Bziau. What is paraconsistent logic? *Frontiers of Paraconsistent logic. Baldock Research Studies Press*, 2000. Cited in page(s) 47

[25] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency tolerance in p2p data integration: an epistemic logic approach. In *Proceedings of the 10th International Workshop on Database Programming Languages (DBPL 2005)*, pages 90–105, 2005. Cited in page(s) 57

[26] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency tolerance in p2p data integration: An epistemic logic approach. *Information Systems, Volume 33, Issues 4-5, June-July 2008, Pages 360-384* , 2008. Cited in page(s) 57

[27] Philippe Chatalic, Gia Hien Nguyen, and Marie-Christine Rousset. Reasoning with inconsistencies in propositional peer-to-peer inference systems. In *Proceedings of The 17th European Conference on Artificial Intelligence, (ECAI06)*, pages 352–356, 2006. Cited in page(s) 154

[28] Laurence Cholvy. A general framework for reasoning about contradictory information and some of its applications. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, pages 233–263, 1998. Cited in page(s) 59

[29] Laurence Cholvy and Christophe Garion. Querying several conflicting databases. *In Journal of Applied Non-Classical Logics, Herms Lavoisier, vol 14(3), 2004.* Cited in page(s) 42, 60

[30] E. Damiani, S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation–based approach for choosing reliable resources in peer–to–peer networks, 2002. In 9th ACM Conference on Computer and Communications Security. Cited in page(s) 128, 132

[31] Sandra de Amo, Walter A.Carnielli, and Joao Marcos. A logical framework for integrating inconsistent information in multiple databases. In *International Symposium on Foundations of Information and Knowledge Systems (FoIKS), Lecture Notes in Computer Science 2284*, pages 67-84, 2002. Cited in page(s) 49

[32] Zoran Despotovic and Karl Aberer. Maximum likelihood estimation of peers' performance in p2p networks. In *The Second Workshop on the Economics of Peer-to-Peer Systems*, 2004. Cited in page(s) 124, 132

[33] D.S.Scott. Lectures on a mathematical theory of computation. *Theoretical foundations of Programming Methodology. Reidel Publication, pages 145-292*, 1982. Cited in page(s) 54

[34] Jennifer Golbeck, Bijan Parsia, and James Hendler. Trust networks on the semantic web. In *Proceedings of Cooperative Information Agents 2003*, Helsinki, Finland, August 27–29, 2003. Cited in page(s) 127

[35] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *International Conference on Data Engineering (ICDE)*, 2003. Cited in page(s) 3, 4

[36] A. Halevy, Z. Ives, I. Tatarinov, and Peter Mork. Piazza: data management infrastructure for semantic web applications. In *World Wide Web Conference (WWW)*, 2003. Cited in page(s) 3, 4

[37] Alon Y. Halevy. *Logic-based artificial intelligence*, chapter Logic-based techniques in data integration, pages 575–595. Kluwer Academic Publishers, 2000. Cited in page(s) 4

[38] P. Hasse and G. Qi. An analysis of approaches to resolving inconsistencies in dl-based ontologies. *In Proceedings of International Workshop on Ontology Dynamics (IWOD)*, 2007. Cited in page(s) 44

[39] Morris H.DeGroot and Mark J.Schervish. *Probability and Statistics*. Addison Wesley, 2002. Cited in page(s) 130

[40] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005. Cited in page(s) 40, 55, 56, 57, 62, 67, 103, 104

[41] Audun Josang and Roslan Ismail. The beta reputation system. *In Proceedings of the 15th Bled Electronic Commerce Conference*, 2002. Cited in page(s) 131, 132

[42] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks, 2003. In *Proceedings of the Twelfth International World Wide Web Conference*. Cited in page(s) 120, 121, 132

[43] Micheal Kifer and Eliezer L.Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning 9(2)*, 1992. Cited in page(s) 47

[44] Hector J. Levesque and Gerhard Lakemeyer. *The logic of knowledge bases*. MIT Press, Cambridge, MA, USA, 2000. Cited in page(s) 58

[45] Andrei Lopatenko and Leopoldo Bertossi. Consistent query answering by minimal-size repairs. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 558–562, Washington, DC, USA, 2006. IEEE Computer Society. Cited in page(s) 41

[46] J. Madhavan and A. Halevy. Composing mappings among data sources. In *International Conference on Very Large Data Bases (VLDB)*, 2003. Cited in page(s) 4

[47] Paolo Massa and Paolo Avesani. Controversial users demand local trust metrics: An experimental study on epinions.com community. In *Proceedings of the National Conference on Artificial Intelligence of the USA, (AAAI)*, pages 121–126, 2005. Cited in page(s) 126, 127, 132

[48] R. Matthew, R. Agrawal, and P. Domingos. Trust management for the semantic web, 2003. Proceedings of the Second International Semantic Web Conference, Sanibel Island, Florida. Cited in page(s) 124

[49] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. 35th Hawaii International Conference on System Science (HICSS), 2002. Cited in page(s) 130, 131, 132

[50] W. Nedjl, B. Wolf, C. Qu, S. Decker, M. Sintek, and al. Edutella: a p2p networking infrastructure based on rdf. In *World Wide Web Conference (WWW)*, 2002. Cited in page(s) 5

[51] Wolfgang Nejdl, Paul Alex, Ru Chirita, Mario Schlosser, and Oana Scurtu. Personalized reputation management in p2p networks. In *Workshop on Trust, Security, and Reputation on the Semantic Web (ISWC)*, 2004. Cited in page(s) 124, 129, 132

[52] G. H. Nguyen, P. Chatalic, and M. C. Rousset. A probabilistic trust model for semantic peer to peer systems. In *DaMaP '08: Proceedings of the 2008 international workshop on Data management in peer-to-peer systems*, pages 59–65, New York, NY, USA, 2008. ACM. Cited in page(s) 155

[53] Cai nicolas Ziegler and Georg Lausen. Spreading activation models for trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service*. IEEE Computer Society Press, 2004. Cited in page(s) 120

[54] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998. Cited in page(s) 118, 132

[55] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In *Proceedings of the 14th international conference on World Wide Web*, pages 633–640, New York, NY, USA, 2005. ACM. Cited in page(s) 44

[56] Guilin Qi, Weiru Liu, and David Bell. A revision-based approach to handling inconsistency in description logics. *Artificial Intelligence Review*, **26**(1-2):115–128, 2006. Cited in page(s) 44

[57] W. Qi, G.; Liu and D.A Bell. Knowledge base revision in description logics. In *Proceedings of 10th European Conference on Logics in Artificial Intelligence (JELIA'06)*. Springer Verlag, 2006. Cited in page(s) 44

[58] J. Sabater and C. Sierra. Regret: A reputation model for gregarious societies, 2000. In Research Report. Institut d'Investigacio i Intelligencia Artificial. Cited in page(s) 125, 132

[59] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003. Cited in page(s) 44, 67

[60] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003. Cited in page(s) 40

[61] Luciano Serafini, Alex Borgida, and Andrei Tamilin. Aspects of distributed and modular ontology reasoning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2003. Cited in page(s) 50, 51

[62] I. Stoica, R. Morris, D. Karger, M.F. Kaasshoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Conference on applications, technologies, archtecture and protocols for computer communications*, 2001. Cited in page(s) 2, 6, 123

[63] H. Stuckenschmidt and M. Klein. Integrity and change in modular ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 900–905, Acapulco, Mexico, 2003. Morgan Kaufmann.*, 2003. Cited in page(s) 60

[64] V.S. Subrahmanian and Lela Amgoud. A general framework for reasoning about inconsistency. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2007. Cited in page(s) 38, 54, 55, 56, 62, 67, 103, 104

[65] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer data management systems. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 539 - 550, 2004. Cited in page(s) 4

[66] W. T. Teacy, Jigar Patel, Nicholas R. Jennings, and Michael Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, **12**(2):183–198, 2006. Cited in page(s) 130, 131, 132